EDITORA

CIP — Catalogação na Publicação

004.4

C837d Costa, Ronald Emerson Scherolt da.

Desenvolvedor Web / [Ronald Emerson Scherolt da Costa, Salvador Alves de Melo Júnior] ; organização de Joscélia Moreira de Azevedo.

Brasília : Editora IFB, 2022.
613 p. : il.

ISBN 978-85-64124-92-9

1. Informática. 2. Programação. 3. Webdesing. 4. Desenvolvimento de software. 5. Web. I. Melo Júnior, Salvador Alves de. II. Azevedo, Joscélia Moreira de. IV. Título.

Elaborado com os dados fornecidos pela editora, sob a responsabilidade da bibliotecária Laysse N. B. Teixeira — CRB/1ª 2727.

EDITORA IFB

REITORIA - Setor de Autarquias Sul, Qd 02, Bloco E - Edifício Siderbrás.
CEP: 70070-020 Asa Sul - Brasília/DF
www.ifb.edu.br
Fone: +55(61)21032112
editora@ifb.edu.br

Sumário

INTRODUÇÃO A INFORMÁTICA

Sumário

LÓGICA DE PROGRAMAÇÃO

Sumário

FUNDAMENTOS DE WEBDESIGN

Sumário

FUNDAMENTOS DE WEBDESIGN

Apresentação

Querido leitor e querida leitora.

É um prazer para nós saber que está dedicando seu precioso tempo para a leitura desse e-book. Mas, antes que você inicie essa jornada, podemos te contar um pouco de como foi a produção desse material?

Esse e-book é o resultado da compilação de material elaborado para o curso Técnico em Informática, ofertado pelo Instituto Federal de Educação, Ciência e Tecnologia de Brasília.

Esse curso, realizado na modalidade a distância, por meio de fomento da bolsa-formação do PRONATEC, foi ofertado entre os anos de 2017 e 2019 e a elaboração do material prezou por uma linguagem dialogada com estudantes que estavam do outro lado da tela do computador ou do celular.

Por isso, te convidamos a fazer essa leitura aproveitando a linguagem dialogada e sentindo-se parte da caminhada que fizemos com nossos estudantes.

Ele está organizado da seguinte maneira: no capítulo 1, veremos, Introdução a Informática, no capítulo 2, Lógica de Programação e, no capítulo 3, Fundamentos de Webdesign .

Desejamos que esse e-book seja uma jornada de aprendizagem para você!

E, além de aproveitar um pouquinho do material de nossos cursos, quem sabe não te incentivamos a vir nos conhecer melhor e estudar conosco, não importa se na modalidade presencial ou a distância!

Boa leitura!

Introdução à Informática

Ronald Emerson Scherolt da Costa

Unidade 1

Histórico e Evolução da Informática

E então, estudante? Vamos começar a compreender como as tecnologias atuais evoluíram? Que tal conhecer alguns marcos históricos da área de Tecnologia da Informação?

Ao término desta unidade esperamos atingir o seguinte objetivo:

- Conhecer os fundamentos tecnológicos e científicos de informática.

Nesta unidade, denominada ***Histórico e evolução da informática***, vamos observar o histórico do desenvolvimento da informática e das tecnologias ligadas à computação em geral. Teremos a oportunidade de compreender como os computadores passaram por grandes modificações e tornaram-se cada vez mais rápidos e úteis.

Em nosso cotidiano empregamos tecnologias em quase tudo que utilizamos, desde um simples celular a computadores bem mais potentes. Não é intrigante compreender como essas modificações ocorreram ao longo do tempo?

Não é simples determinar o ponto de partida para uma síntese histórica da computação, uma vez que existem muitos trabalhos e que diversas descobertas levaram à construção dos primeiros computadores. Desde o tempo antigo, a humanidade usou ferramentas para realizar cálculos, armazenar e processar informações. A primeira ferramenta usada para contar e ao mesmo tempo representar as quantidades contadas eram os dedos, dando origem ao sistema de numeração decimal. Nesta unidade, vamos ter a oportunidade de dialogar e refletir sobre como todo esse processo de evolução da tecnologia ocorreu e como chegamos até às tecnologias atuais. Antes de começar a conhecer a história e evolução dos computadores, vamos compreender alguns termos importantes que estarão presentes em nossas atividades.

Saiba mais!

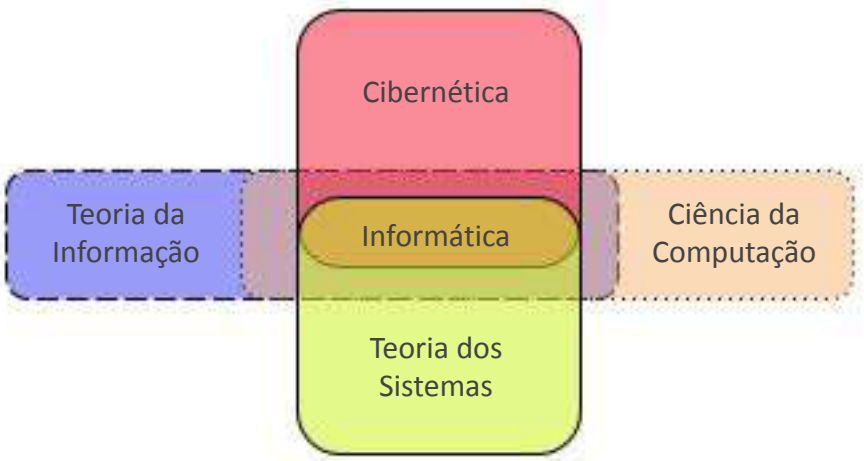
Para conhecer mais sobre o surgimento dos sistemas de numeração e aspectos interessantes da história da matemática visite o site: http://www.matematica.br/historia/index_h_tempo.html


Você sabe o que significa ciência da computação? E informática? E Tecnologia da Informação? E computador?
Hummm... Quantos conceitos!
Vamos refletir...

Vamos começar conversando sobre esses conceitos importantes. Velloso (2011) indica que a **informática** pode ser compreendida "com a informação automática, isto é, o tratamento da informação de modo automático". Logo, podemos compreender a **informática como o uso dos computadores eletrônicos no trato da informação**. Esse neologismo foi criado na França (em francês, *informatique*) em 1996 e depois popularizou-se para o inglês como *informatic*. Para entender a história dos computadores e sua evolução, precisamos compreender como as designações de seu emprego surgiram ao longo do tempo.

A informática situa-se na interseção de quatro áreas do conhecimento: da **ciência da computação**, a qual cuida do processamento de dados, abrangendo a arquitetura das máquinas e as respectivas engenharias de software, isto é, sua programação; da **ciência da informação**, a qual tem seu foco no tratamento da informação desde a busca, o armazenamento, a gestão e a disseminação; da **teoria dos sistemas**, a qual busca a solução de problemas por meio da conjugação dos elementos capazes de levar a objetivos pretendidos; e da **cibernética**, a qual tem por objetivo a busca da eficácia, por meio de ações ordenadas de mecanismos de automação.

Figura 1 - Informática: a interseção de quatro áreas



Fonte: Adaptado de Velloso (2011)

Quando falamos do emprego de computadores, a informática é imprescindível. É ela a responsável pelo desenvolvimento técnico e científico para coletar, tratar e disseminar dados, enquanto matéria-prima obtida por sistemas computacionais, transformando-os em informações.

Veremos nesta unidade que os computadores surgiram na década de 40, nos tempos da Segunda Guerra Mundial e, com eles, o termo processamento de dados. Com o passar do tempo, essa designação do emprego dos computadores evoluiu para a informática, quando eles se popularizaram e deixaram de ser privilégio de poucos. Assim, o termo informática passou a ser utilizado para caracterizar procedimentos diversos que se apoiam no computador.

Vamos refletir?

Os **dados** são elementos conhecidos de um problema. A **informação** é um conjunto estruturado de dados, transmitindo conhecimento.

Logo, **informática** é o uso dos computadores eletrônicos no trato da informação. Ela é utilizada para coletar, tratar e disseminar **dados**, enquanto matéria-prima obtida por sistemas computacionais, transformando-os em **informações**. Ela caracteriza procedimentos diversos que se apoiam no computador.

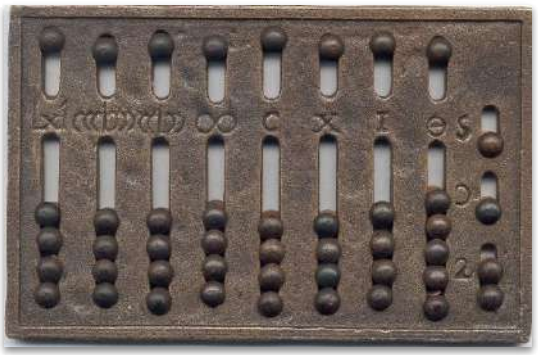
Agora que já compreendemos os conceitos iniciais para nos situarmos no contexto dos computadores, vamos iniciar nossa viagem pelo tempo. Prepare-se!

1.1 Tudo começou pela necessidade de calcular

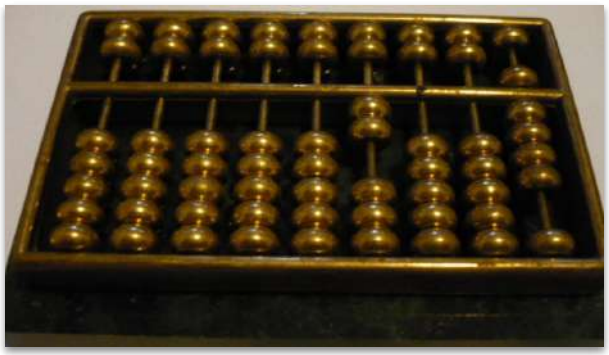
Na antiguidade, havia a necessidade de recursos para facilitar a realização de cálculos. O avançar da civilização trazia consigo novos desafios. A cada dia surgiam novas soluções para problemas mais complexos graças ao incremento dos cálculos também mais complexos. A necessidade de dar velocidade e precisão aos cálculos se tornava maior com o passar do tempo. A história registra que os homens de negócio precisavam de um modo rápido e seguro de fazer contas. Essa necessidade levou o homem antigo a inventar uma máquina simples e rudimentar que ajudava na realização de cálculos no comércio: o ábaco.

O ábaco é um dos mais antigos instrumentos de cálculo que se tem notícia. Segundo muitos historiadores, ele foi inventado na Mesopotâmia, pelo menos em sua forma mais primitiva. O ábaco foi empregado pelos romanos, japoneses e chineses que o aperfeiçoaram. Embora rudimentar e antigo, ele certamente serviu de inspiração para a evolução dos computadores atuais.

Figura 2 - Ábacos antigos



Ábaco Romano



Ábaco Chinês

Fonte: S. W. Song , 2016

As primeiras máquinas de computar que se conhece são bastante antigas e rudimentares. Vejamos, em ordem cronológica, a evolução desses incríveis dispositivos:

a) **Ossos de Napier -** inventado por John Napier (1550-1617), era um dispositivo constituído por tabelas de multiplicação gravadas em bastão, permitindo multiplicar e dividir de forma automática. Esse dispositivo evitava a memorização da tabuada, o que trouxe grande auxílio ao uso de logaritmos, em execução de operações aritméticas como multiplicações e divisões longas. Napier ficou conhecido como o decodificador do logaritmo natural (ou neperiano) e por ter popularizado o ponto decimal.

Figura 3 - Ossos de Napier

As varas ou ossos de Napier reduziram a multiplicação para uma sequência de adições simples. Este instrumento consiste em uma caixa de madeira contendo doze cilindros rotativos cada um marcado com as tabelas de multiplicação de 0 a 9. Este é o tipo de ossos de Napier ilustrados por Gaspard Schott, em 1668.

Fonte: Foto de SSPL/Getty Images

b) A máquina de **Wilhelm Schickard** (1592-1635) foi considerada o primeiro instrumento de calcular capaz de somar, subtrair, multiplicar e dividir. Por muitos anos ficou perdida e, recentemente, foi encontrada alguma documentação sobre ela, quando então passou a ser considerada a primeira máquina de calcular, desbancando **Blaise Pascal (1623-1162)**.

c) A máquina calculadora **Pascaline**, baseada em rodas dentadas, criada por **Blaise Pascal (1623-1162)**, era considerada a primeira máquina de calcular que fazia operações de soma e subtração. Sua máquina foi criada para ajudar seu pai na tarefa de computar impostos em Rouen, na França.

A máquina de Pascal foi aprimorada por um matemático alemão chamado **Gottfried Wilhelm Leibniz (1646-1726)**. O desenvolvimento do cálculo moderno, em particular o desenvolvimento da integral e da regra do produto, é creditado a **Leibniz**. Foi ele quem descreveu o primeiro sistema de numeração binário moderno (1705), tal como o sistema numérico binário utilizado nos dias de hoje.

Figura 4 - La pascaline - Calculadora de Pascal

La pascaline foi a primeira calculadora mecânica do mundo, planejada por Blaise Pascal em 1642. Era uma máquina com um engenhoso sistema de engrenagens que fazia contas de adição e subtração.

Fonte: Imagem retirada da internet. Disponível em: <https://pt.wikipedia.org/wiki/La_pascaline>. Acesso em: 01 de nov.2017.

Uma calculadora não é um dispositivo automático e necessita, consequentemente, da ação constante de um operador. Esse processo não automatizado é um obstáculo para a velocidade e a confiabilidade dos resultados. Todos esses dispositivos trabalhavam apenas com a entrada de números simples. Não havia a possibilidade de programá-los. Eles ainda estavam longe de ser um computador como os que utilizamos atualmente.

1.2 A Revolução dos cartões perfurados - máquinas programáveis

Todas as novas descobertas sempre são impulsionadas por necessidades. Durante o século XVIII, as fábricas de tecelagem desejavam implantar teares automatizados que aceitassem um meio de entrada de dados para automatizar e controlar os desenhos nos tecidos. Assim, em 1801, **Joseph Marie Jacquard (1752-1834)** desenvolveu um tear automático. Sua ideia criou a primeira máquina mecânica programável a representar os padrões em cartões de papel perfurado, que eram tratados manualmente. Sua máquina trabalhava tão bem que milhares de tecelões perderam seus empregos com esse processo de automação. Os trabalhadores se revoltaram e quase mataram o inventor.

Figura 5 - O tear automático programável

Jacquard construiu um tear automático, capaz de ler os cartões e executar as operações na sequência programada. A primeira demonstração prática do sistema aconteceu na virada do século XIX, em 1801. Os mesmos cartões perfurados de Jacquard, que mudaram a rotina da indústria têxtil, teriam, poucos anos depois, uma decisiva influência no ramo da computação. E, praticamente sem alterações, continuam a ser aplicados ainda hoje.

Fonte: Imagem retirada da internet. https://pt.wikipedia.org/wiki/Joseph-Marie_Jacquard. Acesso em: 01 de nov.2017.

O projeto de Jacquard foi tão bem sucedido que sua ideia atravessou o Canal da Mancha e serviu de inspiração para **Charles Babbage (1791-1871)**, um professor de matemática da Universidade de Cambridge.

Babbage estava tentando desenvolver uma máquina capaz de calcular polinômios por meio de diferenças, **o calculador diferencial**. Ele construiu uma máquina de calcular que utilizava cartões perfurados para controlar a forma como os cálculos eram realizados. Ele desenvolveu uma máquina para "tecer números" que conseguia fazer cálculos como funções trigonométricas e logaritmos .

Enquanto ele desenvolvia o calculador diferencial foi inspirado pela ideia de Jacquard, o que fez com que Babbage optasse por construir uma nova e mais complexa máquina, **o calculador analítico**.

O projeto de Babbage não era perfeito. Possuía desvantagens: a primeira era o fato de o seu computador ser mecânico e, a segunda, era a precariedade da engenharia da época.

Apesar das dificuldades, seu invento impressionou o governo inglês. Esta nova máquina era muito semelhante ao computador atual.

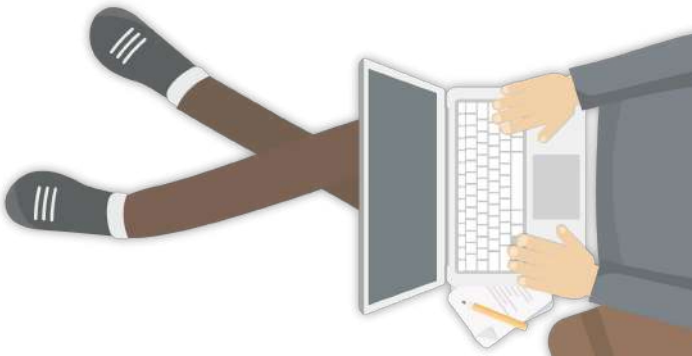
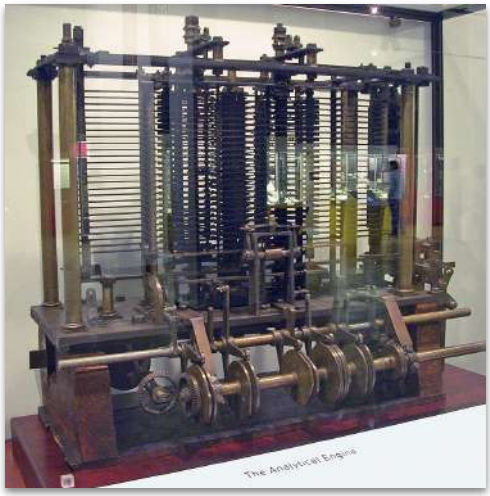


Figura 6 - A máquina diferencial e a máquina analítica

O calculador diferencial: a máquina tinha a capacidade de receber dados, processá-los, armazená-los e exibi-los. Graças a ela, Babbage ficou conhecido como o pai do computador.

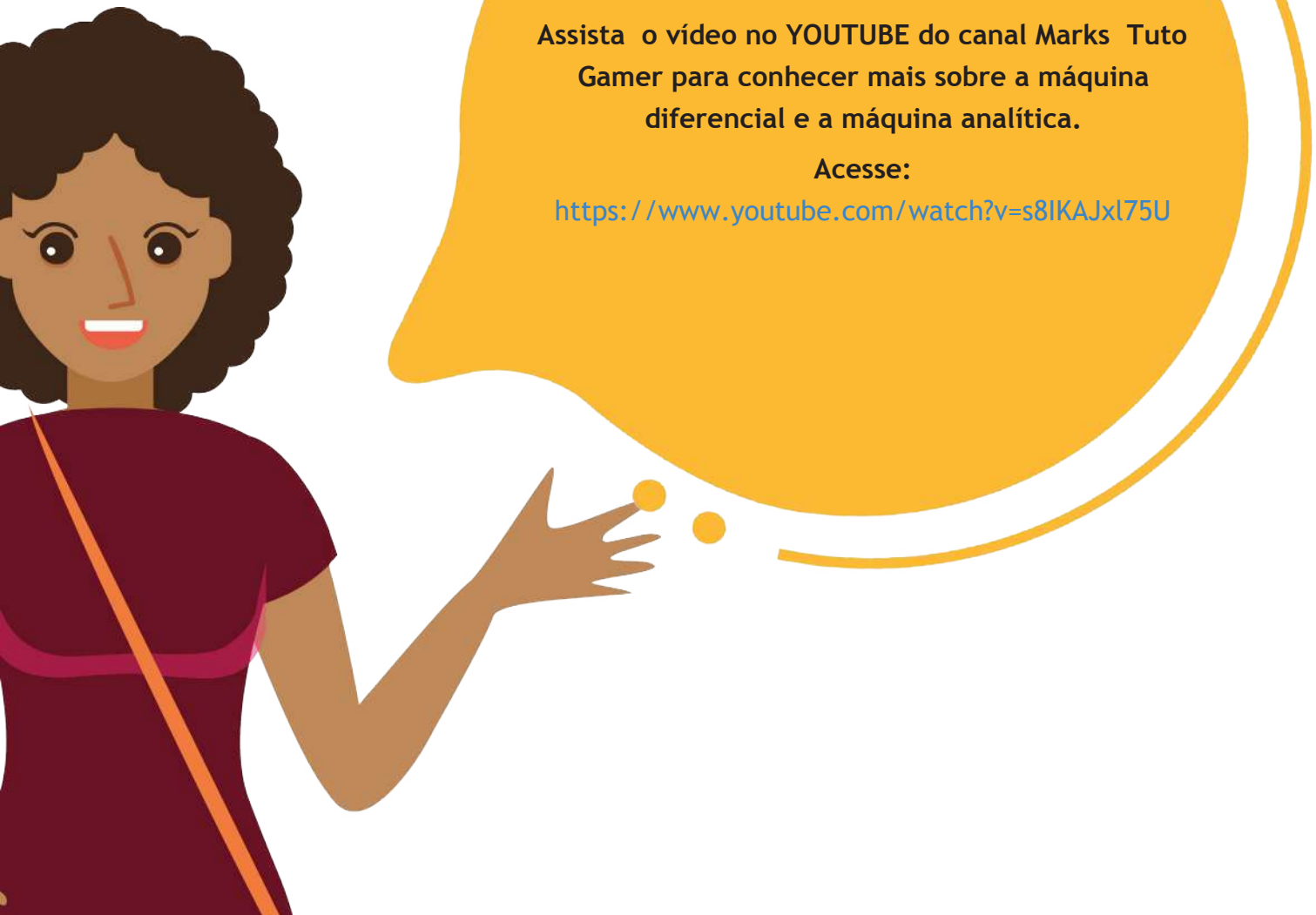
O calculador analítico: é uma máquina de uso geral, mas não ficou operacional. Tem 4 partes: armazenamento, computação, entrada e saída. Primeira programadora: Ada Lovelace.



Fonte: London Science Museum

Saiba mais!

- O ábaco segundo a Wikipédia: Disponível em: https://pt.wikipedia.org/wiki/%C3%81baco Acesso em: 01 de nov.2017.
- Veja o vídeo no YOUTUBE do canal Marks Tuto Gamer para conhecer mais sobre a história do ábaco. Acesse: https://www.youtube.com/watch?v=R3eFtQlNRXQ
- A Evolução dos Computadores: do Ábaco ao ENIAC ao Sunway. Disponível em: https://www.ime.usp.br/~song/mac412/historia2016.pdf. Acesso em: 01 de nov.2017
- Você quer entender ainda mais sobre o funcionamento da "La pascaline" inventada por Pascal? Veja uma animação que demonstra seu funcionamento.
  Acesse: https://historiaybiografias.com/primera_calculadora/
- Assista ao vídeo no YOUTUBE do canal Marks Tuto Gamer que pode ajudar a compreender a máquina de Pascal e sua história.
  Acesse: https://www.youtube.com/watch?v=wVidblCSTH0
- As varas ou ossos de Napier: Disponível em: https://www.gettyimages.com.br/detail/foto-jornal%C3%ADstica/john-napier-discoverer-of-logarithms-created-the-foto-jornal%C3%ADstica/90738490?esource=SEO_GIS_CDN_Redirect#/john-napier-discoverer-of-logarithms-created-the-popular-though-of-picture-id90738490. Acesso em: 01 de nov.2017.
- Assista ao vídeo no YOUTUBE do canal **Marks Tuto Gamer** para conhecer mais sobre a calculadora de Leibniz. Acesse: https://www.youtube.com/watch?v=lHd8mGIPZVg&t=18s



Assista o vídeo no YOUTUBE do canal Marks Tuto Gamer para conhecer mais sobre a máquina diferencial e a máquina analítica.

Acesse:

https://www.youtube.com/watch?v=s8IKAJxl75U

1.3 O surgimento da programação

A jovem **Ada Augusta** era companheira de **Babbage** e trabalhou junto com ele nos anos que se seguiram. Os dois desenvolveram vários projetos com o intuito de aperfeiçoar essa primeira calculadora. Ada era filha do poeta Lord Byron e ficou conhecida como Ada Lovelace. **Foi considerada a primeira programadora da história**. Ela projetava programas para a máquina de Babbage. Os conceitos de subrotina (sequência de instruções que pode ser usada várias vezes), loop (instrução que permite a repetição de uma sequência de cartões) e do salto condicional (permissão para saltar algum cartão caso uma condição seja satisfeita) foram desenvolvidos por Ada.

Ada Lovelace e Babbage foram dois pesquisadores que estavam muito além do seu tempo. Várias outras máquinas mecânicas de somar foram construídas com diversas finalidades como controlar negócios (principalmente caixas registradoras) e cálculos de engenharia, mas não alcançaram grande sucesso. A prova desse avanço e da grandiosidade do feito de Babbage está registrada na história, pois até a década de 1940 nada semelhante foi inventado que pudesse superar o seu computador analítico.

As ideias de Babbage foram comprovadas só por volta de 1936 quando um jovem matemático chamado Alan Turing, publicou um artigo, pouco conhecido, **On computable numbers**.

O nome de **Alan Turing** é quase desconhecido para o público, mas sua contribuição foi fundamental para o desenvolvimento de ideias que ocorreriam antes do computador propriamente dito tornar-se realidade. A partir da publicação de Turing, os cientistas admitiam que a matemática era uma ciência inteiramente relacionada com regras lógicas e não uma arte misteriosa como se imaginava. Alan Turing criou a primeira máquina para decifrar os códigos alemães durante a segunda guerra mundial. Tomas Flowers, em 1942, criou o Colossus, utilizando as mesmas ideias de Turing. Era uma máquina que, uma vez plugada, programada e alimentada, resolvia qualquer questão de criptografia em poucos minutos.

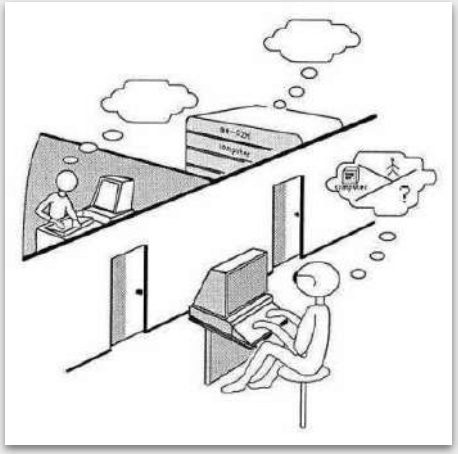
Figura 7 - A colossus e Alan Turing

Fonte: Imagem retirada da internet. Disponível em: https://www.tecmundo.com.br/historia/40576-colossus-heroi-de-guerra-e-um-dos-primeiros-computadores-do-mundo.htm. Acesso em: 01 de nov.2017.

Saiba mais!

O Teste de Turing

Figura 8 - O teste de Turing



Fonte: Imagem retirada da internet. Disponível em: https://turandursun.com/forumlar/showthread.php?t=28165.

O teste de Turing consistia em submeter um operador, fechado numa sala, para descobrir se quem respondia suas perguntas, introduzidas através do teclado, era um outro homem ou uma máquina. O objetivo era descobrir se poderia ser atribuída a uma máquina a noção de inteligência.

Acesse: https://pt.wikipedia.org/wiki/Teste_de_Turing

1.4 A máquina de tabular

O próximo avanço dos computadores foi a invenção da máquina capaz de processar dados baseada na separação de cartões perfurados (pelos seus furos). A façanha foi realizada pelo americano **Herman Hollerith (1860-1929)**. Quando ele era funcionário do Departamento de Recenseamento dos EUA, percebeu que o processamento do censo anual levava cerca de 7 anos para ser terminado apenas contabilizando as respostas de perguntas simples: sim ou não.

Em 1890, **a máquina de Hollerith** foi utilizada e reduziu o tempo de execução dos cálculos para apenas 2 anos e meio. Sua máquina teve o diferencial de utilizar eletricidade na separação, contagem e tabulação dos cartões perfurados. Mais tarde, Hollerith fundou uma companhia para produzir máquinas de tabulação. Anos depois, em 1924, essa companhia veio a se chamar IBM.

Você sabia?

A empresa fundada por Hollerith é hoje conhecida como IBM - Internacional Business Machines.

Interessante não?

Para conhecer mais sobre o surgimento da IBM visite: https://www.ibm.com/br/ibm/history.

1.5 A Lógica binária

Pingala, um matemático indiano, inventou o sistema de numeração binário por volta do século III a.C. O sistema desenvolvido por Pingala estabelece que sequências específicas de uns e zeros podem representar qualquer número, letra, imagem, som ou qualquer outro tipo de informação computacional. Este sistema é largamente empregado no processamento de todos os sistemas computacionais atuais.

Utilizando o sistema binário, **Gotfried Leibniz**, em 1703, desenvolveu a lógica num sentido formal e matemático, no qual os zeros e uns representavam também os conceitos de ligado ou desligado, verdadeiro ou falso, ou ainda, válido ou inválido.

Empregando cartões perfurados para seu controle de produção, em 1801, a máquina de tear inventada por **Joseph Marie Jacquard** também utilizava o conceito binário no qual os buracos no cartão indicavam o um (ligado) e as áreas não furadas indicavam os zeros (desligado). É claro que esse sistema de controle e automação por cartões está longe de ser um computador, mas demonstrou que as máquinas poderiam ser controladas pelo sistema binário empregando programação.

Em 1854, **George Boole** publicou a álgebra booleana como um sistema completo que permitiria a construção de modelos matemáticos para o processamento computacional.

Em álgebra booleana a construção de qualquer informação é feita apenas pelas combinações dos valores 0 e 1 (verdadeiro e falso). Os equipamentos do século XIX utilizavam a base decimal (0 a 9).

O desenvolvimento das tecnologias no futuro demonstrou que ocorreram dificuldades para se implementar o sistema decimal em componentes eletrônicos. O caminho natural de evolução desses novos componentes foi empregando o sistema binário.


E então, turma, gostou da nossa volta no tempo para compreender o surgimento dos primórdios do computador?
Muito intrigante, professor!
Ótimo. Agora que conhecemos seus antecessores, vamos avançar pela nossa trilha no tempo para conhecer como eles avançaram ainda mais! Segurem-se. Vamos à leitura! Boa viagem.

1.6 O surgimento do computador e suas gerações

Os computadores podem ser compreendidos como máquinas que conseguem realizar milhares de cálculos automaticamente, compostos de dispositivos de processamento, armazenamento, entrada e saída. A computação pode ser compreendida como a busca de uma solução para um problema a partir de entradas (inputs) e tem os seus resultados (outputs) depois de trabalhada através de um algoritmo (programa ou software). A teoria da computação trabalha com estes aspectos, sendo um subcampo da ciência da computação e da matemática. Durante milhares de anos, a computação foi realizada com caneta e papel, ou com giz e ardósia, ou mentalmente, por vezes com o auxílio de tabelas ou utensílios artesanais como vimos a partir da evolução da história do computador. Assim, para entender a evolução dos computadores precisamos entender a evolução da tecnologia.

Os computadores atuais foram desenvolvidos com pesquisas na área da informática durante a **Segunda Guerra Mundial**. Foi naquela época que nasceram os computadores atuais. A união de esforços entre as forças armadas e a academia levou ao aprimoramento das tecnologias existentes, tomando por base as invenções anteriores como o calculador analítico de Babbage.

Como exemplo desse tipo de esforço, em 1948, nos EUA, a Marinha, a Universidade de Harvard e a IBM construíram o Mark I, um gigante eletromagnético. Mark I ocupava 120 m3, tinha milhares de relés e fazia muito barulho. Uma multiplicação de números de 10 dígitos levava 3 segundos para ser efetuada. Este é apenas um exemplo, e ele não foi o primeiro.

Para facilitar a compreensão vamos dividir a geração de computadores de acordo com as tecnologias utilizadas para sua construção.

Em cerca de 50 anos, as tecnologias deram um salto que pode ser caracterizado por 4 fases: **a válvula a vácuo; o transistor; o circuito integrado e o microprocessador**. A evolução do poder de cálculos (processamento).

A seguir vamos nos debruçar sobre as gerações de computadores. As datas utilizadas para definir os intervalos não são tão precisas e tomam por base as invenções, o surgimento das tecnologias e lançamentos de equipamentos da época. Elas servem apenas para balizar a linha do tempo.

a) Primeira geração de computadores (1946 - 1957) - a válvula a vácuo

Foi marcada pela utilização de relés, circuitos eletrônicos e de válvulas. Realizavam operações internas em ciclos medidos em segundos (ciclo: o tempo para buscar um dado, processar e retornar com o resultado). Esse tempo era alto devido ao uso de relés mecânicos muito lentos. Os relés foram mais tarde substituídos por válvulas.

As válvulas aquecem bastante e costumam queimar com facilidade. Além disso, a programação era realizada diretamente na linguagem de máquina, o que dificultava a programação e, consequentemente, despendia muito tempo.

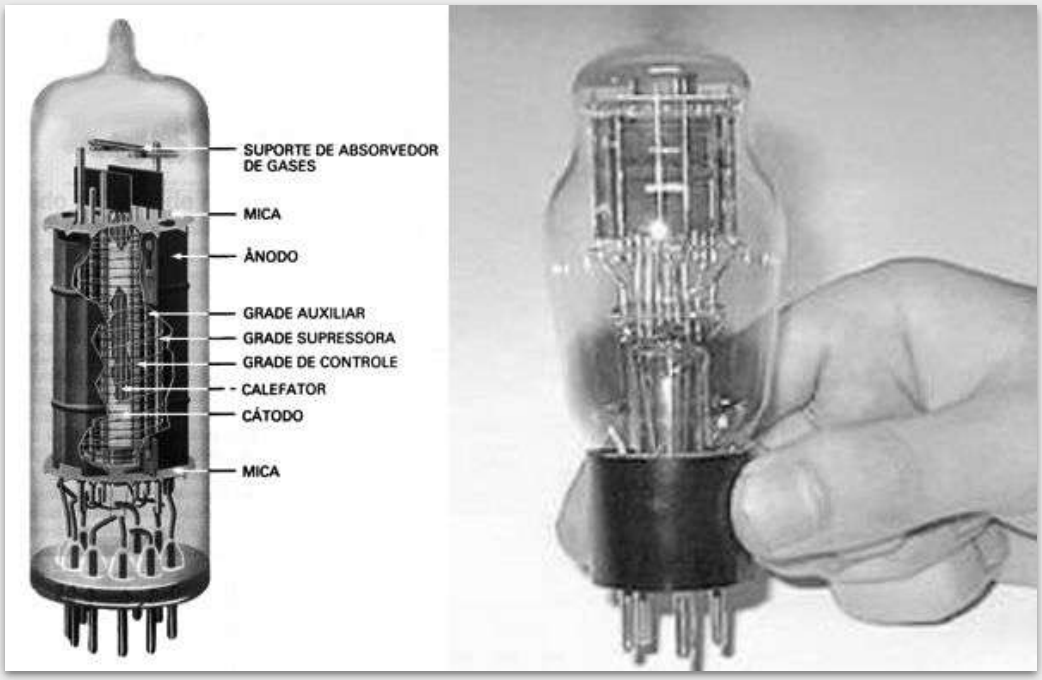
O armazenamento dos dados era realizado em cartões perfurados que depois passaram a ser feitos em fita magnética.

Nenhum dos computadores da primeira geração possuíam aplicação comercial, eram utilizados para fins balísticos, predição climática, cálculos de energia atômica e outros fins científicos.

Saiba mais!

A válvula é um tubo de vidro, similar a uma lâmpada fechada sem ar em seu interior, ou seja, um ambiente fechado a vácuo, e contendo eletrodos, cuja finalidade é controlar o fluxo de elétrons. As válvulas eram do tamanho de uma lâmpada.

Figura 9 - A válvula



Fonte: imagem retirada da internet. Disponível em: http://dboanarede.blogspot.com/2014/08/valvulas_26.html

O primeiro computador inventado pelo homem foi o **ENIAC (Eletronic Numerical Integrator and Computer)** que tinha como objetivo computar trajetórias táticas durante a II Guerra Mundial, em 1943, mas foram lançadas apenas em 1946, pelos cientistas John Eckert e John Mauchly. Ele possuía 17.468 válvulas, pesava 30 toneladas, tinha 180 m^2 de área construída, sua velocidade era da ordem de 100 kHz. O ENIAC possuía apenas 200 bits de memória RAM.

Nesta época, **John Von Neumann (1945)** formalizou o projeto lógico de um computador. Neste processo de desenvolvimento, ele sugeriu que as instruções (o programa constituído por um conjunto de instruções) fossem armazenadas na memória do computador, de maneira diferente ao que se fazia nos computadores anteriores que liam os cartões perfurados e executam as instruções uma a uma. Até hoje os computadores seguem o que foi preconizado por Von Neumann em seu modelo de arquitetura computacional.

Saiba mais!

O ENIAC começou a ser desenvolvido em 1943, durante a II Guerra Mundial, para computar trajetórias táticas que exigissem conhecimento substancial em matemática, mas só se tornou operacional após o final da guerra. O ENIAC era programado através de milhares de interruptores, podendo cada um deles assumir o valor 1 ou 0 consoante o interruptor estava ligado ou desligado.

Para o programar era necessário uma grande quantidade de pessoas que percorriam as longas filas de interruptores dando ao ENIAC as instruções necessárias para computar, ou seja, calcular.

Figura 10 - ENIAC - Representante da primeira geração de computadores

Fonte: imagem retirada da internet. Disponível em: http://evo-computadores.blogspot.com/2012/10/eniac-o-primeiro-computador.html

b) Segunda geração de computadores (1958 - 1964) - o transistor

A segunda geração de computadores foi marcada pela substituição da válvula pelo transistor. O transistor revolucionou a eletrônica, em geral, e os computadores em especial. Eles eram muito menores do que as válvulas a vácuo e tinham outras vantagens: não exigiam tempo de pré-aquecimento, consumiam menos energia, geravam menos calor e eram mais rápidos e confiáveis. No final da década de 50, os transistores foram incorporados aos computadores.

Na segunda geração, o conceito de Unidade Central de Processamento (CPU), memória, linguagem de programação e entrada e saída foram desenvolvidos. O tamanho dos computadores diminuiu consideravelmente.

Outro desenvolvimento importante foi a mudança da linguagem de máquina para a linguagem assembly, também conhecida como linguagem simbólica. A linguagem assembly possibilita a utilização de mnemônicos para representar as instruções de máquina. Em seguida, vieram as linguagens de alto nível, como, por exemplo, Fortran e Cobol.

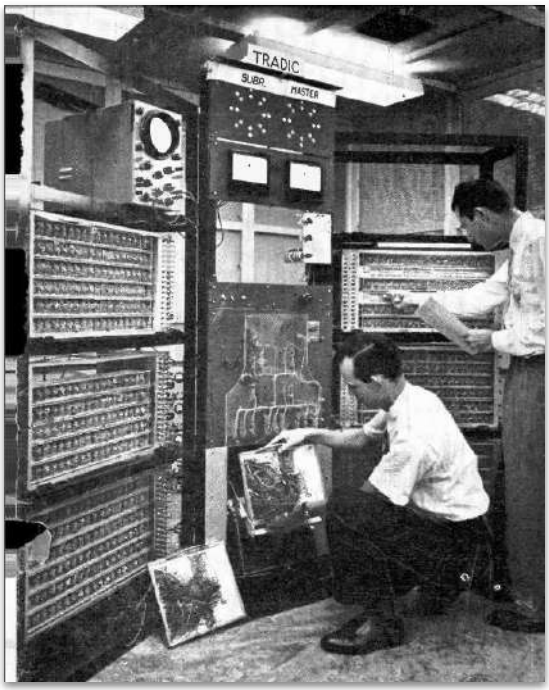
No mesmo período, surgiu o armazenamento em disco, complementando os sistemas de fita magnética e possibilitando ao usuário acesso rápido aos dados desejados.

Os computadores dessa época foram denominados de mainframes ou computadores de grande porte. Suas salas de operação eram enormes, acondicionados em ambientes de baixa temperatura, para proteger seus inúmeros transistores. Apenas órgãos governamentais e universidades possuíam esses equipamentos devido ao seu alto custo.

Alguns computadores dessa geração foram:

- TRADIC
- PDP-1
- IBM 1620
- IBM 7094
- CDC 1604
- CDC 3600
- UNIVAC 1108

Figura 11 - Computadores da segunda geração



Prototype TRADIC at Bell Labs, 1955

Foto retirada da internet. Disponível em: <https://en.wikipedia.org/wiki/TRADIC#/%20%20media/File:TRADIC_computer.jpg>. Acesso em: 01 de nov.2017.

PDP-1

Foto retirada da internet. Disponível em: <https://en.wikipedia.org/wiki/PDP-1#/%20%20media/File:Steve_Russell_and_PDP-1.png>. Acesso em: 01 de nov.2017.

Saiba mais!

O transístor (português europeu) ou transistor (português brasileiro) é um componente eletrônico que começou a popularizar-se na década de 1950, tendo sido o principal responsável pela revolução da eletrônica na década de 1960. São utilizados, principalmente, como amplificadores e interruptores de sinais elétricos, além de retificadores elétricos em um circuito, podendo ter variadas funções. O termo provém do inglês transfer resistor (resistor/resistência de transferência), como era conhecido pelos seus inventores.

Figura 12 - Transistor

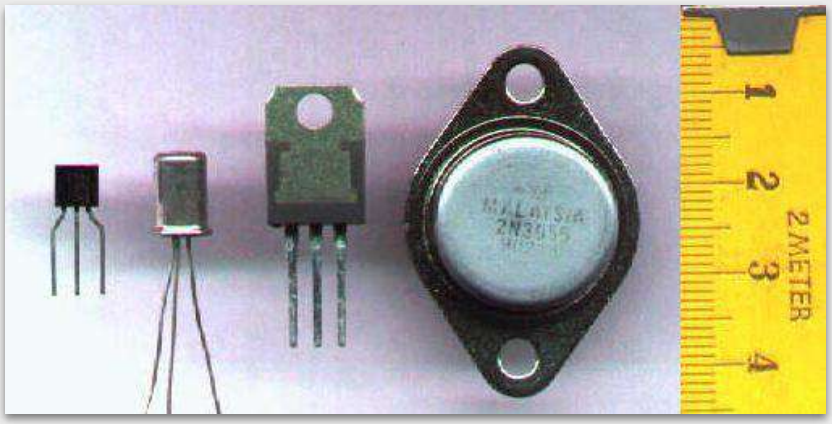


Imagem retirada da internet. Disponível em: <https://upload.wikimedia.org/wikipedia/commons/f/f8/Transistor-photo.JPG>. Acesso em: 01 de nov.2017.

c) Terceira geração de computadores (1965 - 1980) - o circuito integrado

O emprego de semicondutores (materiais de silício) com condutividade elétrica maior que a de um isolante, mas menor que a de um condutor, garantiu aumentos significativos na velocidade e na eficiência dos computadores. A terceira geração foi marcada pelos circuitos integrados, conhecidos como microchips. Esses novos componentes eram agrupamentos de um grande número de transistores, o que levou a produção de computadores menores, mais baratos e mais rápidos. O aumento de velocidade permitiu que mais tarefas fossem desempenhadas em períodos mais curtos de tempo.

Vamos refletir?

O diferencial dos circuitos integrados não era apenas o tamanho, mas o processo de fabricação que possibilitava a construção de vários circuitos simultaneamente, facilitando a produção em massa. Este avanço pode ser comparado ao advento da imprensa que revolucionou a produção dos livros. Didaticamente os circuitos integrados são categorizados de acordo com a quantidade de integração que eles possuem:

- LSI (Large Scale Integration - 100 transistores): computadores da terceira geração;
- VLSI (Very Large Scale Integration - 1.000 transistores): computadores da quarta geração;
- ULSI (Ultra-Large Scale Integration - milhões de transistores): computadores da quinta geração.

Nesta geração, surgem o teclado para digitação de comandos e o monitor que permite a visualização de sistemas operacionais primitivos. Nada que se compare aos sistemas gráficos que conhecemos e utilizamos hoje.

Figura 13 - Computadores da terceira geração

Exemplar do Altair 8800 no Museu de História Americana no Smithsonian. Primeiro computador pessoal portátil, produzido industrialmente para venda em massa.

Fonte: foto retirada da internet. Disponível em: <https://pt.wikipedia.org/wiki/Altair_8800#/media/%20%20File:Altair_8800.jpg>. Acesso em: 01 de nov.2017.

Apple II Lançado em 1976, por Steve Jobs e Steve Wozniak (fundadores da Apple Corp.) Foi o primeiro microcomputador pessoal a ter sucesso comercial.

Foto retirada da internet. Disponível em: <https://pt.wikipedia.org/wiki/Apple_II#/media/File:Apple-II.jpg>. Acesso em: 01 de nov.2017.

Um computador que representa a terceira geração foi o IBM's System/360, voltado para o setor comercial e científico. Ele possuía uma arquitetura plugável, na qual o cliente poderia substituir as peças que dessem defeitos. Além disso, um conjunto de periféricos eram vendidos conforme a necessidade do cliente. Outro evento importante desta época foi que a IBM passou a separar a criação de hardware do desenvolvimento de sistemas, iniciando o mercado da indústria de softwares. Isto foi possível devido à utilização das linguagens de alto nível nestes computadores.

Saiba mais!

Linguagem de alto nível

Uma linguagem é considerada de alto nível quando ela pode representar ideias abstratas de forma simples, diferente da linguagem de baixo nível que representa as próprias instruções de máquina.

Exemplo de linguagem de alto nível:

```
x = y * 7 + 2
```

Mesmo código em baixo nível (assembly):

```
load y    // carrega valor de y
mul 7     // multiplica valor carregado por 7
add 2     // adiciona 2
store x   // salva o valor do último resultado em x
```

Os códigos **load, mul, add** e **store** são os mnemônicos que representam as instruções em código de máquina (binário).

d) Quarta geração de computadores (1981 - 1995) - microprocessador

Os computadores da quarta geração são reconhecidos pelo surgimento dos processadores — unidade central de processamento. Os sistemas operacionais como MS-DOS, UNIX, Apple's Macintosh foram construídos. Linguagens de programação orientadas a objeto como C++ e Smalltalk foram desenvolvidas. Discos rígidos eram utilizados como memória secundária. Impressoras matriciais, e os teclados com os layouts atuais foram criados nesta época. Os computadores eram mais confiáveis, mais rápidos, menores e com maior capacidade de armazenamento. Esta geração é marcada pela venda de computadores pessoais.

Principais características:

- Introdução dos microprocessadores;
- Desenvolvimento dos computadores pessoais (PC);
- Evolução dos diversos componentes (hardware e software);
- Escala de integração - VLSI: Very Large Scale Integration;
- Computadores pessoais e estações de trabalho;
- Sistemas operacionais MS-DOS, Windows, UNIX, GNU/Linux;
- Sistemas operacionais de rede;
- Evolução dos dispositivos diversos componentes (hardware e software);
- Processamento em paralelo generalizado;
- Microprogramação.

Figura 14 - Computadores pessoais da quarta geração

IBM PC (Personal Computer ou "computador pessoal") - modelo 5150

Foto retirada da internet. Disponível em:<https://pt.wikipedia.org/wiki/IBM_PC>. Acesso em: 01 de nov.2017.

Apple Macintosh 512KB

Foto retirada da internet. Disponível em: <https://pt.wikipedia.org/wiki/Macintosh>. Acesso em: 01 de nov.2017.

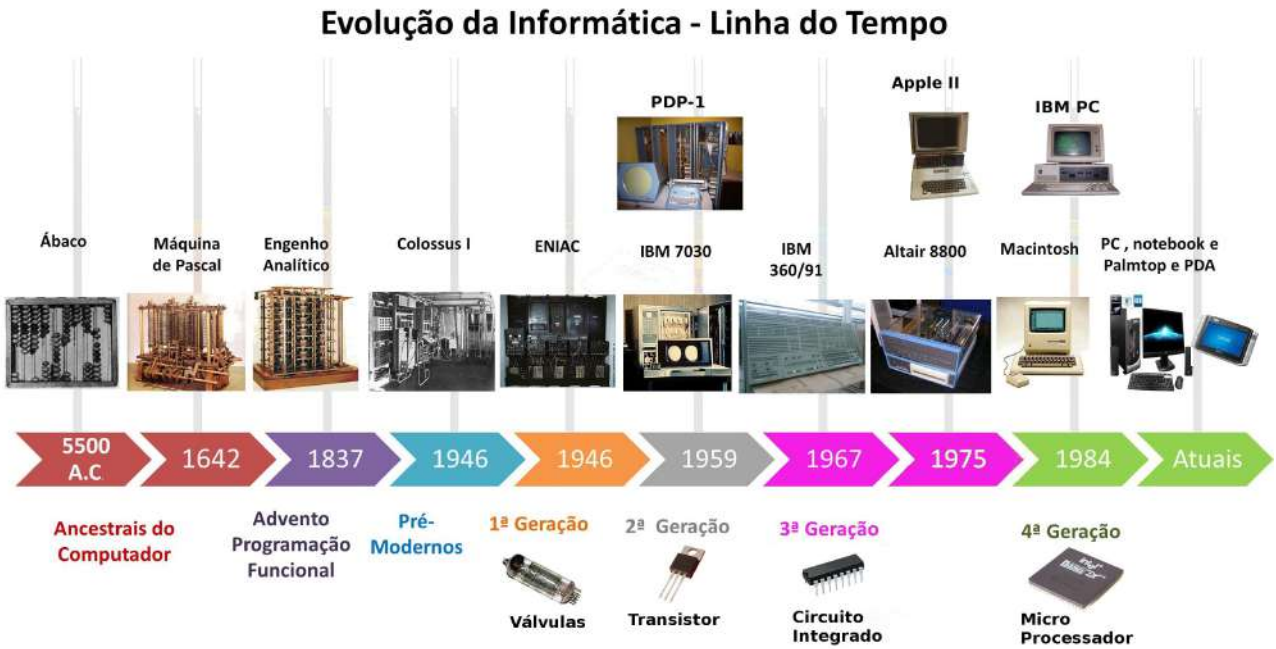
e) Quinta geração (1996 — dias atuais)

Os computadores da quinta geração usam processadores com milhões de transistores. Nesta geração, surgiram as arquiteturas de 64 bits, os processadores que utilizam tecnologias RISC e CISC, discos rígidos com capacidade superior a 600GB, pen-drives com mais de 1GB de memória e utilização de disco ótico com mais de 50GB de armazenamento.

A quinta geração está sendo marcada pela inteligência artificial e por sua conectividade. A inteligência artificial pode ser verificada em jogos e robôs ao conseguir desafiar a inteligência humana. A conectividade é cada vez mais um requisito das indústrias de computadores.

Hoje em dia, queremos que nossos computadores se conectem ao celular, à televisão e a muitos outros dispositivos como geladeira e câmeras de segurança. Esta geração é marcada pela miniaturização dos componentes e do vertiginoso crescimento dos programas, detalhes gráficos e multimídia como 3D, além de realidade virtual.

Figura 15 - Linha do Tempo - Evolução da Informática



Fonte: elaborada pelo autor

f) A evolução dos processadores e a convergência digital

O contínuo processo de crescimento dos nossos computadores atuais sem dúvidas é fruto da evolução dos processadores e da convergência digital. A Lei de Moore trata da capacidade de processamento dos PCs. Publicada pela primeira vez na Electronics Magazine, em 1965, a regra proposta por Gordon Earl Moore, engenheiro e um dos fundadores da Intel, ditava o ritmo com o qual a indústria de chips deveria evoluir. A regra afirmava que o número de transistores em um chip deve dobrar a cada 18 meses, mantendo o custo e o espaço ocupado. Na prática, isso significa que a capacidade de processamento deve aumentar em 100% a cada um ano e meio. O poder bruto não é a única modificação; o chip deve ser mais eficiente, o que significa que ele deve ser capaz de realizar mais tarefas sem consumir mais energia.

E isso se aplica a celulares, cada vez mais finos e potentes, aos chips usados em objetos domésticos inteligentes, tecnologia vestível e tudo mais. Facilita a difusão da internet e da informação, também.

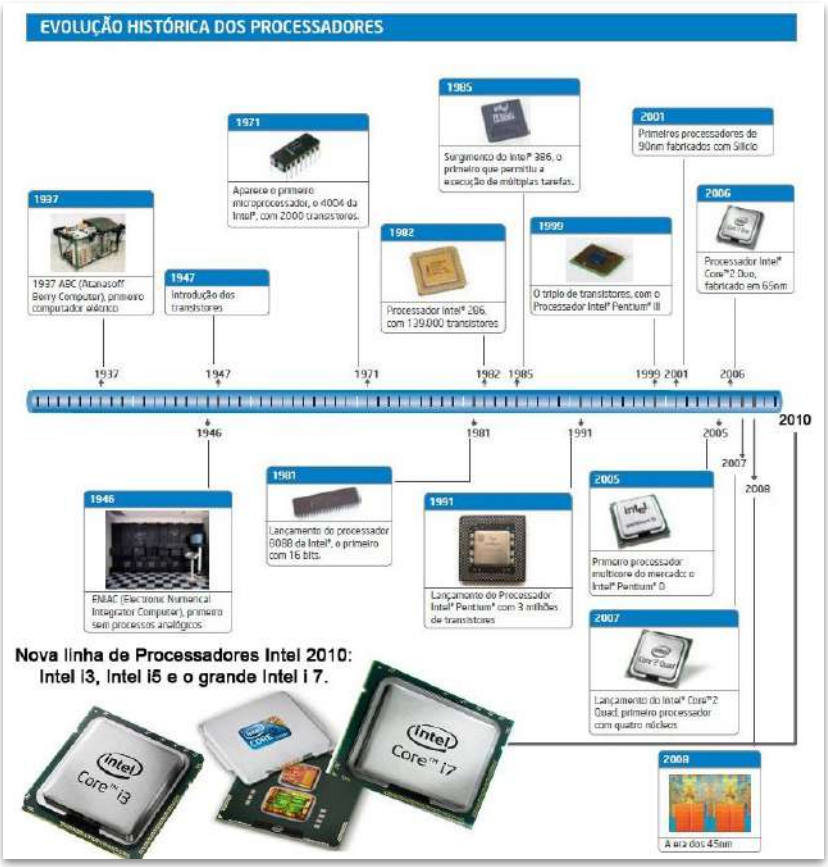
A convergência digital está cada vez mais presente em nosso dia a dia. A migração de funcionalidades para um dispositivo único é uma realidade. Até nossos celulares e dispositivos móveis são computadores, pois além de realizar ligações telefônicas executam outras funções que estavam em outros equipamentos separados, mas agora estão presentes em apenas um equipamento (máquina fotográfica, computador, modem de acesso à internet, videogame, rádio, tv, gps etc.)

Questiona-se a continuidade da vigência da Lei de Moore e por quanto mais tempo ela ainda poderá se manter válida. Precisamos refletir que a evolução da tecnologia não vai parar, mas o ritmo poderá mudar tanto para cima quanto para baixo, a depender das novas descobertas científicas. As empresas podem considerar que o investimento para manter este ritmo é muito alto e não compensa mais; ao mesmo tempo, a competitividade pela inovação

pode fazer acelerar este ritmo, alavancando o investimento ou ainda o surgimento de novos materiais e técnicas. Existe também o problema da física: por quanto mais tempo será possível continuar reduzindo os transistores e implantando em seus chips?

A fabricante Intel fez a previsão de que o silício não será mais usado em seus processadores de 7 nanômetros, previstos para 2017. A mudança de materiais, a evolução da nanotecnologia são fatores que podem chacoalhar a indústria e colocar a Lei de Moore no passado.

Figura 16 - Evolução dos processadores



Evolução vertiginosa dos processadores desde a sua introdução.
<http://4.bp.blogspot.com/_ofMdY3_XPW8/S8M-7uG4Y2I/AAAAAAAAAH4/8fJV2lSszVw/s1600/Evolu%C3%A7%C3%A3o+historica+processadores2.jpg>.
Acesso em: 01 de nov.2017.

Você sabia?

O primeiro bug de computador!

O termo “bug” foi associado a interferências e mal funcionamento bem antes de que existissem os computadores modernos, sendo Thomas Edison um dos primeiros a usar este significado.

Mas foi uma mulher, Grace Murray Hopper, quem, em 1945, documentou o primeiro bug da informática. Ao longo dos anos este termo se popularizou e hoje em dia utilizamos frequentemente para referir-se aos erros nos programas de computador. Grace Brewster Murray Hopper (1906-1992), doutora em matemática pela Universidade de Yale, passou pela história por ser uma inovadora programadora durante as primeiras gerações de computadores. Em 1943, durante a segunda guerra mundial, decidiu incorporar-se à marinha americana e foi enviada para o laboratório de cálculo Howard Aiken na Universidade de Harvard, onde trabalhou como programadora no Mark I. Em 9 de setembro de 1945, o grupo de trabalho de Aiken e Grace se encontrava na sala do Mark II tentando averiguar porque o computador não funcionava adequadamente. Depois de um exame minucioso, conseguiram detectar o problema, uma mariposa entre os contatos de uns dos relés do Mark II. Mais tarde, Grace registraria o incidente no caderno de registros, colou a borboleta que causou o problema e anotou embaixo a frase “First atual case of bug being found” (ver imagem abaixo).

A partir de então, cada vez que algum computador dava problemas eles diziam que tinha bugs. Anos mais tarde, Grace também cunharia o termo “debug” para referir-se à depuração de programas.

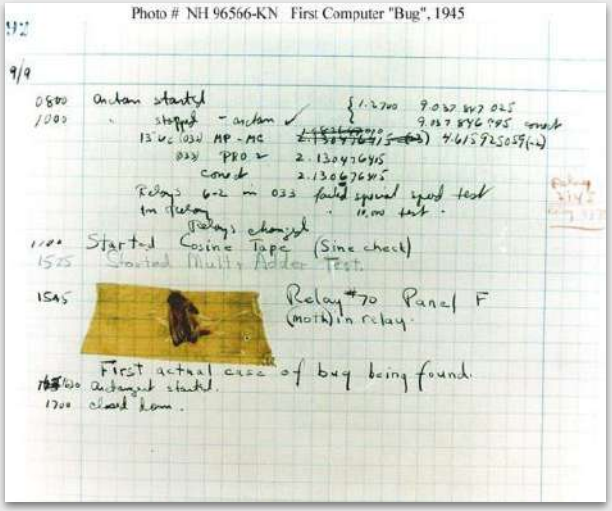
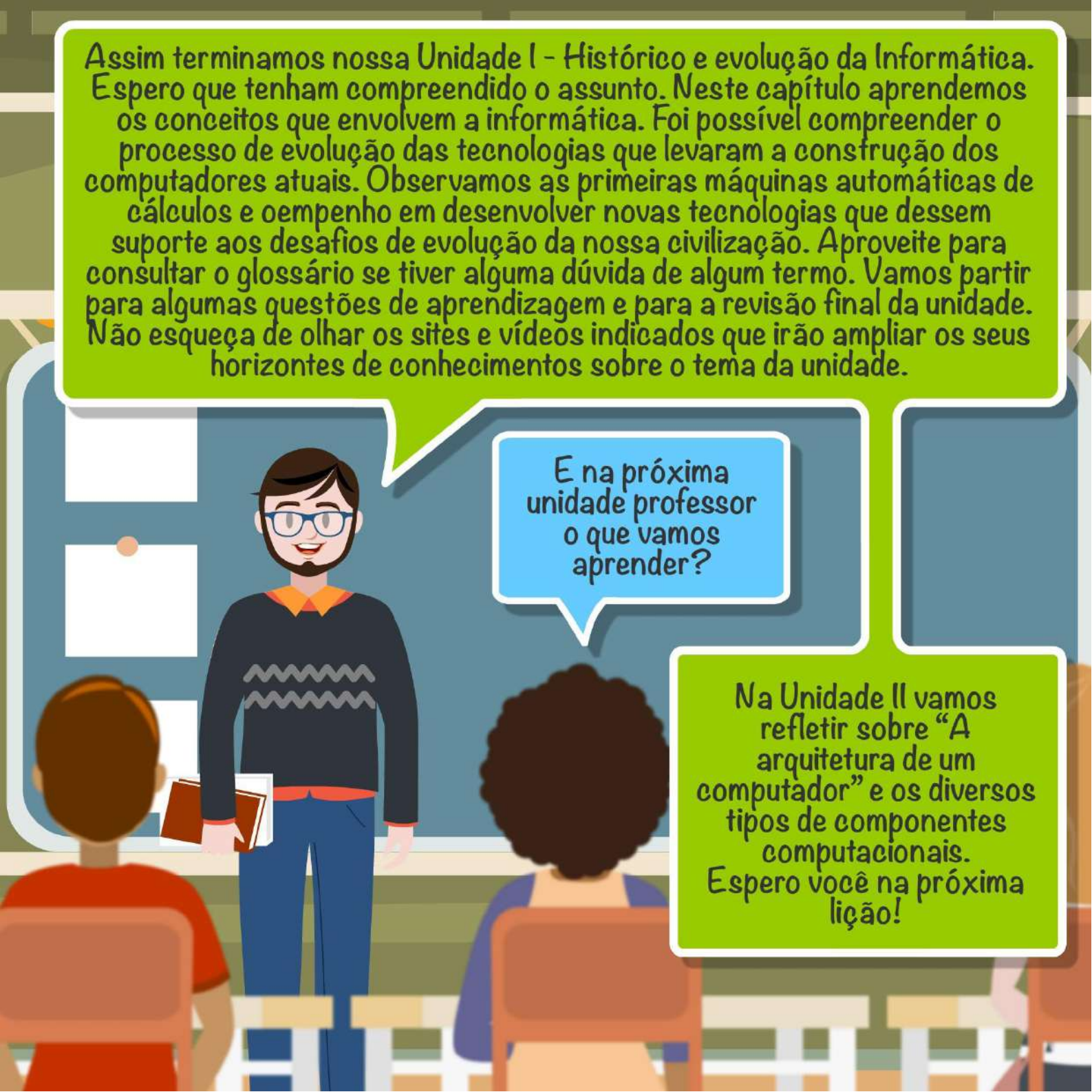


Figura 17 - Registro do primeiro bug de computador

Fonte: Imagem retirada da internet. Disponível em: <https://imagens.mdig.com.br/diversos/928_04.jpg>. Acesso em: 01 de nov.2017.


Assim terminamos nossa Unidade I - Histórico e evolução da Informática. Espero que tenham compreendido o assunto. Neste capítulo aprendemos os conceitos que envolvem a informática. Foi possível compreender o processo de evolução das tecnologias que levaram a construção dos computadores atuais. Observamos as primeiras máquinas automáticas de cálculos e oempenho em desenvolver novas tecnologias que dessem suporte aos desafios de evolução da nossa civilização. Aproveite para consultar o glossário se tiver alguma dúvida de algum termo. Vamos partir para algumas questões de aprendizagem e para a revisão final da unidade. Não esqueça de olhar os sites e vídeos indicados que irão ampliar os seus horizontes de conhecimentos sobre o tema da unidade.
E na próxima unidade professor o que vamos aprender?
Na Unidade II vamos refletir sobre "A arquitetura de um computador" e os diversos tipos de componentes computacionais. Espero você na próxima lição!

Vamos rever?

A necessidade de calcular levou o homem a construir mecanismos que o ajudassem a realizar essa tarefa de forma rápida e confiável. O ábaco foi um dos primeiros instrumentos utilizados para essa finalidade. Com o passar do tempo e a evolução das tecnologias esses inventos passaram a ficar mais complexos, automatizados e rápidos.

De acordo com Tanembaum (2007), diversos inventores conseguiram construir máquinas de calcular em meados da década de 40 do século XX, dentre os quais podemos destacar: Howard Aiken, em Harvard; John Von Neumann, no Instituto para Estudos Avançados de Princeton; J. Presper Eckert e Willian Mauchley, na Universidade da Pensilvânia e Konrad Zuse, na Alemanha.

A invenção dos cartões perfurados e do computador analítico de Babbage deram ao homem a possibilidade de controlar a produção de teares e contribuíram para o surgimento das máquinas programáveis e com elas o nascimento da programação com Ada Lovelace. Alan Turing confirmou as teorias de Babbage e criou a máquina para decifrar os códigos alemães e mudou a história da computação.

George Boole publicou a álgebra booleana com um sistema completo que permitiria a construção de modelos matemáticos para o processamento computacional.

Vamos rever?

Os computadores atuais foram desenvolvidos com pesquisas na área da informática durante a Segunda Guerra Mundial.

Em cerca de 55 anos, as tecnologias deram um salto que pode ser caracterizado por 4 fases que definiram as gerações dos computadores: a válvula a vácuo (1ª geração); o transistor (2ª geração); o circuito integrado (3ª geração) e o microprocessador (4ª geração). A evolução de uma geração para outra ocorre a partir da evolução das tecnologias permitindo o avanço do poder de cálculos (processamento).

O ritmo de evolução dos processadores e computadores depende do interesse em pesquisas, das descobertas científicas, da inovação e do interesse da indústria. A Lei de Moore indica que a capacidade de processamento deve aumentar em 100% a cada um ano e meio. Mas até ela pode estar com os dias contados, afinal existe também o problema da física: por quanto mais tempo será possível continuar reduzindo os transistores e implantando em seus chips? A mudança de materiais, a evolução da nanotecnologia são fatores que podem chacoalhar a indústria e colocar a Lei de Moore no passado.

Saiba mais!

Sites indicados:

1. **Tec Mundo - Infográfico sobre a evolução dos computadores** - Disponível em:https://www.tecmundo.com.br/infografico/9421-a-evolucao-dos-computadores.htm
2. Computer History Museum (Califórnia - EUA) - Linha do tempo da computação - Disponível em: http://www.computerhistory.org/timeline/

Audiovisuais indicados:

1. Demonstração da Máquina Diferencial de Charles Babbage (inglês) - Disponível em: http://youtu.be/BlbQsKpq3Ak
2. Um dia feito de vidro (A Day made of glass) - Disponível em: https://www.youtube.com/watch?v=wk146eGRUtI
3. História: A evolução dos computadores - Disponível em: https://www.youtube.com/watch?v=mFdUqqwzbVs
4. Vida e Obra de Alan Turing - Disponível em: http://youtu.be/yIluxaHL0v0

Questões de autoaprendizagem

1) Como funciona um relé e uma válvula? Faça uma pesquisa para explicar o funcionamento de cada um deles.

2) A partir do texto sobre os computadores da primeira geração, faça uma reflexão e explique a afirmação: “Os computadores das primeiras gerações eram usados apenas pelos projetistas que os construíram”.

3) O que mudou nas gerações de computadores a partir da criação dos transistores e circuitos integrados? Faça um comparativo com a realidade atual de nossos computadores.

4) Pesquise sobre a Arquitetura de Von Neumann. Até hoje, os computadores seguem o que foi preconizado por ele em seu modelo de arquitetura computacional. Como funciona esse modelo? Monte um diagrama para explicar.

5) Reflita sobre o porquê do celular hoje não ser apenas utilizado para fazer ligações. O que aconteceu com ele? O que isso tem a ver com convergência digital?

Unidade 2

A Arquitetura de um Computador

Olá, Vamos continuar nossos estudos?
Nesta unidade, vamos conhecer a arquitetura dos computadores.
Ao término de seu estudo, esperamos atingir o seguinte objetivo: diferenciar os diversos tipos de componentes computacionais.

Nesta unidade, denominada **A arquitetura de um computador**, vamos aprender a distinguir entre componentes de hardware e de software que compõem um computador. Aprenderemos a identificar os principais componentes de um computador e suas funcionalidades. Depois vamos olhar com um pouco mais de detalhes as funcionalidades dos principais dispositivos de hardware e suas características técnicas. E, claro, iremos também conhecer a função básica de um Sistema Operacional e sua diferença em relação a outros programas. Finalizando, vamos aprender a fazer uso de manuais técnicos de hardware e de software.

No mundo de hoje, os computadores são usados para quase todas as tarefas imagináveis. Atividades de rotina, como pagar contas, comprar mantimentos ou se comunicar com um amigo podem ser feitas com um computador. É por isso que é importante não só saber como usar um computador, mas também entender os componentes de um computador e o que eles fazem.


Olá turma! Preparados para um novo desafio? Nesta unidade vamos conhecer como um computador funciona e como são organizados. O que compõe um computador? O que há por dentro dele?
Que massa! Vamos ver como tudo funciona.
Vamos começar!

2.1 O computador

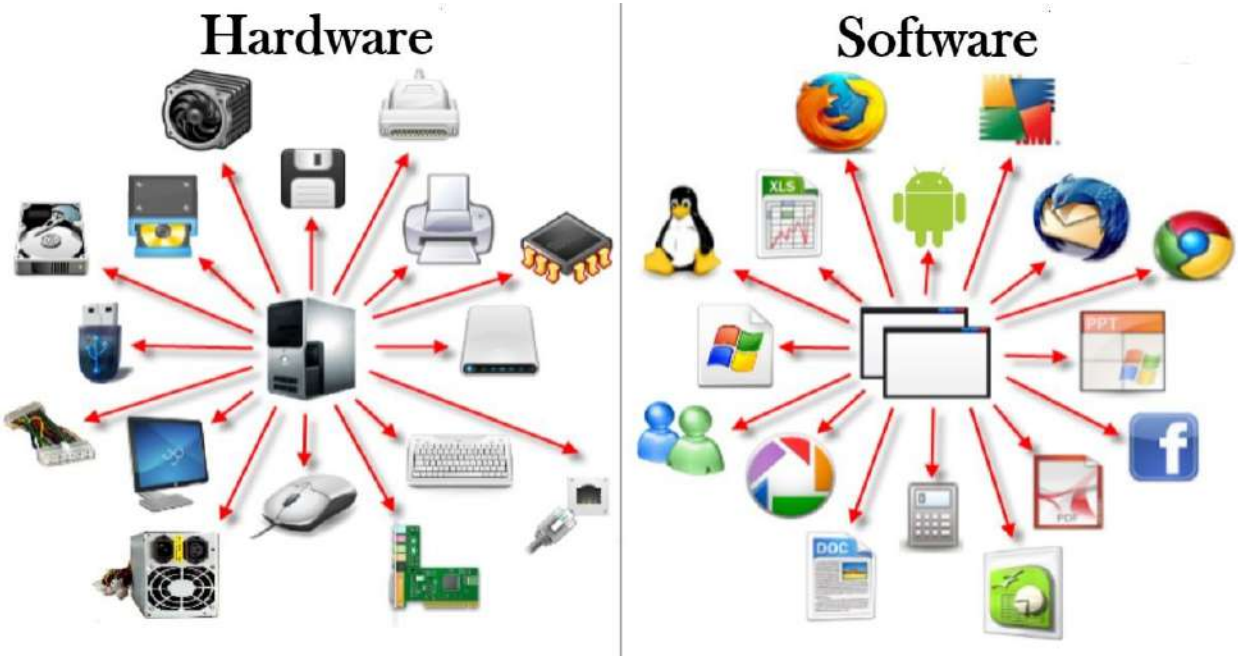
Vamos refletir: um computador pode ser compreendido como um dispositivo eletrônico usado para armazenar e processar informações. Seu papel em nossas vidas é muito importante. Os computadores são utilizados em áreas como educação e pesquisa e, também, para divulgar notícias, enviar mensagens de amigos e familiares, criar apresentações, manter registros pessoais e profissionais, realizar previsões meteorológicas, bem como para uma variedade de atividades de lazer ou de negócios.

Graças aos computadores, a economia de tempo, esforço e dinheiro é considerável. Um computador é um equipamento que pode sistematicamente colher, manipular e gerar resultados a partir dos dados que coletou para um ou diversos objetivos para o qual o programamos.

Podemos definir um computador como um conjunto de dispositivos, interligados entre si, capazes de realizar um tratamento automático das informações seguindo as instruções de um programa. Portanto, os computadores não são dispositivos simples. Eles são integrados por um conjunto de máquinas e dispositivos que funcionam de maneira sincronizada para processar automaticamente a informação. O conjunto de elementos físicos que compõem um computador é chamado de HARDWARE .

Além disso, eles exigem programas que indiquem o tipo de tratamento ou processamento a ser realizado. Os programas que os computadores utilizam são chamados de SOFTWARE.

Figura 1 - O Hardware e o Software em um computador



Fonte: Imagem retirada da internet – Disponível em:
<https://www.todamateria.com.br/hardware-e-software/>

Um Sistema Computacional pode ser compreendido como a interação existente entre os diversos componentes de hardware, software e peopleware trabalhando conjuntamente sobre um determinado conjunto de dados, produzindo informações e resultados de interesse para outros sistemas ou usuários.

2.2 Onde são utilizados?

Já parou para pensar em quais atividades utilizamos um computador atualmente?

Os computadores desempenham um papel crucial no nosso dia a dia. Eles são usados em empresas, escolas, escritórios, órgãos governamentais e lojas. Eles nos permitem comunicar com a família e os amigos, criar um orçamento para casa, reservar passagens de avião ou ingressos para o cinema ou ainda administrar um negócio.

Nas empresas, eles são usados para manter contas, criar registros de pessoal, rastrear inventário, preparar apresentações e relatórios, gerenciar projetos e se comunicar por e-mail. Eles também podem ser usados para projetar qualquer tipo de publicação, desde boletins informativos simples até revistas de moda, materiais de marketing, livros ou jornais. No setor de educação, os professores usam o computador na sala de aula como apoio a disciplinas ou cursos, através de material audiovisual complementar, ou ainda para manter um registro dos alunos e para acompanhar o seu desempenho. Assim como os alunos o utilizam para buscar informações sobre vários temas, criar ou enviar trabalhos de casa.

Se pensarmos em nível governamental, eles servem para organizar a informação em registros armazenados e atualizados. Eles também servem para oferecer serviços aos cidadãos. Assim, em um computador é possível consultar informações sobre as políticas atuais, cidadania e os assuntos do governo.

Na medicina, os médicos utilizam computadores para verificar os registros médicos de seus pacientes e encontrar informações sobre os medicamentos mais recentes disponíveis para tratar uma doença. Eles também usam computadores para discutir e compartilhar informações sobre várias doenças.

O computador também é usado para verificar os detalhes de uma conta bancária. Os comerciantes de ações usam computadores para obter informações imediatas sobre os mercados de ações, para comercializar e gerenciar seus investimentos. Os cientistas usam computadores para pesquisa, bem como para coletar e analisar informações. Por exemplo, para visualizar imagens do espaço e publicar informações sobre pesquisas recentes.

Os computadores também são usados para criar desenhos e imagens. Os fotógrafos usam computadores para editar e aprimorar imagens. Os escritores usam computadores para escrever o conteúdo de seus livros e criar ilustrações. Graças a eles, os escritores podem mudar o conteúdo com muita facilidade e economizar muito tempo.

Os computadores também são divertidos, pois através deles você pode ouvir música, assistir filmes, salvar e imprimir fotografias, enviar saudações e jogar.

Puxa, como utilizamos computadores em nosso dia a dia!

2.3 O processamento de dados em informação

O computador também é conhecido como um equipamento de processamento eletrônico de dados por ser composto de vários circuitos e componentes eletrônicos. Os dados processados de forma manual ou automática resultam em um produto final acabado: a informação. Apenas depois de processados e transformados em informação é que eles passam a ter algum significado para alguém ou para o próprio computador.

O processo de transformação de dados em informação realizado pelo computador é sempre orientado por um conjunto de instruções (programação) de maneira a produzir resultados com a mínima intervenção humana.

Dentre os benefícios para o emprego de computadores, podemos destacar:

A. grande velocidade no processamento e disponibilização de informações;
B. precisão no fornecimento das informações;
C. adequação para execução de tarefas repetitivas;
D. redução de custos operacionais;
E. compartilhamento de dados.

2.4 Peopleware

O termo peopleware refere-se aos agentes humanos (usuários, técnicos e programadores) que fazem uso e configuram as ações a serem realizadas pelo hardware e software. Simples, não? Somos nós que utilizamos e programamos os computadores.

2.5 Hardware

De forma geral, é a parte que podemos tocar no computador. Coletivamente, os equipamentos elétricos, eletrônicos e mecânicos que compõem um computador são chamados de hardware. Os dispositivos que se conectam a unidade do sistema (ou seja, o teclado, o mouse, os alto-falantes, o monitor e assim por diante) são conhecidos como dispositivos periféricos, ou seja, estão ligados ao computador.

a) Arquitetura geral de um computador

A arquitetura dos computadores atuais é atribuída a Von Neumann . A arquitetura de computadores se refere ao comportamento de um sistema computacional visível para o programador, ou seja, aos aspectos relacionados à execução lógica de um programa. A organização de computadores se refere às unidades estruturais e seus relacionamentos lógicos e eletrônicos (STALLINGS, 2010).

O que diferencia um computador de um equipamento eletrônico é a capacidade de poder executar qualquer programa que seja carregado em sua memória. Assim, refletindo sobre essa diferença podemos considerar que um computador é composto pela CPU, memória e dispositivos de E/S (Entrada/Saída). Para executar os programas (desempenhar a função básica de um computador) esses componentes precisam estar relacionados e interconectados de alguma maneira.

Vamos refletir?

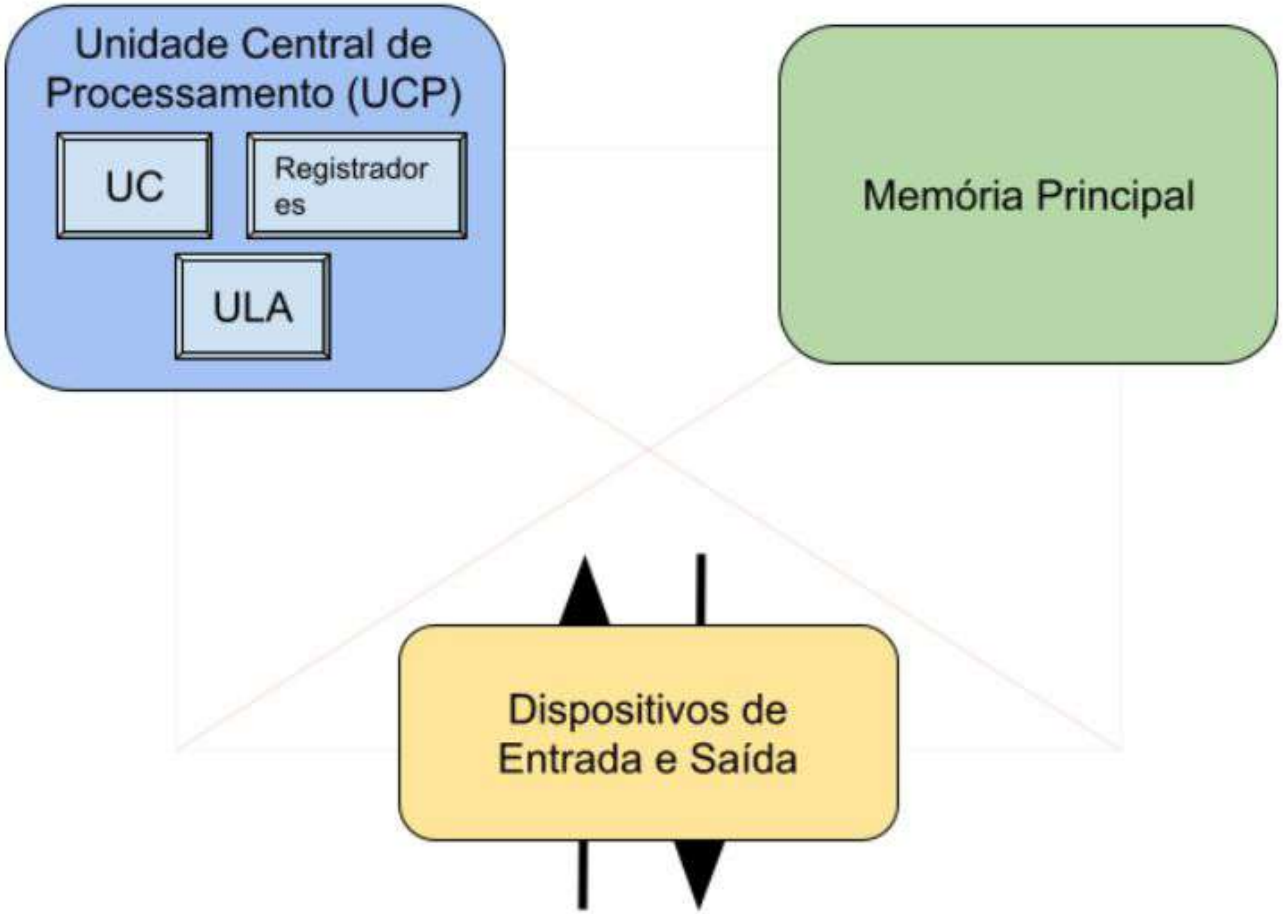
Os computadores manipulam dados (entrada e processamento) para produzir informações (saída) e seguram (armazenam) essas informações para uso futuro. Essas operações são concluídas de forma incrivelmente rápida. Um supercomputador atual pode executar 1.8 trilhões de operações por segundo. Se uma pessoa fizesse uma operação aritmética por segundo, sem parar, levaria mais de 31.000 anos para executar o número de operações que um supercomputador pode fazer em um segundo.

Saiba mais!

Arquitetura de Von Neumann. Disponível em: <https://pt.wikipedia. org/wiki/Arquitetura_de_von_Neumann>. Acesso em: 01 de nov.2017.

De acordo com a arquitetura de Von Neumann, esse modelo é composto pela Unidade Central de Processamento (CPU), pela memória e pelos dispositivos de entrada e saída, todos conectados entre si pelo barramento, conforme descrito no diagrama a seguir.

Figura 2 - A arquitetura de um computador



Fonte: Elaborado pelo autor

A Arquitetura de Von Neumann toma por base três conceitos importantes:

- as instruções e os dados ficam armazenados em uma única memória de leitura e escrita;
- o acesso ao conteúdo dessa memória é identificado por um endereço que indica sua posição, não importando o tipo de dados nele contido;
- a execução das instruções ocorre sempre em sequência, a não ser que o programa altere essa sequência encaminhando o fluxo de execução para a instrução seguinte.

O conceito introduzido por Von Neumann permitiu executar diversas aplicações diferentes sobre um mesmo hardware sem a necessidade de realizar modificações no computador para aceitar a execução de determinado programa. Nasceu, neste momento, o conceito de armazenamento de código (programa gravado e reutilizável), assim como a possibilidade de se ter um programa para cada propósito, sem necessidade de mudar o hardware.

Vamos refletir?

É importante perceber que neste modelo de Neumann a memória é um componente importante, pois para executar um programa, o computador primeiro precisa carregar os dados a partir de um dispositivo de entrada. Depois de carregá-lo em memória, na execução, provavelmente, necessitará fazer cálculos e armazenar esses resultados para aproveitá-los posteriormente.

Uma operação (aritmética ou lógica) pode precisar de mais de um dado de cada vez e é necessário ter um local para guardá-los e recuperar sempre que precisar. Outra questão é que nem sempre um programa é executado de forma sequencial e precisaremos acessar outras partes do código de forma não sequencial. Assim um computador precisa da memória principal para armazenar instruções e dados temporariamente.

b) Os componentes e suas funções

Vários componentes são necessários para que um computador possa funcionar. A Unidade Central de Processamento (UCP) ou processador é composto por componentes que possuem os módulos que interpretam os códigos (software) e os que executam as funções lógicas e aritméticas de propósito gerais. Porém, os dados e instruções devem ser inseridos de alguma forma no sistema e, desta forma, é necessário um módulo que permita a entrada de dados. De igual forma, o sistema precisa ser capaz de exibir os resultados depois do processamento. Esses módulos são chamados de dispositivos de Entrada e Saída. Essa organização está descrita na Figura 2.

De acordo com Stallings (2010), a função básica desempenhada por um computador é executar o conjunto de instruções, que constituem um programa, armazenadas na memória. Assim, o processador realiza o trabalho efetivo de executar as instruções especificadas no programa que foi carregado na memória do computador.

c) Dispositivos de Entrada e Saída (E/S)

Os dispositivos de entrada e saída (E/S) são responsáveis pela comunicação com os usuários e o mundo externo (monitor, teclado ou mouse, por exemplo). Cada dispositivo de E/S possui um controlador responsável pela comunicação com o computador. Esse exemplo fica melhor compreendido se pensarmos na placa mãe (computador) e uma placa de vídeo (controlador) para o dispositivo de saída monitor.

d) Memória

É na memória principal que os dados são armazenados. Eles são transferidos por meio dos barramentos que interligam os demais componentes sempre que necessários.

Segundo Monteiro (2007), num sistema de computação não é possível construir ou utilizar apenas um tipo de memória, já que ela por si mesma já se constitui de um subsistema, sendo constituída por vários tipos de componentes.

A memória principal é conhecida como memória RAM (Random-Access Memory – Memória de Acesso Aleatório). Conforme já observamos, uma das características principais do modelo de Von Neumann é o uso do conceito de programa armazenado. O processador pode acessar instruções uma atrás da outra, imediatamente, pois elas estão armazenadas na memória. A memória especificada para armazenar o programa e os seus dados é chamada de memória principal.

A memória RAM (memória de acesso aleatório) tem como característica principal a sua volatilidade. Isso quer dizer que ela necessita de energia constante para que os dados fiquem armazenados na sua estrutura. Assim, a RAM é utilizada para armazenamento temporário de dados.

e) Unidade Central de Processamento (UCP)

Na região conhecida como Unidade Central de Processamento UCP (CPU – Central Processing Unit), ou processador, estão isolados os circuitos que executam operações com dados, tais como adição e subtração. A unidade central de processamento divide-se em: Unidade de Controle (UC), necessária ao sincronismo da tarefa; ULA (Unidade Lógica e Aritmética), responsável pelos cálculos e os registradores, para armazenar temporariamente os resultados.

Saiba mais!

Que tal assistir a um vídeo para compreender melhor como a CPU se comunica com os dispositivos do computador?

Para saber um pouco mais sobre como a CPU interage com a memória e os dispositivos de entrada e saída clique no link abaixo e assista ao vídeo que selecionamos. Se desejar pode ler o QR-CODE e acessar diretamente no seu celular.

Acesse: https://www.youtube.com/watch?v=hCB-5X4FkCQ

A CPU troca dados com a memória por meio dos barramentos. O registrador de endereçamento à memória é responsável por guardar o endereço da memória a ser usado para executar a próxima instrução de leitura ou escrita.

O registrador temporário de dados recebe um dado lido da memória e armazena um valor a ser gravado na memória. Existe ainda o registrador de endereçamento de E/S que armazena o endereço de um determinado módulo de dispositivo de E/S, e um registrador temporário de dados de E/S que é usado para trocar dados entre a CPU e o módulo do dispositivo de E/S.

f) Processamento de Instruções

O processamento de instruções de um computador pode ser dividido em dois passos simples:

- as instruções são lidas na memória pelo processador, uma de cada vez (ciclo de busca);
- cada instrução é executada pelo processador, podendo utilizar diversas operações (ciclo de execução). Todo o processamento utilizado para o ciclo de execução é definido como o ciclo de instrução.

O processador busca uma instrução na memória a cada início de ciclo de instrução. Utilizando o registrador denominado de contador de programa (program counter - PC) o processador guarda o endereço da próxima instrução que deve ser lida na memória.

Depois de cada busca de nova instrução, o processador faz o incremento do contador de programa, mas nem sempre de forma sequencial. A instrução lida na memória é armazenada em um registrador conhecido como registrador de instruções (instruction register – IR).

O registrador de instruções armazena o código (em bits) que define qual será a ação do processador ao executar essa instrução.

De acordo com Stallings (2010), essas ações podem ser organizadas em quatro categorias:

- Processador-memória: ação de transferência de dados do processador para a memória e vice-versa;
- Processador-ES: ação de transferência de dados entre processador e um dispositivo de E/S, através de um módulo de E/S;
- Processamento de dados: execução de operações lógicas ou aritméticas com os dados;
- Controle: determinadas instruções podem dizer que a sequência de execução das instruções seja alterada, alterando a sequência do contador de programa.

Como forma de melhorar a eficiência de processamento quase todos os computadores implementam um mecanismo de interrupções. Esse mecanismo permite que componentes, como E/S e memória, possam interromper a sequência normal de execução de instruções do processador.

Monteiro (2007) indica que uma interrupção consiste em uma série de procedimentos que suspendem o funcionamento corrente do processador, desviando sua atenção para outra atividade. A interrupção é uma forma válida para evitar o desperdício de tempo do processador para atender a dispositivos bem mais lentos, como uma impressora, por exemplo.

Saiba mais!

Vamos fixar o conteúdo de forma divertida?

Você já ouviu falar da saga do processador? Não? Pois bem. Trata-se de uma animação divertida que apresenta a atuação do processador dentro de um computador. A série possui vários episódios. Selecionamos o primeiro aqui para vocês. Indico que depois de assistir localize os demais episódios no YOUTUBE. Preste atenção aos conceitos apresentados na animação.

Acesse: https://www.youtube.com/watch?v=HTt3TVQpjrE

2.6 Classificações dos computadores

As classificações dos computadores são bem didáticas e não são únicas, porém são bastante interessantes para auxiliar a compreender suas características, particularmente, no momento de aquisição de um equipamento. Podemos classificar os computadores de diversas formas. As formas de classificação apresentadas aqui são apenas exemplos, pois podemos classificar um determinado computador de diversas maneiras: capacidade de processamento, velocidade de processamento e volume de transações, capacidade de armazenamento, tamanho de memória, entre outros. Vejamos algumas classificações possíveis:

a) Quanto à característica de construção

Conforme aprendemos na Unidade I - Histórico e evolução da Informática, os computadores podem ser divididos em gerações (primeira, segunda, terceira, quarta, quinta...) de acordo com suas características de construção (válvulas, transistores, circuitos integrados, microprocessador...).

b) Quanto ao princípio de construção (quanto à natureza)

Analógico - O computador analógico representa variáveis por meio de analogias físicas. Trata-se de uma classe de computadores que resolve problemas referentes a condições físicas, por meio de quantidades mecânicas ou elétricas, utilizando circuitos equivalentes como analogia ao fenômeno físico que está sendo tratado.

Como exemplos de computadores analógicos, podemos citar:

- O computador diferencial, um computador analógico mecânico projetado para resolver equações diferenciais por integração, usando mecanismos de engrenagem para realizar a integração. Inventado em 1876, foram construídos, primeiramente, entre as décadas de 1920 e 1930.

- As miras de bombas da segunda guerra mundial usavam computadores analógicos mecânicos.
- O computador MONIAC foi um modelo hidráulico de economia nacional construído no início da década de 1950.
- Heathkit EC-1 é um computador analógico educacional construído pela Heath Company, EUA, por volta de 1960.

Digital - Processa informações representadas por combinações de dados discretos ou descontínuos. Mais especificamente: trata-se de um dispositivo projetado para executar sequências de operações lógicas e aritméticas. São os nossos computadores atuais.

c) Quanto ao âmbito

Específico - Os computadores de âmbito específico são desenhados para desempenhar um conjunto muito reduzido de tarefas. São utilizados, por exemplo, no controle de mecanismos industriais e em cálculos científicos.

Geral - Os computadores de âmbito geral são capazes de desempenhar uma grande variedade de tarefas por meio da execução de um grande número de programas. Estes computadores são bastante utilizados em escritórios, escolas e em nossas casas.

d) Quanto à utilização

Científico - O computador científico é empregado em áreas de cálculos e pesquisas científicas, nas quais são requeridos resultados de maior precisão e pequeno volume de entrada e saída de dados.

Comercial - O computador comercial constitui a grande maioria dos equipamentos utilizados nas empresas; caracteriza-se por permitir o trato rápido e seguro de problemas que comportam grande volume de entrada e saída de dados.

Uso geral - São disponibilizados pela maioria dos fabricantes e comportam o emprego tanto na área científica quanto na área comercial.

e) Quanto ao Porte (porte, custo, desempenho e propósito)

Supercomputadores - São computadores com grande poder de processamento, sendo utilizados, principalmente, no campo científico, nos quais se destacam as simulações. Podem ser aplicados também na previsão de tempo e modelagem tridimensional. Estes computadores são de âmbito específico e realizam tarefas bastante específicas. Seu custo é muito elevado, possuem um tamanho muito grande e necessitam de condições especiais de funcionamento. Temos, como exemplos, o Sunway TaihuLight , o Santos Dumont e o IBM Blue Gene.

Figura 3 - Exemplos de Supercomputadores

Supercomputador Sunway TaihuLight
Fonte: Imagem da internet. Disponível em: https://brasil.elpais.com/brasil/2016/10/26/tecnologia/1477480235_164392.html
Acesso em: 01 de nov.2017.

Supercomputador Santos Dumont, do Laboratório Nacional de Computação Científica de Petrópolis (RJ).
Fonte: Imagem da internet. Disponível em: http://s2.glbimg.com/grNoEn-WO7tRse-wkb_tc8naewM=/s.glbimg.com/jo/g1/f/original/2015/11/18/santos-dumont-g1.jpg
Acesso em: 01 de nov.2017.

Mainframes - Também conhecidos como computadores de Grande Porte são sistemas projetados para manusear considerável volume de dados e executar simultaneamente programas de uma grande quantidade de usuários a partir de terminais.

Um mainframe possui a ele conectado uma grande quantidade de terminais que podem ser terminais burros (todo processamento é feito no computador principal - o mainframe - e o terminal apenas é um meio para entrar e visualizar dados) ou terminais inteligentes (este terminal faz parte do processamento nele mesmo e normalmente é um computador pessoal conectado ao mainframe).

Saiba mais!

Entenda o que são os supercomputadores!
O supercomputador Watson, da IBM, foi premiado como Personalidade do Ano pelo Webby Awards, conhecido como o "Oscar da internet".

O supercomputador da IBM venceu dois humanos em um programa de perguntas e respostas na TV americana, o Jeopardy. O Watson é composto por 90 computadores conectados e 16 TB de memória RAM. Mas você sabe o que são supercomputadores? Qual a capacidade de processamento dos mais poderosos e como vai o Brasil nessa área? Veja como funcionam essas supermáquinas que levam horas para fazer cálculos complexos que levariam anos em PCs comuns.

Para saber um pouco mais sobre as curiosidades dos supercomputadores, explore a seção de infográficos desenvolvida pelo portal Terra.

Acesse: https://www.terra.com.br/noticias/infograficos/supercomputadores/

O supercomputador mais rápido do mundo. Veja como funciona essa "divindade da supercomputação". Disponível em: <https://brasil.elpais.com/brasil/2016/10/26/tecnologia/1477480235_164392.html>. Acesso em: 01 de nov.2017.

Os Mainframes são utilizados por grandes empresas que necessitam armazenar grande quantidade de informação e ter um acesso rápido a essa informação. Ao contrário dos supercomputadores, estes são de âmbito amplo e podem ser utilizados em grandes organizações como bancos, companhias de seguros e centros de investigação. Como exemplo clássico de Mainframes, estão os equipamentos fabricados pela IBM (S/390, G5/G6, z800/ z900, System z9 e z10).

Minicomputadores - Os minicomputadores são resultado da redução de tamanho dos computadores. A miniaturização de componentes levou ao surgimento desta categoria de computadores de porte médio com grande capacidade de processamento e terminais conectados. São adequados para a realização de tarefas de controle de processos industriais e de gestão de sistemas multiusuário. O desenvolvimento da tecnologia dos processadores levou ao aparecimento dos microcomputadores, deixando a distinção entre as duas categorias cada vez menos clara. Como exemplo desta categoria, podemos citar os Sistemas AS/400/IBM.

Estações de trabalho -As estações de trabalho estão situadas logo abaixo dos minicomputadores. São equipamentos que normalmente possuem arquitetura RISC e sistema operacional UNIX. A capacidade de memória e velocidade de processamento de uma estação de trabalho se assemelha as de um minicomputador. Contudo, ao contrário dos minicomputadores, elas são empregadas por usuários, sendo projetadas para realizar tarefas “pesadas”, em geral, na área científica ou industrial, como cálculos matemáticos, projetos que utilizam processamento de imagem ou renderização 3D, entre outras finalidades mais complexas. Podemos citar como exemplos: Sun Workstation W2100z, Dell Precision T5600.

Computadores pessoais - Os computadores pessoais estão presentes em nosso dia a dia com diversos formatos, modelos e finalidades. Hoje até carregamos um computador pessoal na mão e ele também serve para telefonar. Brincadeiras a parte, os computadores pessoais (em inglês Personal Computers - PC) também são conhecidos como "microcomputadores" ou simplesmente "micro". Esse tipo de computador tem a característica marcante de ter um processador único (ainda que com vários núcleos) e sua capacidade de processamento evoluiu rapidamente com o passar do tempo. Aos poucos, devido a sua grande capacidade de processamento, foi tomando conta do mercado e desbancando outras versões de computadores como minicomputadores e workstations. Devido à redução de preços (custo dos equipamentos), o computador pessoal se tornou cada vez mais acessível, facilitando ainda mais o upgrade (atualização e ampliação do hardware) de sua capacidade computacional. A tendência atual é que cada vez mais o computador pessoal diminui em tamanho. Ele está presente em todos os tipos de empresas, em nossos lares, escolas e em quase todas as nossas atividades cotidianas.

Podemos ainda, considerando o tamanho físico do equipamento e seu grau de portabilidade, classificar os microcomputadores da seguinte forma:

Desktop

Computador de mesa. É o mais comum deles. Composto por monitor de vídeo, teclado, mouse, gabinete (CPU), caixas de som.

Figura 4 - Desktop

Fonte: Imagem disponível em: <https://images.app.goo.gl/GzS1dSCnA8Ut7HMx8>. Acesso em: 01 de nov.2017.

Notebook

Computador portátil e compacto, com todos os componentes integrados. Em geral, possui bateria. É mais caro. Capacidade similar ao Desktop.

Figura 5 - Notebook
Fonte: Imagem disponível em: <https://pixabay.com/pt/vectors/notebook-computador-laptop-neg%C3%B3cios-303161/>.

Palmtop

Cabe na palma da mão. Capacidade de processamento menor. Também conhecido como PDA (Personal Digital Assistants)

Figura 6 - Palmtop
Fonte: Imagem disponível em: <https://amcmuseum.org/wp-content/uploads/2015/06/palmtop-pc.jpg>.
Acesso em: 01 de nov.2017.

MiniPC

Os mini PCs são computadores suficientemente pequenos para ficar escondidos em uma gaveta. A limitação está na expansibilidade da configuração.

Figura7 - MiniPC
Fonte: Imagem disponível em: <https://pt.made-in-china.com/co_dajinsheng/product_Fashion-High-Quality-H81u-Support-Windows-System-Mini-PC_rssehnnog.html>.

Tablet, iPad

É um dispositivo pessoal em formato de prancheta que pode ser usado para acesso à internet, organização pessoal, visualização de fotos, vídeos, leitura de livros, jornais e revistas e para entretenimento com jogos.

Figura 8 - Tablet
Fonte: Imagem disponível em:<http://escolhasegura.com.br/wp-content/uploads/2015/01/Tablet-Sony-S.jpg>. Acesso em: 01 de nov.2017.

Smartphones

Um telefone inteligente (do termo em inglês) que combina recursos de um computador pessoal com um telefone com funcionalidades avançadas que podem ser estendidas por meio de aplicativos instalados.

Figura 9 - Smartphones
Fonte: Imagem disponível em: <https://www.mobilebit.com.br/noticias/2015/01/03/o-que-e-um-smartphone-diferenca-celular/>.

Saiba mais!

Quer saber um pouco mais sobre os Smartphones?

Para saber um pouco mais sobre as curiosidades dos smartphones no Brasil visite o site ShowMeTech e veja o infográfico “A evolução dos smartphones no Brasil”.

Acesse: https://www.showmetech.com.br/brasil-uma-historia-contada-por-celulares/

2.7 Software

O software é um conjunto flexível e modificável de instruções, ordenadas em sequência lógica, fornecidas ao hardware do computador para a execução de procedimentos necessários à solução de problemas e de tarefas do processamento de dados. É esse conjunto de códigos que torna possível que um computador genérico tenha uma variedade ilimitada de usos e aplicações. Um conjunto particular e específico destas instruções é chamado de programa.

Os programas são componentes de um software e normalmente estão armazenados em memórias secundárias (unidades de armazenamento). Os softwares e programas são desenvolvidos utilizando-se linguagens de programação. A Engenharia de Software é a área de estudo que se preocupa com o desenvolvimento de software.

Quando um computador está usando um programa em particular, dizemos que ele está rodando ou executando aquele programa.

a) Software básico

Na categoria de software básico podemos contemplar os sistemas operacionais e os softwares utilitários.

O sistema operacional é um software essencial, ou seja, um conjunto de rotinas que são executadas pelo processador para facilitar o acesso aos componentes de hardware (processador, memória, dispositivos de E/S), e gerenciar o uso do sistema de computação (hardware e software). Tradicionalmente, os sistemas operacionais eram escritos em linguagem Assembly. Os sistemas operacionais mais atuais como Windows, GNU/Linux e MacOs são escritos em linguagens de alto nível, como a linguagem C e C++.

Os softwares utilitários são programas que preenchem a lacuna entre a funcionalidade de um sistema operacional e as necessidades dos usuários. Para muitos usuários, um computador com um sistema operacional e apenas aplicações básicas é inconveniente. Os programas utilitários introduzem ao sistema operacional funcionalidade que ele não possui.

b) Software aplicativo ou Sistema Aplicativo

Os aplicativos, como popularmente são conhecidos, são um conjunto de programas de computador desenvolvidos em linguagem de programação para realizar, em combinação com atividade humana, tarefas ou processos relacionados com o processamento de informações que atenda uma finalidade específica. Os softwares aplicativos abrangem diversas áreas do conhecimento e podem ser classificados de acordo com o tipo de tarefa que executam:

- Administrativos

  Relacionamento com clientes (Customer Relationship Management - CRM), sistemas de faturamento, contas a pagar, folha de pagamento, controle de estoque, controle da produção, contabilidade e outros.

- Tecno-Científicos

  Cálculo de estruturas, simulação, planejamento e controle de projetos, pesquisa operacional, problemas de engenharia, desenvolvimento de projetos, CAD e outros relacionados com atividades científicas ou de engenharia.

- Automação Industrial

  Controle de processos, telemetria, controle de fabricação, CAM e outros relacionados com atividades industriais.

- Automação Comercial

  Reserva de passagens, contas correntes, pontos de venda e outros relacionados com atividades comerciais.

- Apoio Educacional

  Assistência à instrução, ensino auxiliado pelo computador e outras atividades relacionadas ao ensino.

- Especiais e Científicos

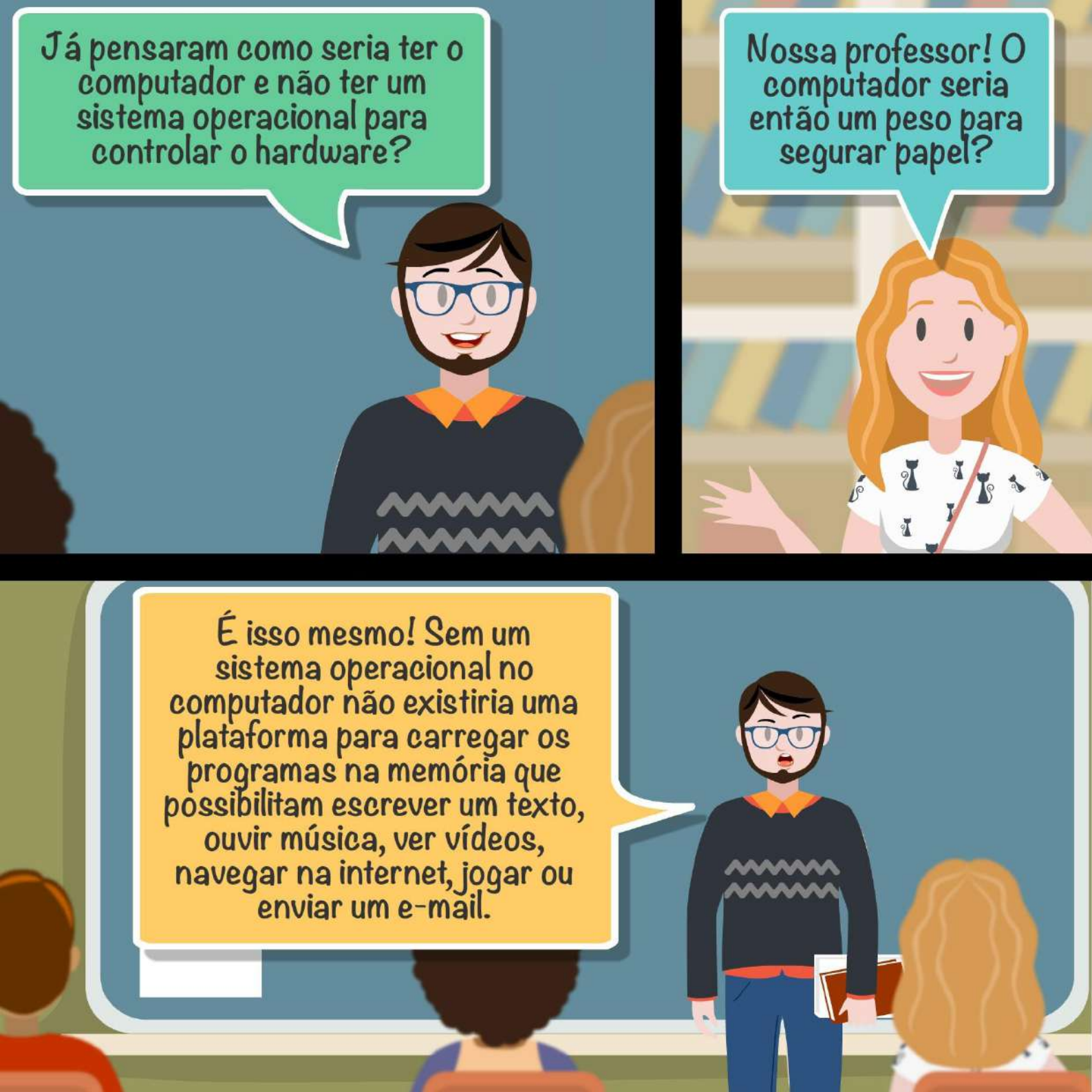
  Teleprocessamento, comunicações militares, pesquisas espaciais, previsões meteorológicas e outros.

Como exemplos de aplicativos podemos citar:

- Processador (ou editor) de textos (Word ou LibreOffice Writer)
- Planilha eletrônica (Excel ou LibreOffice Calc)
- Editor de apresentações (PowerPoint ou LibreOffice Impress)
- Editor de Imagem (Adobe Photoshop ou Gimp)
- Editoração Eletrônica (Corel Draw ou Inkscape)
- Matemática (Mathcad ou LibreOffice Math)
- Engenharia e Arquitetura (AutoCAD, 3D Studio ou QCad)

2.8 Sistemas Operacionais

O software mais importante de um computador é o sistema operacional. Quando executado é ele quem dá a possibilidade de usarmos e controlarmos o hardware. O Sistema operacional é importante, pois permite que o usuário interaja e, literalmente, dê ordens ao computador. Ele funciona como uma interface entre o usuário e o hardware e, por meio dele, interagimos com os periféricos de entrada e saída. Um computador sem sistema operacional é inútil.


Já pensaram como seria ter o computador e não ter um sistema operacional para controlar o hardware?
Nossa professor! O computador seria então um peso para segurar papel?
É isso mesmo! Sem um sistema operacional no computador não existiria uma plataforma para carregar os programas na memória que possibilitam escrever um texto, ouvir música, ver vídeos, navegar na internet, jogar ou enviar um e-mail.

a) Qual a finalidade de um sistema operacional?

Você deve estar se perguntando: o que ele faz afinal? O Sistema operacional é responsável por todos os recursos do computador. Ele cuida de todo software que o usuário quer executar e de todo o hardware que o usuário deseja utilizar no computador. De forma resumida, ele administra o computador para você. Quando você conecta um pendrive, por exemplo, é o sistema operacional que faz toda "a mágica" de disponibilizar os arquivos para você. Entendeu?

Quando você pressiona o botão Ligar do seu computador, ele realiza uma série de testes para garantir que tudo irá funcionar corretamente e se certifica de que os componentes físicos estejam disponíveis. Depois dessa verificação, o computador dá início a execução do sistema operacional.

O computador, normalmente, vem de fábrica com um sistema operacional previamente instalado. Na maioria das vezes, o usuário não faz modificações nele. No entanto, é possível atualizar o sistema operacional, por questões de segurança, ou ainda, caso o usuário tenha interesse fazer a troca do sistema operacional. Outra opção é rodar dois sistemas operacionais ao mesmo tempo, um dentro do outro, por meio de um conceito chamado virtualização. Há também a possibilidade de ter duas opções de sistemas operacionais diferentes no mesmo computador e, neste caso, ao ligar o equipamento, o usuário escolhe qual deles irá rodar, porém um de cada vez. Esse processo é chamado "Dual Boot".

Os sistemas operacionais atuais utilizam interfaces gráficas do utilizador (GUI) bastante amigáveis. Elas permitem que o usuário utilize o mouse para mover um ponteiro (indicador) e clicar sobre ícones, botões ou interagir com qualquer elemento ou objeto na tela, executando tarefas ou ações. Em alguns casos é possível comandar o indicador com o próprio dedo em uma tela sensível ao toque, fazendo as mesmas tarefas do mouse. É desta forma que o usuário dá ordens ao computador.

O mercado atual de computadores desktop permite ao usuário escolher diversos sistemas operacionais. As opções mais comuns são: Microsoft Windows, Mac OS X e Linux.

Cada sistema operacional utiliza uma interface gráfica do utilizador (GUI) de aparência diferente, portanto, se você mudar de um sistema para outro, provavelmente, no início, estranhará um pouco. Contudo, não se preocupe, pois todos eles são desenhados para serem fáceis de usar. Os princípios básicos são os mesmos em qualquer sistema operacional.

b) Microsoft Windows

O sistema operacional Windows foi desenvolvido pela Microsoft em meados da década de 1980. Desde então, diversas versões desse sistema operacional foram desenvolvidas. Atualmente as versões mais utilizadas do Windows são a 7, 8 e 10.

Devido a acordos comerciais com fabricantes de hardware, o Windows é o sistema operacional que vem instalado na maioria dos novos computadores, tornando-o o sistema operacional mais popular. É também o que apresenta maiores vulnerabilidades aos ataques de pragas virtuais, uma vez que é o que está presente em grande parte dos computadores.

Vamos refletir?

Antes da interface gráfica do utilizador (GUI) ser criada, as pessoas usavam comandos no teclado para poder se comunicar com o computador.

Por outro lado, em servidores, principalmente os que estão hoje funcionando em nuvem (cloud), é muito comum acessá-los remotamente via rede apenas utilizando o teclado, ou seja, sem interface gráfica.

Figura 10 - Sistema operacional Windows 10

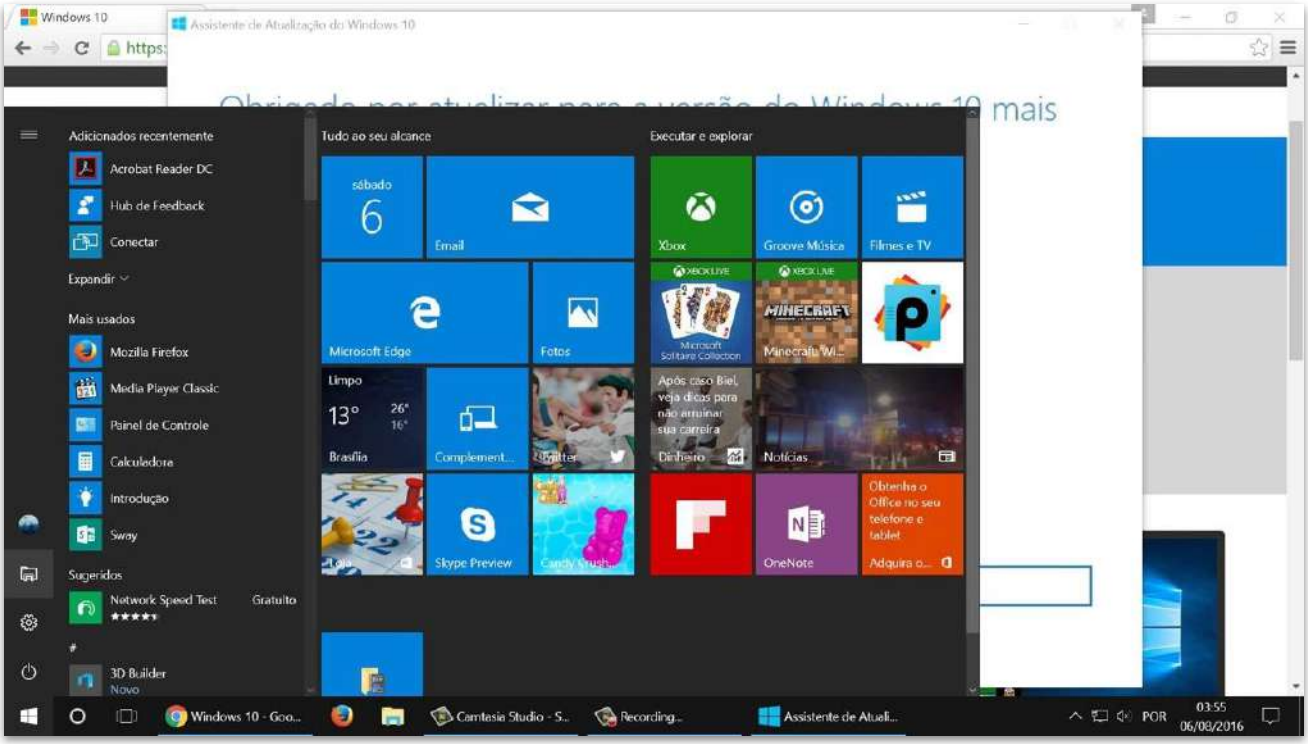


Imagem disponível em: <https://i.ytimg.com/vi/WTssvhVxtgQ/maxresdefault.jpg>. Acesso em: 01 de nov.2017.

c) Mac OS X

O Sistema Operacional Mac OS X foi desenvolvido pela Apple Inc. Todos os computadores Mac, fabricados pela Apple, vêm com ele instalado. As versões recentes são conhecidas como macOS. O macOS v10.12 "Sierra" é o décimo terceiro sistema operacional da família macOS, sendo o sucessor do OSX v10.11 "El Capitan". Anunciado em junho de 2016, traz, além da mudança no nome, cujo objetivo foi padronizar com os demais sistemas da Apple (iOS, tvOS e watchOS), melhorias nos recursos de continuidade, função picture in picture, abas para todas as janelas de apps e integração com a assistente virtual Siri.

Figura 11 - macOS v10.12 “Sierra”

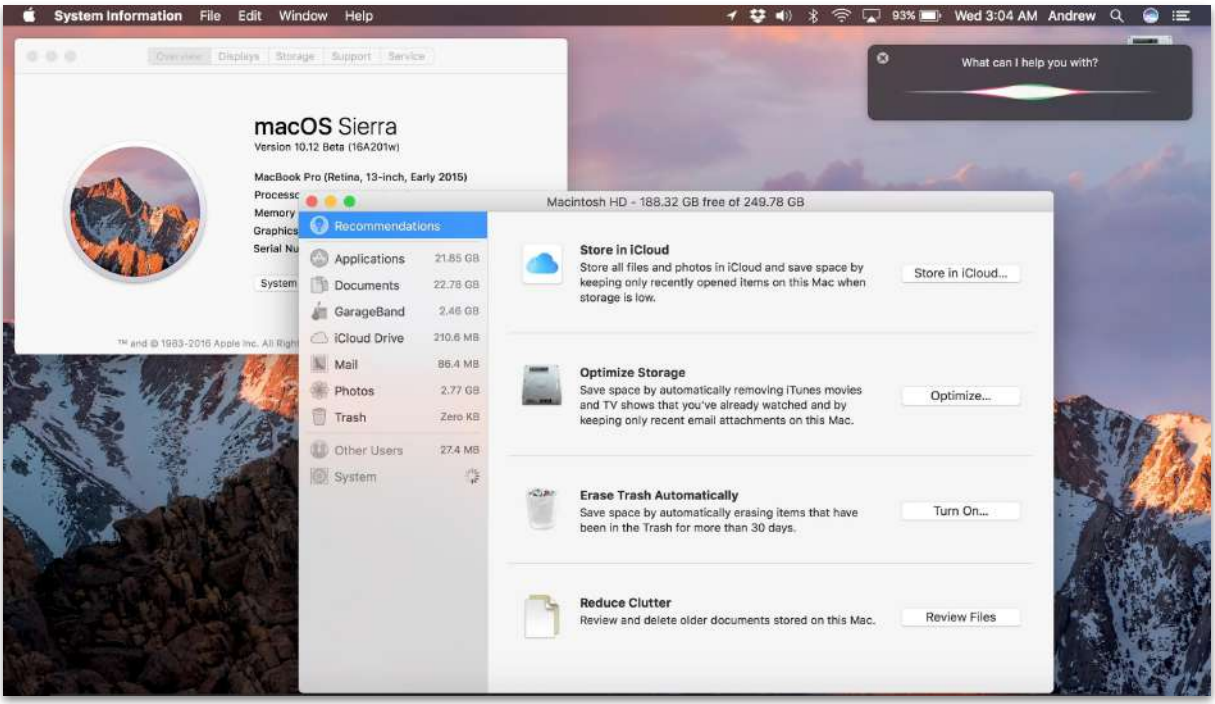


Imagem disponível em: https://cdn.arstechnica.net/wp-content/uploads/2016/09/sierra-1.jpg. Acesso em: 01 de nov.2017.

d) GNU / Linux

O sistema operacional GNU / Linux tem uma fantástica história de criação. Ele está relacionado diretamente com o surgimento do conceito do Software Livre idealizado por Richard Stalman e de sua vontade de criar um sistema operacional livre. Stalman começou a criar o sistema pelos softwares. Em seu primeiro passo, criou o compilador GCC para a linguagem C e outros softwares. Sua proposta era desenvolver depois, em um segundo momento, o Kernel do Sistema Operacional chamado Hurd. O destino se encarregou de fazer o resto.

Tomando por base os softwares desenvolvidos por Stalman, um jovem estudante de computação Finlandês chamado Linus Torvald resolveu também criar um sistema operacional, mas começou pelo Kernel. Porém ele usaria os softwares desenvolvidos por Stalman para testar seu Kernel e, assim, surgiu a primeira versão do GNU / Linux.

Esse sistema operacional toma por base o uso de código aberto. Significa que ele pode ser estudado, modificado e redistribuído por qualquer pessoa que tenha esse interesse em qualquer lugar do mundo. O GNU/Linux é como uma sorveteria em que se pode escolher diversos sabores. Cada sabor de sorvete é uma distribuição diferente do GNU/Linux. Todos tem acesso à receita de como o sorvete foi feito. Todos podem modificar e distribuir a receita sem pagar nada por isso. Qualquer usuário pode escolher sua distribuição, ou seja, pode escolher o sorvete que mais lhe agrada.

As versões mais populares são Ubuntu, Debian, Linux Mint e Fedora.

Figura 12 - Ubuntu 17.10

Imagem disponível em: <https://pplware.sapo.pt/wp-content/uploads/2017/09/ubuntu-17-10.jpg>. Acesso em: 01 de nov.2017.

Saiba mais!

O que acha de conhecer um pouco mais sobre o universo dos sistemas operacionais livres?

Para saber um pouco mais sobre como instalar um Windows 10 e um Ubuntu no mesmo computador, convido vocês a acessar esse vídeo que selecionei. É o canal do YOUTUBE chamado DIOLINUX que trata sobre Software Livre e GNU/Linux.

Acesse: https://www.youtube.com/watch?v=-3YnsQ5-qjw

Assim terminamos nossa Unidade II - A arquitetura de um computador. Espero que tenham compreendido bem o assunto. Lembrem-se de aprofundar as pesquisas e leituras. Neste capítulo aprendemos os conceitos que envolvem a arquitetura de um computador. Foi possível compreender o que é e como um computador pode ser utilizado. Verificamos como ocorre o processamento e transformação de um dado em informação. Depois vimos o hardware, o software, a arquitetura e a composição de um computador. Entendemos como a UCP, a memória, o barramento e os dispositivos de E/S se relacionam. Foi possível compreender a importância do processamento de instruções. Observamos a classificação dos computadores e softwares e finalmente mergulhamos nos sistemas operacionais. Aproveite para consultar o glossário se tiver alguma dúvida de algum termo. Vamos partir para algumas questões de aprendizagem e para a revisão final da unidade. Não esqueça de olhar os sites e vídeos indicados que irão ampliar os seus horizontes de conhecimentos sobre o tema da unidade.

Vamos rever?

Os computadores são utilizados em áreas como educação e pesquisa e, da mesma forma, para divulgar notícias, enviar mensagens de amigos e familiares, criar apresentações, manter registros pessoais e profissionais e realizar previsões meteorológicas, bem como para uma variedade de atividades de lazer ou de negócios.

Podemos definir um computador como um conjunto de dispositivos, interligados entre si, capazes de realizar um tratamento automático das informações seguindo as instruções de um programa.

O conjunto de elementos físicos que compõem um computador é chamado de HARDWARE.

Os programas que os computadores utilizam são chamados de SOFTWARE.

Um Sistema Computacional pode ser compreendido como a interação existente entre os diversos componentes de hardware, software e peopleware trabalhando conjuntamente sobre um determinado conjunto de dados produzindo informações e resultados de interesse para outros sistemas ou usuários.

Os computadores manipulam dados (entrada e processamento) para produzir informações (saída) e seguram (armazenam) essas informações para uso futuro.

Vamos rever?

A arquitetura de Von Neumann é um modelo de computador composto pela Unidade Central de Processamento (CPU), pela memória e pelos dispositivos de entrada e saída, todos conectados entre si pelo barramento.

Esse modelo deu origem ao conceito de armazenamento de código (programa gravado e reutilizável). O processador realiza o trabalho efetivo de executar as instruções especificadas no programa que foi carregado na memória do computador.

Os dispositivos de entrada e saída (E/S) são responsáveis pela comunicação com os usuários e o mundo externo.

É na memória principal que os dados são armazenados. Eles são transferidos por meio dos barramentos que interligam os demais componentes sempre que necessários.

A CPU divide-se em: Unidade de Controle (UC), necessária ao sincronismo da tarefa; ULA (Unidade Lógica e Aritmética), responsável pelos cálculos e pelos registradores para armazenar temporariamente os resultados.

O processador busca uma instrução na memória a cada início de ciclo de instrução.

O software mais importante de um computador é o sistema operacional. Quando executado é ele quem dá a possibilidade de usarmos e controlarmos o hardware.

Saiba mais!

Sites indicados:

1. Intel Chips Timeline - Disponível em: https://www.intel.com.br/content/www/br/pt/history/history-intel-chips-timeline-poster.html
2. Diolinux - Open Source, Ubuntu, Android e tecnologia Disponível em: http://www.diolinux.com.br/
3. DistroWatch.com: Put the fun back into computing. Use Linux, BSD. (DistroWatch.com: volte para a computação. Use Linux, BSD.) - Disponível em: https://distrowatch.com/?language=PT
4. Distribuição GNU/Linux Duzeru - Disponível em: http://duzeru.org/
5. Distribuição GNU/Linux Ubuntu - Disponível em: https://www.ubuntu.com/

Saiba mais!

Audiovisuais indicados:

1. Como surgiu e como funciona o Computador

   Disponível em: https://www.youtube.com/watch?v=rr6P9bsjFzc

2. Série Bits e Bytes - 04 - Os arquivos de computador

   Disponível em: https://www.youtube.com/watch?v=P3HrAY2fQVc

3. Processador (CPU) - O que é?

   Disponível em: https://www.youtube.com/watch?v=zzx5p_VGf44

Questões de autoaprendizagem:

a) Faça uma pesquisa e defina com as suas palavras o que é um sistema computacional.

b) Descreva as características de cada uma das partes que constituem esse sistema computacional.

c) Apresente as diferenças entre um computador digital e um computador analógico. Dê exemplos. Atualmente usamos computadores digitais? Explique.

d) Qual o significado das descobertas e das proposições feitas por John Von Neumann para a computação?

e) Explique como o modelo de arquitetura para computadores proposto John Von Neumann se relaciona com os computadores atuais.

Questões de autoaprendizagem:

Vamos verificar se você compreendeu os conceitos principais relacionados à Arquitetura de Computadores? Relacione as colunas abaixo:

(1) Processador.

(2) Registrador de endereçamento de memória.

(3) Registrador de endereço de E/S.

(4) Local onde se armazena dados de leitura e escrita.

(5) É utilizado para interligar os componentes do modelo de Neumann.

(6) Registrador temporário de dados de E/S.

(7) Arquitetura que utiliza o conceito de programa armazenado.

(8) Interrupção de E/S.

(9) Interrupções.

(10) Ciclo de instrução.

(11) Contador de programa.

( ) Alternativa válida para evitar o desperdício de tempo do processador com dispositivos lentos.

( ) Todo processamento necessário para o ciclo de execução.

( ) Guarda o endereço da próxima instrução a ser buscada na memória.

( ) Gerada por um controlador de E/S para sinalizar a conclusão de uma operação.

( ) Barramento.

( ) Contém o endereço a ser usado para a próxima instrução de leitura e escrita.

( ) Memória principal.

( ) Utiliza, basicamente, dois registradores para trocar dados com a memória.

( ) Especifica um determinado dispositivo de E/S

( ) Modelo de Von Neumann.

( ) Utilizado para trocar dados entre o módulo de E/S e a CPU.

Questões de autoaprendizagem

Pesquisa

Atividade de pesquisa - realizar uma pesquisa sobre:

1) Tipos de arquivos, organização e operações com arquivos em sistemas operacionais Windows 10, Ubuntu 17.10 e macOS High Sierra;

2) Periféricos de entrada e saída para computadores;

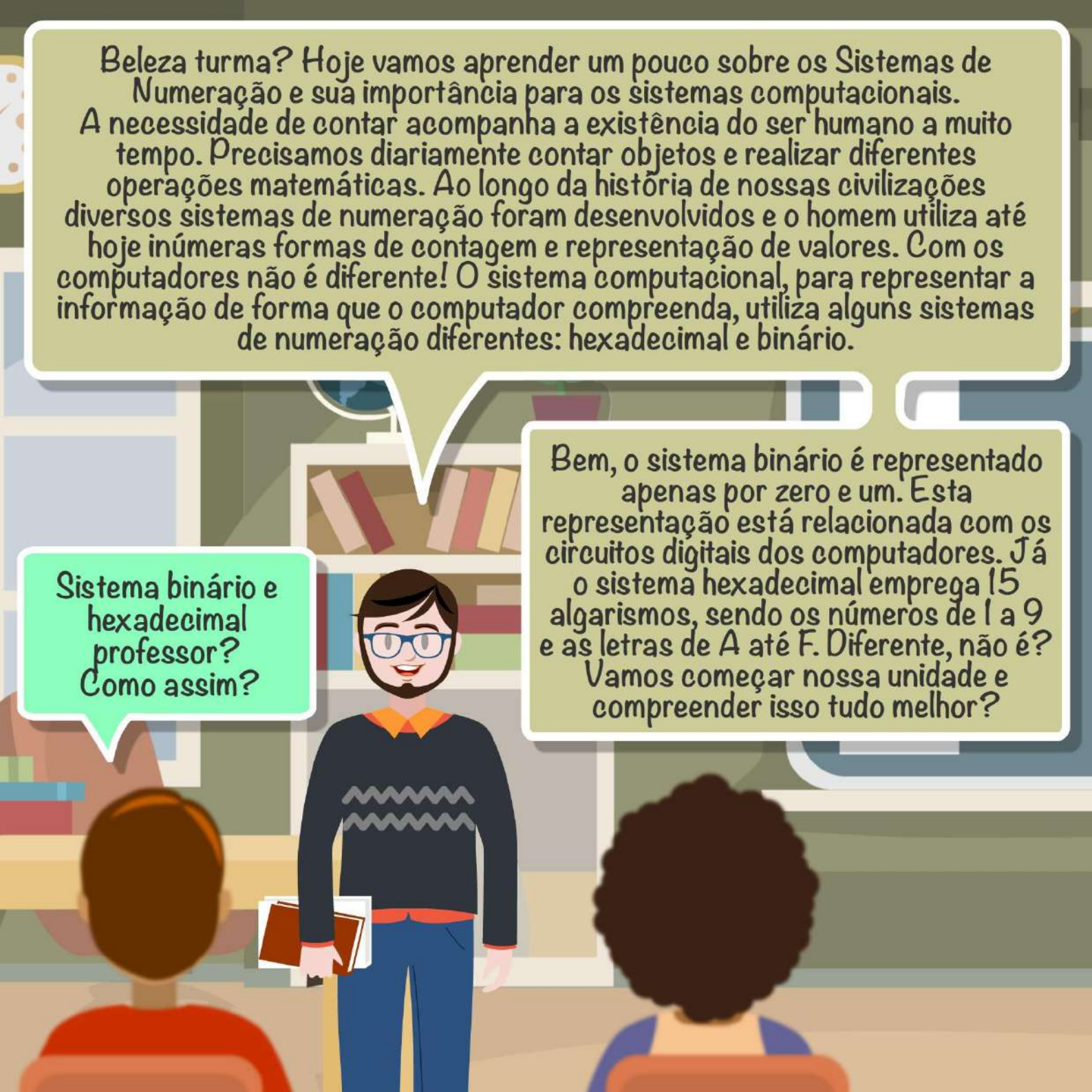
3) História dos processadores - Criar uma linha do tempo com pelo menos 15 processadores diferentes.

Regras sugeridas para a Pesquisa

- Deve apresentar as referências, ou seja, a(s) fonte(s) de onde foram retiradas as informações da pesquisa;
- Deve conter um último tópico chamado Conclusão, com pelo menos um e no máximo três parágrafos apresentando o seu entendimento sobre o assunto;
- As fontes da pesquisa deverão ser os sites da internet;

Unidade 3 Sistemas de Numeração

Nesta unidade, denominada Sistemas de Numeração, vamos compreender os sistemas de numeração, suas bases, representações e processos de conversão de bases.
Ao término desta unidade esperamos atingir o seguinte objetivo:
entender e lidar com o sistema de numeração existente nos computadores.


Beleza turma? Hoje vamos aprender um pouco sobre os Sistemas de Numeração e sua importância para os sistemas computacionais. A necessidade de contar acompanha a existência do ser humano a muito tempo. Precisamos diariamente contar objetos e realizar diferentes operações matemáticas. Ao longo da história de nossas civilizações diversos sistemas de numeração foram desenvolvidos e o homem utiliza até hoje inúmeras formas de contagem e representação de valores. Com os computadores não é diferente! O sistema computacional, para representar a informação de forma que o computador compreenda, utiliza alguns sistemas de numeração diferentes: hexadecimal e binário.
Sistema binário e hexadecimal professor? Como assim?
Bem, o sistema binário é representado apenas por zero e um. Esta representação está relacionada com os circuitos digitais dos computadores. Já o sistema hexadecimal emprega 15 algarismos, sendo os números de 1 a 9 e as letras de A até F. Diferente, não é? Vamos começar nossa unidade e compreender isso tudo melhor?

3.1 Como tudo começou?

O termo *digital* deriva de *dígito*, que por sua vez tem origem no latim *digitus* que significa dedo. Assim, ao longo de sua evolução a humanidade desenvolveu o processo de contagem e os dedos sempre foram os instrumentos mais eficientes e simples para a contagem de pequenos valores. O sistema de numeração decimal, o mais usado atualmente, é um sistema de base dez, pois são dez os dedos das duas mãos dos seres humanos.

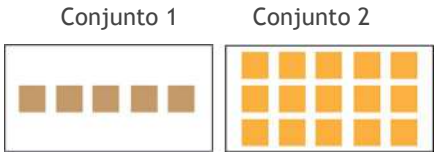
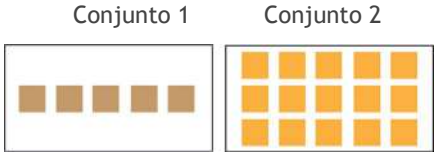
Figura 1 - Sistema de numeração decimal

Fonte: Imagem retirada da internet. Disponível em: <http://2.bp.blogspot.com/-_JJeIB8MbdI/VRWvzrDBHNI/AAAAAAAABrs/jHj4POJ9fsA/s1600/DEDOS.jpg>. Acesso em: 04 de nov.2017.

Fica fácil compreender o motivo do sistema decimal que empregamos ter a base 10, afinal temos dez dedos na mão e podemos contar com eles.

Os sistemas de numeração nasceram com a evolução da civilização, com o objetivo de atender a necessidade de registrar informações sobre quantidades. Vamos refletir sobre as representações numéricas. Acompanhe o raciocínio na tabela 1 abaixo.

Tabela 1 - Representações Numéricas

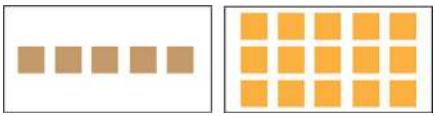
1	2	3
As representações numéricas podem ser feitas por números e por quantidades (conjuntos).	Sabemos a quantidade de elementos de um conjunto pela contagem. Podemos comparar as quantidades mesmo sem nomeá-las (sem representar por números). 	Os números são representações convenientes para as quantidades. 

Fonte: elaborada pelo autor

Mas esta é a única forma de representação numérica? Claro que não! Não é a única e tão pouco foi a primeira inventada pelo ser humano. Quando falamos em nosso sistema de numeração decimal, estamos nos referindo ao sistema de representação decimal com números hindu-arábicos.

Porém, existem outras formas de representação como, por exemplo, o sistema de numeração romano. Eles utilizavam letras para representar quantidades. Vamos ver, na tabela 2, esta representação.

Tabela 2 - Sistema de representação romano

4	5
A mesma representação do exemplo anterior utilizando números romanos. Conjunto 1 Conjunto 2  O conjunto 1 tem V elementos. O conjunto 2 tem XV elementos.	A representação utilizada pelos romanos era diferente. eles utilizavam letras para representar quantidades. Não tinham um símbolo para representar o zero.

1	I	30	XXX
2	II	40	XL
3	III	50	L
4	IV	100	C
5	V	500	D
6	VI	1000	M
7	VII	2000	MM
8	VIII	3000	MMM
9	IX	4000	$\overline{IV}$
10	X	5000	$\overline{V}$
20	XX	10000	$\overline{X}$

Fonte: elaborada pelo autor

Os computadores têm como base para seu funcionamento a utilização de eletricidade. No caso dos computadores, os números servem para representar os circuitos ou fios. Pense da seguinte forma. Um fio, ou circuito, pode ter dois estados:

a) passa corrente (ligado)
b) não passa corrente (desligado)

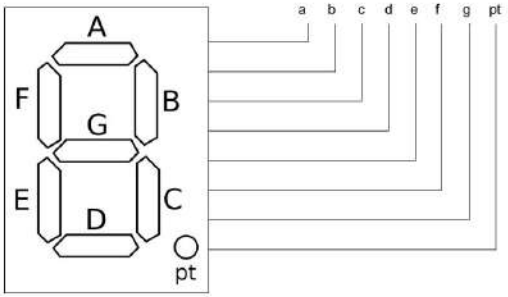
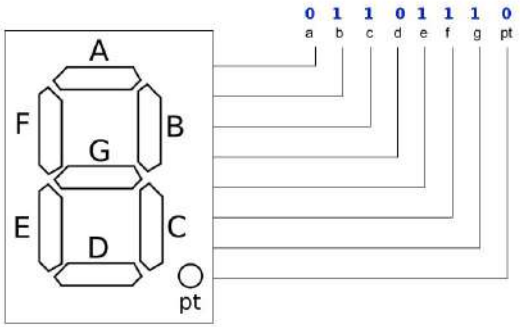
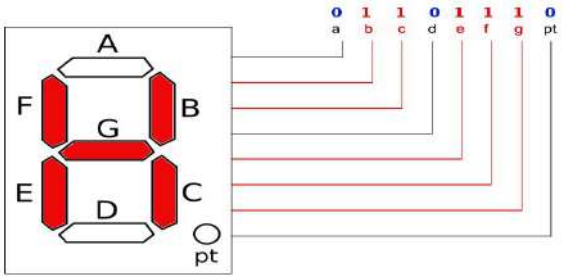
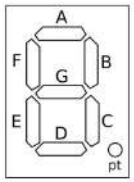
Quando pensamos em circuitos digitais de computadores, normalmente utilizaremos a representação binária, na qual cada dígito binário é chamado de bit e representa um fio ou circuito.

Diferente de outras máquinas em que a presença ou a ausência de eletricidade apenas significa se estão ligadas ou desligadas, um computador deve utilizá-la para manipular e armazenar informações.

Assim, em um computador, trabalhamos com sinais elétricos que são representados por 0 e 1.

Esses números formados com zeros e uns tem qual significado? Bem, isso depende da interpretação e da convenção. Observe o exemplo a seguir, na tabela 3.

Tabela 3 - Representação de circuito elétrico

1	2
O que representa o número 01101110b, por exemplo? A resposta mais prudente, é: depende. No caso de um display de LEDs, por exemplo, poderia ser um caractere. Vamos considerar o seguinte display. 	Colocamos o número sobre os circuitos do display, representados pelas letras e pelo (pt), que seria o fio terra para fechar o circuito. 
3	**4**
Vamos ligar todos os circuitos em que temos o número 1, na sequência de números do nosso exemplo. Temos a letra H demonstrada em nosso display. 	Seguindo o mesmo raciocínio, podemos representar outros caracteres no display, conforme a tabela abaixo. 

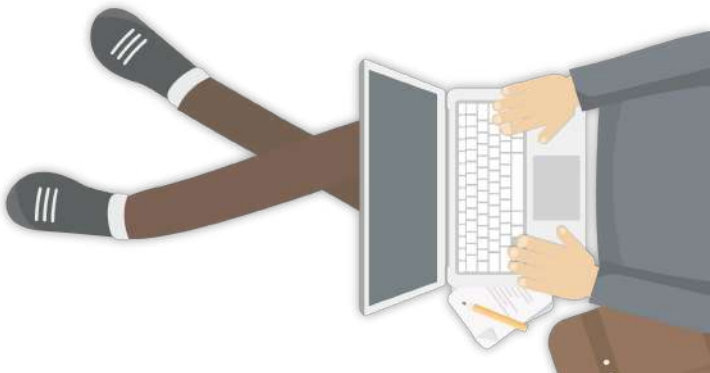
a	b	c	d	e	f	g	Led
1	1	1	1	1	1	0	0
0	1	1	0	0	0	0	1
1	1	0	1	1	0	1	2
1	1	1	1	0	0	1	3
0	1	1	0	0	1	1	5
1	0	1	1	0	1	1	5
1	0	1	1	1	1	1	6
1	1	1	0	0	0	0	7
1	1	1	1	1	1	1	8
1	1	1	1	0	1	1	9

Fonte: elaborada pelo autor

A tabela 3 traz apenas um exemplo de representação empregado para indicar caracteres em um display, mas e para computadores? Vejamos a seguir.

a) Representação dos valores lógicos

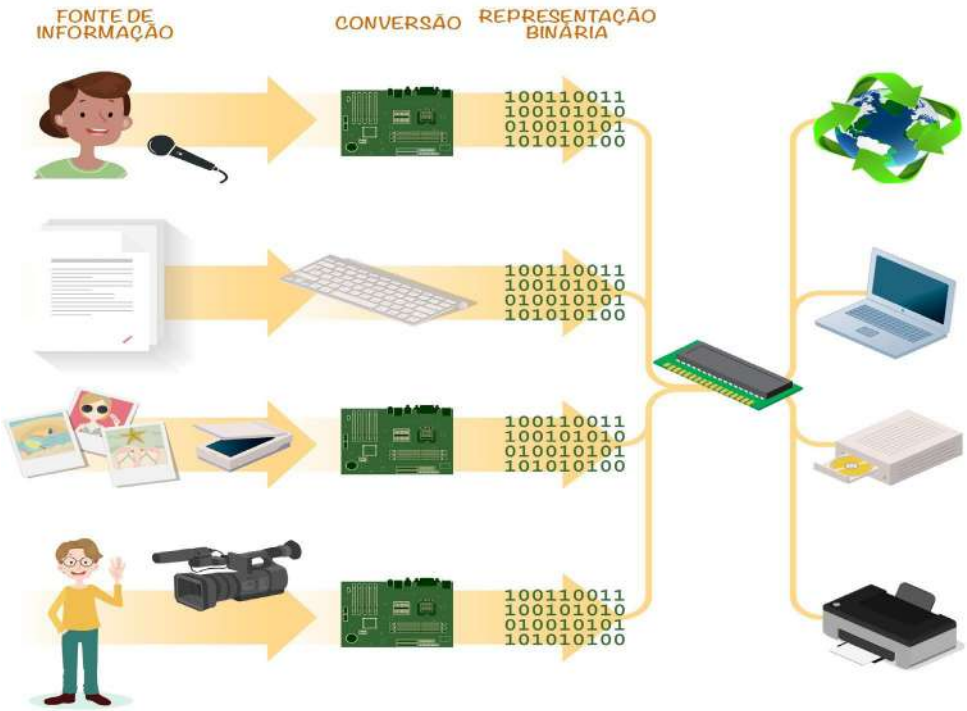
Como estudamos na unidade anterior, o computador é um sistema (máquina) capaz de sistematicamente coletar (receber em sua entrada), manipular (processar) e fornecer resultados (mostrar dados em suas saídas). Processar dados representa uma série de atividades realizadas em ordem com objetivo de gerar um conjunto determinado de resultados e informações. Um computador, de acordo com a arquitetura de Von Neumann, é formado pela Unidade Central de Processamento (CPU), Memória e Dispositivos de E/S que, integrados segundo uma estrutura de sistemas, compõem um conjunto de componentes que manipula dados na forma digital (0s e 1s), com base de numeração binária. O computador é uma máquina binária capaz de compreender apenas dois valores diferentes (0 e 1) que representam o SIM/NÃO, ABERTO/FECHADO, ACIMA/ABAIXO, LIGADO/DESLIGADO (lógica booleana).



b) Representação de dados

Tudo para um computador se transforma em números. Dentro de um computador, a informação é armazenada, transferida e manipulada em grupos de bits. Assim, toda informação que introduzimos em um computador (sejam dados que serão processados ou instruções de um programa) precisa ser compreendida pelo computador, para que ele tenha a capacidade de reconhecê-la e processá-la.

Figura 2 - Representação da informação no computador



Fonte: adaptada da internet pelo autor.

Representar informações apenas com 0 e 1 é algo bastante limitado. Assim os bits podem ser agrupados para ter um significado útil. O byte é o menor grupo ordenado de bits que representa uma informação útil e inteligível para o ser humano. A palavra de um computador

pode ser compreendida como o tamanho da capacidade de armazenamento e transferência de informação (na memória e CPU), ou como sendo um conjunto de bits que representa uma informação útil naquele sistema computacional.

Os dados em um computador são representados na forma binária:
Bit (1 dígito binário) ou Binary Digit: valor 0 ou 1
Byte = 8 bits
Caracter = conjunto de n bits que define 2n caracteres.
Palavra = conforme a arquitetura ocupará n bytes.

Por meio da combinação de bits, através de um código, evidentemente, pode-se chegar a representações variadas. Desta forma, foram criados padrões para representar dados em computadores.

A representação de caracteres pode ser feita por meio de um CSS (Conjunto de Caracteres Codificados - "Coded Character Set" ou "Character Encoding"). O CCS é definido como um mapa de um conjunto abstrato de caracteres para um conjunto de valores inteiros não negativos. Esses conjuntos de "Character Encoding" são regulamentados por meio de normas ISO.

Tabela 4 - Normas para representação e mapeamento de caracteres codificados

Nome/Norma	Repertório (conjunto de caracteres)
ISO/IEC 8859-1	ASCII plus Latin - 1
The Unicode Standard, Version 4.0 ISO/IEC 10646:2003	Mesmo repertório e mapeamento do ASCII

Fonte: elaborada pelo autor

Podemos citar como exemplo o Código ASCII do inglês, “American Standard Code for Information Interchange” (Código Padrão Americano para o Intercâmbio de Informação) de 7 posições (bits) que compreendem a representação de 96 caracteres voltados para leitura/ impressão e mais 32 para controle, sem nenhuma representação gráfica.

Posteriormente, o código ASCII foi ampliado para representar também um conjunto maior de símbolos (8 bits), como forma de abranger outros idiomas além do inglês que passaram a utilizar essa codificação.

Observe a tabela ASCII apresentada na Figura 3. A tabela apresenta 256 caracteres (de 0 até 255). O primeiro quadro da tabela são os caracteres de controle (ASCII control caracteres). No segundo quadro, temos os caracteres imprimíveis (de 32 até 126). No terceiro quadro, a extensão do código ASCII para atender outros idiomas (de 128 até 255).

Os três quadros da tabela são compostos por três colunas que demonstram a representação de um mesmo código ASCII em dois formatos diferentes (sistemas numéricos diferentes): decimal e hexadecimal.

É claro que existem outras codificações utilizadas como o sistema UNICODE Standard que permite a distinção entre glifos e caracteres, sendo importantes para idiomas como árabe, hebreu, idiomas orientais e idiomas com acentuação.

Figura 3 - Tabela de codificação ASCII

Binário	Decimal	Hexa	Glifo	Binário	Decimal	Hexa	Glifo	Binário	Decimal	Hexa	Glifo
0010 0000	32	20		0100 0000	64	40	@	0110 0000	96	60	`
0010 0001	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a
0010 0010	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
0010 0011	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
0010 0100	36	24	$	0100 0100	68	44	D	0110 0100	100	64	d
0010 0101	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
0010 0110	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
0010 0111	39	27	'	0100 0111	71	47	G	0110 0111	103	67	g
0010 1000	40	28	(	0100 1000	72	48	H	0110 1000	104	68	h
0010 1001	41	29	)	0100 1001	73	49	I	0110 1001	105	69	i
0010 1010	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
0010 1011	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k
0010 1100	44	2C	,	0100 1100	76	4C	L	0110 1100	108	6C	l
0010 1101	45	2D	-	0100 1101	77	4D	M	0110 1101	109	6D	m
0010 1110	46	2E	.	0100 1110	78	4E	N	0110 1110	110	6E	n
0010 1111	47	2F	/	0100 1111	79	4F	O	0110 1111	111	6F	o
0011 0000	48	30	0	0101 0000	80	50	P	0111 0000	112	70	p
0011 0001	49	31	1	0101 0001	81	51	Q	0111 0001	113	71	q
0011 0010	50	32	2	0101 0010	82	52	R	0111 0010	114	72	r

Fonte: imagem retirada da internet. Disponível em: <https://www.marciocunha.eti.br/tabela-codigo-ascii/>.

Dentro deste mesmo contexto, as memórias armazenam, acessam e recuperam informações byte a byte. Assim, existem unidades de medida para representar o tamanho das memórias sejam principais (RAM) ou secundárias (discos rígidos, cds, pendrives, etc.). Veja a Figura 4 com a representação dessas unidades de medida.

Figura 4 - Unidades de medida de bits/bytes

Unidade	Símbolo	Valor Equivalente	Múltiplo
Bit	b*		
Byte	B*	8 bits	10^0
Kilobyte	KB	1024 B	10^3
Megabyte	MB	1024 KB	10^6
Gigabyte	GB	1024 MB	10^9
Terabyte	TB	1024 GB	10^{12}
Petabyte	PB	1024 TB	10^{15}
Exabyte	EB	1024 PB	10^{18}
Zettabyte	ZB	1024 EB	10^{21}
Yottabyte	YB	1024 ZB	10^{24}

Fonte: elaborada pelo autor

Lembre-se: O computador armazena informações utilizando o sistema hexadecimal (bytes) e realiza seus cálculos utilizando o sistema binário (bits).

Como vimos na unidade II, os sistemas computacionais armazenam grandes volumes de dados em memórias secundárias, pois elas têm custo mais barato que uma memória principal. Os dados ficarão gravados na memória secundária e poderão ser acessados quando o usuário necessitar. Esses dados normalmente estão organizados em forma de arquivos e podem conter dados (códigos, textos, números), sons, imagens e vídeos. Todos esses tipos de dados quando gravados são transformados para dados digitais e no final acabam por se transformar em zeros e uns.

Então, utilizando algum tipo de codificação, podemos representar caracteres, símbolos e informações. Veja a Tabela 5 que apresenta alguns exemplos de caracteres ASCII em diversos sistemas de numeração.

Tabela 5 - Caracteres ASCII e os sistemas de numeração

Caracteres ASCII	Representações			
	Decimal	Hexadecimal	Octal	Binário
@	64	40	100	01000000
$	36	24	044	00100100
A	65	41	101	01000001
a	97	61	141	01100001

Fonte: elaborada pelo autor

Uma imagem, por exemplo, é decomposta em pixels (pontos na tela), de acordo com a sua resolução da imagem (largura e altura). De acordo com a qualidade da imagem, teremos uma padrão de cores diferentes. Vejamos um exemplo de uma imagem de resolução 320x230. Esta imagem terá cerca de 73600 pixels. Considerando que a imagem tem 16 bits de cores para cada pixel, obtendo então 1177600 bits no tamanho total da imagem, ou seja, cerca de 144 KB.

Para que possamos entender melhor como isso tudo ocorre, vamos ver alguns sistemas de numeração e suas bases que são utilizados em sistemas computacionais, conforme observamos no exemplo da tabela ASCII da Figura 3.

3.2 Sistemas de numeração e bases

Em um sistema de numeração, compreendemos por base numérica o conjunto de símbolos (algarismos) usados para representar uma certa quantidade ou número.

a) Sistema decimal - sistema de numeração que possui 10 algarismos (0, 1, 2, 3, 4, 5, 6, 7, 8 e 9), logo sua base (mão) é 10.

b) Sistema hexadecimal - sistema de numeração que possui 16 algarismos (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F), logo sua base (mão) é 16.

c) Sistema octal - sistema de numeração que possui 8 algarismos (0, 1, 2, 3, 4, 5, 6, 7), logo sua base (mão) é 8.

d) Sistema binário - sistema de numeração que possui 2 algarismos (0, 1), logo sua base (mão) é 2.

Figura 5 - Tabela dos sistemas de numeração de 0 a 15.

Hexadecimal	Octal	Binário	Decimal
0	0	0	0
1	1	1	1
2	2	10	2
3	3	11	3
4	4	100	4
5	5	101	5
6	6	110	6
7	7	111	7
8	10	1000	8
9	11	1001	9
A	12	1010	10
B	13	1011	11
C	14	1100	12
D	15	1101	13
E	16	1110	14
F	17	1111	15

Fonte:elaborada pelo autor

Saiba mais!

Vamos compreender melhor a representação binária e hexadecimal?

O vídeo “O Hit dos Bits” pretende apresentar o sistema de numeração binário e mostrar aplicações de sistemas de numeração diferentes do decimal.

Após gravar uma música, Janis pergunta a seu produtor, Celsão, se é possível levar em seu pen drive o arquivo com a gravação. A partir daí, Celsão explica como é armazenada a informação contida na música. Para isso, ele fala do sistema binário de numeração e ensina que os computadores atuais trabalham com este sistema para processar e armazenar dados. Acesse o link abaixo e aproveite para assistir essa explicação bastante interessante.

Acesse: <https://www.youtube.com/watch?v=cN_J5FqnHGw>

3.3 A notação posicional

A forma mais utilizada para a representação numérica é a notação posicional.

A notação posicional representa o valor de um número em função da posição e do valor de cada algarismo dentro do número. De acordo com Monteiro (2007), na notação posicional, os algarismos que compõem um número assumem valores diferentes, dependendo de sua posição relativa nele. O valor total do número é a soma dos valores relativos de cada algarismo. Logo, de acordo com o sistema de numeração utilizado, dizemos que a quantidade de algarismos que o compõem é denominada base.

Desta forma, empregando o conceito de notação posicional, podemos realizar a conversão entre diferentes bases. Vejamos como podemos compreender a notação posicional empregando o número decimal 7450.

Ex: 7450 na base 10

$7450_{(10)}$

$0 \times 10^0 = 0 \times 1 = 0$

$5 \times 10^1 = 5 \times 10 = 50$

$4 \times 10^2 = 4 \times 100 = 400$

$7 \times 10^3 = 7 \times 1000 = 7000$

Logo: $7450 = 0 + 50 + 400 + 7000$

Vamos refletir?

BASE é a quantidade de algarismos de um determinado sistema numérico. Portanto, BASE 10 no sistema decimal e BASE 2 no sistema binário.

O emprego da notação posicional é resultado da utilização dos numerais hindu-arábicos.

Observe, de acordo com o item 5 da Tabela 2, como os números romanos não empregam a notação posicional.

Naquele sistema de representação numérica quando desejamos efetuar uma operação de soma ou subtração, basta colocar um número acima do outro e efetuar a operação desejada entre os numerais, obedecendo a sua ordem.

Considerando um sistema de numeração posicional qualquer, podemos generalizar que um número N é expresso da seguinte forma:

$$N = d_{n-1} \times b^{n-1} + d_{n-2} \times b^{n-2} + \ldots + d1 \times b1 + d0 \times b0$$

No qual:

d: é cada um dos algarismos do número

n-1 (índice): é a posição de cada algarismo

b: é a base de numeração

n: é o número de dígitos inteiros

Vejamos mais alguns exemplos do processo de numeração posicional:

$\mathbf{9874_{10}} = 9 \times 10^3 + 8 \times 10^2 + 7 \times 10^1 + 4 \times 10^0 = 9x1000 + 8x100 + 7x10 + 4x1 = \mathbf{9874_{10}}$

$\mathbf{11011_2} = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 16 + 8 + 0 + 2 + 1 = \mathbf{27_{10}}$

$\mathbf{2345_8} = 2 \times 8^3 + 3 \times 8^2 + 4 \times 8^1 + 5 \times 8^0 = 512 + 192 + 32 + 5 = \mathbf{1253_{10}}$

$\mathbf{2CFE_{16}} = 2 \times 16^3 + 12(C) \times 16^2 + 15(F) \times 16^1 + 14(E) \times 16^0 = 2x4096 + 12x256 + 15x16 + 14x1 = \mathbf{11518_{10}}$

No caso do Hexadecimal, é necessário converter as letras aos valores decimais para o cálculo na fórmula da numeração posicional. No exemplo acima (2CFE16), mantivemos as letras para facilitar a compreensão. Observe que realizamos a conversão do valor decimal de cada letra utilizando a tabela de valores da Figura 5.

3.4 Conversão de bases

Para o processo de conversão de bases, podemos utilizar um dos métodos descrito a seguir. Em alguns casos, a depender do método de conversão, é importante utilizar a tabela de valores descrita na Figura 5 para ajudar no processo de conversão.

a) De binário (2) para octal (8) ou decimal (16)

Os números octais e hexadecimais equivalem a agrupamentos de 3 bits ou 4 bits, respectivamente, descritos na tabela da Figura 5. Assim, buscando esses valores naquela tabela temos uma rápida conversão entre os sistemas de numeração.

Exemplos:

$111111011111_2 = ????_8$ (Qual o número equivalente octal?)

Separamos de 3 em 3 bits e olhamos o equivalente octal na tabela da Figura 5.

111	111	011	111	=	7737_8
7	7	3	7		

$1111\ 1101\ 1111_2 = ????_{16}$ (Qual o número equivalente hexadecimal?) Separamos de 4 em 4 bits e olhamos o equivalente hexadecimal na tabela da Figura 5.

1111	1101	1111	=	FDF_{16}
F	D	F		

b) De octal (8) ou hexadecimal (16) para binário (2)

A conversão, neste caso, é semelhante a anterior, porém inversa. Agora, iremos identificar o valor binário de cada algarismo octal ou hexadecimal na tabela da Figura 5.

Exemplos:

$6737_8 = ????_2$ (Qual o número equivalente binário?)
Buscamos cada algarismo octal na tabela da Figura 5.

6	7	3	7	=	1101110111112
110	111	011	111		

$BFE_{16} = ????_2$ (Qual o número equivalente binário?)
Buscamos cada algarismo hexadecimal na tabela da Figura 5.

B	F	E	=	101111111110_{16}
1011	1111	1110		

c) De octal (8) ou hexadecimal (e vice-versa)

Um número octal e um hexadecimal possuem a mesma representação binária. O que muda é apenas a questão de agrupamento dos binários que no octal é de 3 em 3 e no hexa é de 4 em 4.

Exemplos:

$6737_8 = ????_{16}$ (Qual o número equivalente hexadecimal?)

Convertemos o octal para binário primeiro. Depois reagrupamos o binário de 4 em 4 bits e, em seguida, buscamos o valor hexadecimal de cada algarismo binário na tabela da Figura 5.

6	7	3	7	=	DDF_{16}
110	111	011	111		
Reagrupamos de 4 em 4	1101	1101	1111		
Valor hexadecimal	D	D	F		

$BFE_{16} = ????_8$ (Qual o número equivalente octal ?)

Buscamos cada algarismo hexadecimal na tabela da Figura 5.

Hexa	B	F	E		=	5776_8
Binário	1011	1111	1110			
Reagrupamos de 3 em 3	101	111	111	110		
Valor octal	5	7	7	6		

d) De uma base B (qualquer) para decimal (10) - Notação posicional

Utilizamos o procedimento da notação posicional já explicado no item 3 desta unidade.

Exemplos: $10011010_2 = ?_{10}$

b=2, n=8

$10011010_2 = 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 =$

$1x128 + 0x64 + 0x32 + 1x16 + 1x8 + 0x4 + 1x2 + 0x1 = 154_{10}$

$47_8 = ?_{10}$

b=8, n=2

$47_8 = 4 \times 8^1 + 7 \times 8^0 = 32 + 7 = 39_{10}$

$CFE_{16} = ?_{10}$

b=16, n=3

$CFE_{16} = 12(C) \times 16^2 + 15(F) \times 16^1 + 14(E) \times 16^0 =$

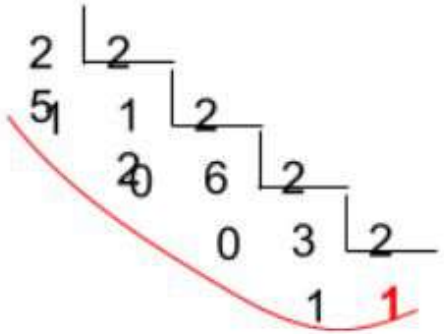
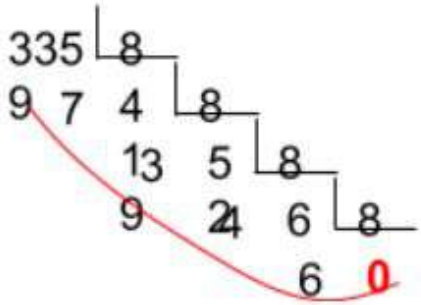
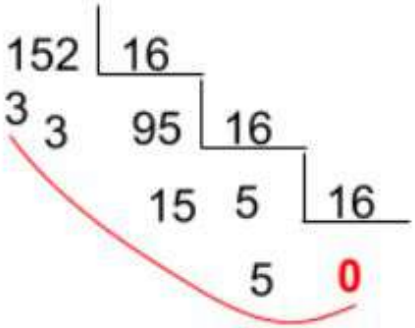
$12x256 + 15x16 + 14x1 = 3326_{10}$

e) De decimal (10) para uma base (B) qualquer - Divisões sucessivas

Basicamente o processo é feito utilizando-se o método de divisões sucessivas do número decimal definido pela base para a qual desejamos fazer a conversão. A divisão será feita até que o quociente seja menor que a base pretendida. Para a formação do número final, utilizamos os restos em ordem inversa, começando pelo último quociente.

Vejamos alguns exemplos a seguir:

Figura 6 - Divisões sucessivas

$25_{10} = ?_2$	$3359_{10} = ?_8$	$1523_{10} = ?_{16}$
		
11001_2	6437_8	$0\ 5\ 15\ 3 = 5F3_{16}$

Fonte elaborada pelo autor

Saiba mais!

Vamos aprender um pouco mais sobre números binários e sistemas de representação e conversão de bases?

Assista a uma vídeo-aula interessante do Professor Nivaldo Jr sobre base numérica. Acesse o link abaixo e aproveite para assistir a essa explicação bastante interessante.

Acesse: <https://www.youtube.com/watch?v=IUPECKg7ll0>

Assim terminamos nossa Unidade III. Os sistemas de numeração são importantes para quem atua em Tecnologia da Informação, especialmente para os que desejam avançar em Programação de Computadores. Lembrem-se de aprofundar as pesquisas e leituras. Neste capítulo aprendemos os conceitos que envolvem a origem dos sistemas de numeração, sua relação com a representação de dados. Avançamos para a compreensão das diferentes bases numéricas e da notação posicional e aprendemos a converter valores entre bases diferentes. Aproveite para consultar o glossário se tiver alguma dúvida de algum termo. Agora vamos realizar as atividades propostas e acompanhar as ações presenciais da disciplina.
Não esqueça de olhar os sites e vídeos indicados que irão ampliar os seus horizontes de conhecimentos sobre o tema da unidade.

O que temos para a próxima unidade professor? Para onde nossa trilha de aprendizagem irá?

Em nosso próximo passo vamos iniciar a Unidade V - Portas Lógicas. Nela vamos compreender melhor a Álgebra de Boole e identificar as portas lógicas, circuitos combinacionais e suas funções. Espero você na próxima lição!

Vamos rever?

A necessidade de contar acompanha a existência do ser humano há muito tempo. Precisamos diariamente contar objetos e realizar diferentes operações matemáticas.

Ao longo da história de nossas civilizações, diversos sistemas de numeração foram desenvolvidos e o homem utiliza até hoje inúmeras formas de contagem e representação de valores.

O termo digital deriva de dígito, que por sua vez tem origem no latim digitus que significa dedo. Assim, ao longo de sua evolução a humanidade desenvolveu o processo de contagem e os dedos sempre foram os instrumentos mais eficientes e simples para a contagem de pequenos valores.

O sistema de numeração decimal, o mais usado atualmente, é um sistema de base dez, pois são dez os dedos das duas mãos dos seres humanos. Os computadores têm como base para seu funcionamento a utilização de eletricidade.

Assim, em um computador trabalhamos com sinais elétricos que são representados por 0 e 1 e que, por sua vez, representam o SIM/NÃO, ABERTO/FECHADO, ACIMA/ABAIXO, LIGADO/DESLIGADO (lógica booleana).

Vamos rever?

Representar informações apenas com 0 e 1 é algo bastante limitado. Assim os bits podem ser agrupados para ter um significado útil.

O byte é o menor grupo ordenado de bits que representa uma informação útil e inteligível para o ser humano. A palavra de um computador pode ser compreendida como o tamanho da capacidade de armazenamento e transferência de informação (na memória e CPU), ou como sendo um conjunto de bits que representa uma informação útil naquele sistema computacional.

Por meio da combinação de bits, através de um código evidentemente, pode-se chegar a representações variadas. Desta forma, foram criados padrões para representar dados em computadores. Como exemplo dessas representações, temos o ASCII e o UNICODE.

Então, utilizando algum tipo de codificação, podemos representar caracteres, símbolos e informações. O sistema decimal possui base 10, o hexadecimal base 16, o octal base 8 e o binário base 2.

A notação posicional permite transformar qualquer valor de uma base N para a base decimal. Para transformar de decimal para qualquer base usamos o método de divisões sucessivas.

Questões de autoaprendizagem

1) Efetue as conversões de base:

a) $10101101_2 = (\ )_{16} = (\ )_8$

b) $FB7_{16} = (\ )_2 = (\ )_8$

c) $4325_8 = (\ )_2 = (\ )_{16}$

d) $1111_2 = (\ )_{16} = (\ )_8$

e) $37_8 = (\ )_2 = (\ )_{16}$

f) $A7B_{16} = (\ )_2 = (\ )_8$

g) $10101101_2 = (\ )_{16} = (\ )_8$

h) $2746_8 = (\ )_2 = (\ )_{16}$

i) $4746_{16} = (\ )_2 = (\ )_8$

j) $1100110111110111_2 = (\ )_{16} = (\ )_8$

k) $2746_{16} = (\ )_{10}$

l) $2746_{10} = (\ )_{16}$

m) $2746_8 = (\ )_{10}$

Questões de autoaprendizagem

1) Faça uma pesquisa na Internet como se fosse adquirir um computador e descreva os componentes do computador que selecionou. Busque informações técnicas sobre ele e seus componentes (especificações técnicas). Identifique os principais componentes de hardware do computador selecionado, suas funcionalidades. Descreva a funcionalidade dos principais dispositivos de hardware e suas características técnicas

2) Pesquise como é feita a representação dos diversos tipos de informação em arquivos binários e como isso está relacionado com os conceitos aprendidos nesta unidade. Utilize a bibliografia de nossa disciplina para aprofundar a pesquisa.

3) Como está o mercado de trabalho na área de programação no Distrito Federal? Faça uma pesquisa sobre 3 empresas de desenvolvimento de software. Identifique os contatos dessa empresa e o local onde ela divulga vagas de emprego. Busque três anúncios de cada empresa sobre a vaga de PROGRAMADOR. Entre os nove anúncios selecionados qual é a linguagem mais solicitada? Quais as características de perfil mais solicitadas?

Saiba mais!

Site do Guia do Hardware - Sistemas de Numeração. Disponível em: <http://www.hardware.com.br/artigos/sistemas-numeracao-informatica/> Acesso em: 04 de nov.2017.

Calculadora ONLINE - Conversão de bases. Disponível em: <http://www.calculadoraonline.com.br/conversao-bases>. Acesso em: 04 de nov.2017.

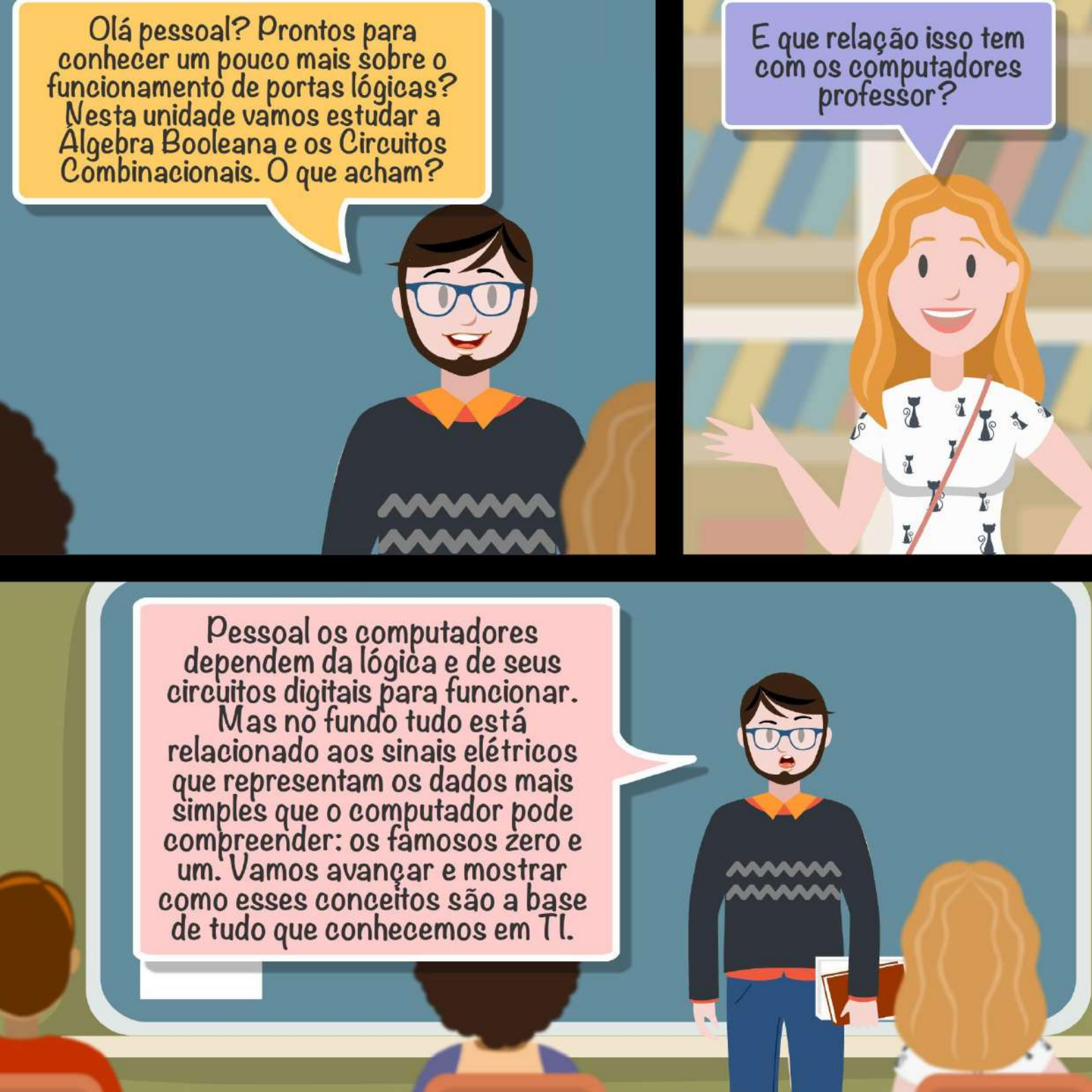
Dicas de conversão. Disponível em: <https://dicasdeprogramacao.com.br/as-10-conversoes-numericas-mais-utilizadas-na-computacao/>. Acesso em: 04 de nov.2017.

Unidade 4 Portas Lógicas

Ao término desta unidade esperamos atingir o seguinte objetivo: reconhecer e saber as funções das portas lógicas.

Nesta unidade, denominada Portas Lógicas, buscamos apresentar a história do surgimento do transistor e dos computadores digitais. Vamos estudar as portas lógicas e como elas se integram para formar circuitos integrados.

Vamos começar!


Olá pessoal? Prontos para conhecer um pouco mais sobre o funcionamento de portas lógicas? Nesta unidade vamos estudar a Álgebra Booleana e os Circuitos Combinacionais. O que acham?
E que relação isso tem com os computadores professor?
Pessoal os computadores dependem da lógica e de seus circuitos digitais para funcionar. Mas no fundo tudo está relacionado aos sinais elétricos que representam os dados mais simples que o computador pode compreender: os famosos zero e um. Vamos avançar e mostrar como esses conceitos são a base de tudo que conhecemos em TI.

4.1 A Álgebra Booleana

A lógica é uma palavra que vem do grego “logos” e pode ser compreendida como razão, discurso ou linguagem. Derivamos o verbo “leigenin” a partir da palavra “logos” com o significado de colher, reunir, juntar, calcular ou ordenar. A lógica está inserida neste sentido, denotando uma relação entre a linguagem e o conhecimento, considerando que o conhecimento é expresso por meio do rigor e da precisão do discurso linguístico.

O filósofo Thales de Mileto (623 a 548 a.C.) está relacionado entre os sete maiores filósofos da antiguidade. Devido a seu esforço na busca de um princípio único, a essência do universo, é descrito como o primeiro filósofo. Seu esforço rendeu grandes descobertas matemáticas, dentre as quais a primeira previsão do eclipse solar. Dentre os seus trabalhos destacam-se o uso e a aplicação de demonstrações e provas na ciência e matemática em geral.

Entre os sete grandes filósofos gregos, encontra-se também o discípulo de Platão, Aristóteles (384 a 322 a.C), o qual também foi professor de Alexandre “o Grande”. A Lógica que conhecemos tem múltiplas influências das contribuições de Aristóteles, dentre as quais destacamos:

a) a identificação dos conceitos básicos da lógica;

b) a introdução de letras para denotar os termos;

c) a criação de termos fundamentais para analisar a lógica do discurso: “Válido”, “Não Válido”, “Contraditório”, “Universal”, “Particular”.

Aristóteles foi considerado o pai da lógica ainda que naquela época o termo lógica não tivesse sido cunhado até então. No século II a.C., quando os filósofos estoicos passaram a utilizar a palavra lógica como centro do seu pensamento é que o termo passou a ser conhecido.

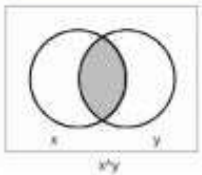
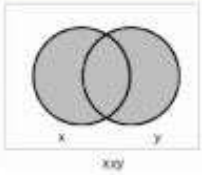
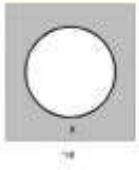
Os silogismos eram a base da Lógica Aristotélica.	Todo homem é mortal. → Sócrates é homem. → Logo, Sócrates é mortal.

Já na idade média, Gottfried Wilhelm Leibniz (1646 a 1716) teve papel importante quando observou que "a única maneira de garantir a consistência de nossos raciocínios e torná-los tão tangíveis quanto os dos matemáticos…" e "se duas pessoas discordarem, basta calcular quem está certo". Ele foi o primeiro a propor o sistema binário. Também contribuiu para a formatação dos princípios da nossa Lógica atual.

Na idade moderna, George Boole (1815 a 1864) foi considerado um dos fundadores da Ciência da Computação, apesar dos computadores não existirem em sua época. A Álgebra Booleana foi desenvolvida por ele. Sua formulação utilizava símbolos algébricos (conectivos) como 'x', 'y', 'z', 'p', 'q', 'r' para denotar palavras, frases, ou proposições (representando as expressões por letras).

Seu sistema algébrico era composto por operações como adição e multiplicação e métodos de resolução de equações, o que exigia a formulação de uma linguagem simbólica de pensamento. A Álgebra de Boole permite que uma afirmação (lógica) possa ser expressa matematicamente. O objetivo da lógica é modelar o raciocínio humano.

Figura 1 - Operações básicas da Álgebra Booleana

Entradas		conjunção x ^ y = xy	disjunção x v y = x + y – xy	Entrada	negação ¬x = 1 - x
					
X	Y	x ^ y	x v y	X	¬x
0	0	0	0	0	1
1	0	0	1		
0	1	0	1	1	0
1	1	1	1		

Fonte: Elaborada pelo autor.

A resolução de uma equação não levaria a uma resposta numérica, mas sim a formulação de uma conclusão lógica. Nascia, então, a "álgebra do pensamento" conhecida como lógica binária.

4.2 A história das portas lógicas

A história das portas lógicas começa em meados da década de 1970 nos Estados Unidos quando a primeira central telefônica entrou em operação. O emprego das centrais telefônicas popularizou o uso do telefone. A operação dessas centrais era realizada por telefonistas.

Sua função era realizar as conexões necessárias entre as ligações. O sistema evoluiu com a adição de um código que permitia que qualquer telefonista intermediasse a ligação. Com a popularização, a necessidade de telefonistas cresceu. Essa era terminou com o surgimento do telefone de discagem direta e a ligação automática. E como conseguimos chegar a essa fase automatizada? Tudo está relacionado, turma! Vejam a imagem, a seguir, que demonstra essa cena interessante.

Figura 2 - Centrais telefônicas - EUA

Fonte:Imagem retirada da internet. Disponível em: <http://soulsoretro.blogspot.com/2018/05/a-era-de-ouro-das-telefonistas.html>. Acesso em: 05 de nov. 2017.

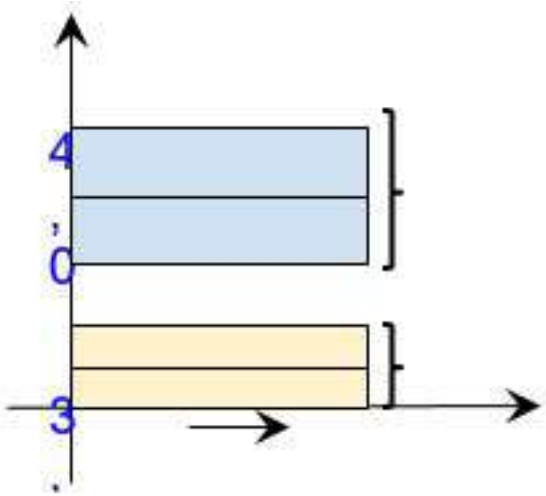
Em 1937, pulamos da Álgebra de Boole aos projetos de circuitos digitais quando um matemático e engenheiro eletrônico americano chamado Claude Shannon, de apenas 21 anos de idade, percebeu que havia uma semelhança entre álgebra booleana e circuitos de chaveamento de telefone. O tema de sua tese de mestrado no Massachusetts Institute of Technology (MIT) era a aplicação da Álgebra Booleana a sistemas elétricos. Sua tese de mestrado fundou a ciência dos projetos de circuitos digitais e foi considerada a mais importante do século XX. Sua ideia foi colocada em prática imediatamente nos projetos de circuitos de chaveamento para telefones encerrando a era das telefonistas. Em seu trabalho, a exploração dessa característica de interruptores elétricos criou a lógica e os conceitos mais básicos dos computadores digitais.

4.3 O transistor e o computador digital

O computador digital é construído a partir de um conjunto convenientemente distribuído de circuitos lógicos (portas). Tais circuitos têm funções diferenciadas: armazenar valores, permitir e controlar o fluxo de sinais entre componentes e realizar operações matemáticas.

A informação binária (0 ou 1) é representada em um sistema digital por quantidades físicas (sinais elétricos). Essas operações são combinações simples de operações aritméticas e lógicas bem básicas, tais como: somar bits, complementar bits(para fazer subtrações), comparar bits, mover bits. As operações são físicas e realizadas por meio de circuitos eletrônicos denominados de circuitos digitais, compostos basicamente por portas lógicas que permitem ou não a passagem de sinais elétricos.

Figura 3 -
Representação física dos sinais elétricos



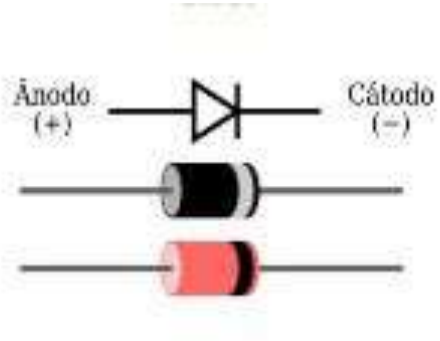
Fonte: Elaborada pelo autor.

Assim, a lógica baseia-se no fato de um transistor realizar a função de uma chave binária (porta lógica) cujo tempo de comutação (chaveamento) é pequeno (nanossegundos). As portas lógicas combinam sinais elétricos de baixa intensidade (cerca de 1mA - um miliampere). Os transistores e diodos são circuitos semicondutores.

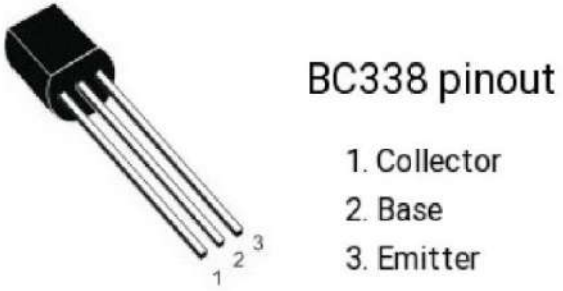
Semicondutores são materiais que se encontram em uma posição intermediária, não sendo bons condutores e nem bons isolantes, mas, ao serem tratados por processos químicos, permitem controlar a passagem de uma corrente elétrica.

Os transistores são formados por uma Base, um Coletor e um Emissor (veja a Figura 4).

Figura 4 - Diodo e Transistor



Fonte: Imagem retirada da internet. Disponível em: <https://pt.wikipedia.org/wiki/Diodo_semicondutor>. Acesso em: 05 de nov.2017.



Fonte: Imagem retirada da internet. Disponível em: <https://netcomputadores.com.br/p/bc33840-transistor-npn-25v-800ma/36768>. Acesso em: 05 de nov.2017.

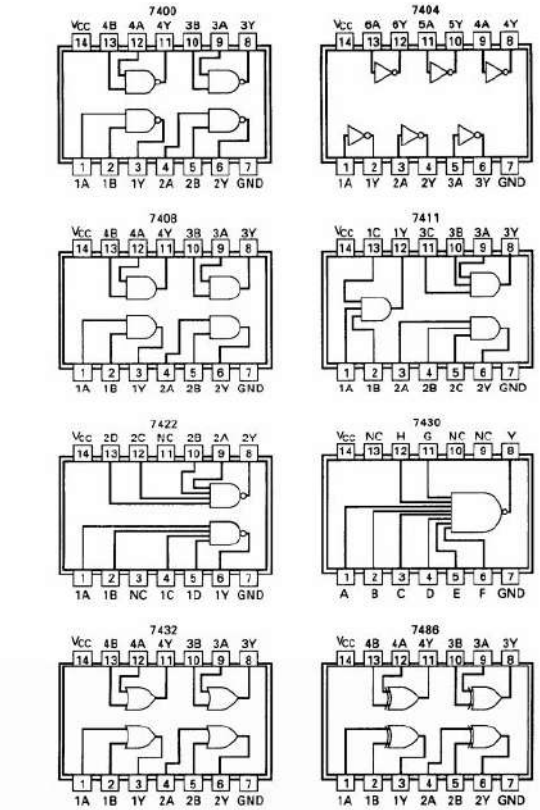
As portas lógicas são fornecidas em dispositivos denominados circuitos integrados ou CI´s. Um circuito integrado (chip) é um cristal semicondutor normalmente fabricado em silício. Cada circuito integrado comporta um certo número de portas lógicas.

o número de portas é limitado pelas características físicas do componente, como o número de terminais. Os circuitos integrados comerciais são classificados em famílias de acordo com o grau de integração (SSI, MSI, LSI, VLSI e ULSI).

Figura 5 - Circuitos integrados e portas lógicas

Nível de integração	Número de Portas	Aplicação
SSI (*Small-Scale Integration*) - Integração em pequena escala	Menos de 12	portas básicas simples.
MSI (*Medium-Scale Integration*) - Integração em média escala	Menos de 100	funções elementares, somadores, etc
LSI (*Large-Scale Integration*) - Integração em larga escala	até alguns milhares	pequenos processadores, etc.
VLSI (*Very Large-Scale Integration*)- Integração em escala muito larga.	a partir de alguns milhares	microprocessadores, etc.

TTL	CMOS	Especificações
7400	4011	4 portas NAND de 2 entradas
7402	4001	4 portas NOR de 2 entradas
7404	4009	6 portas INVERSORAS
7408	4081	4 portas AND de 2 entradas
7432	4071	4 portas OR de 2 entradas
7486	4030	4 portas XOR de 2 entradas
7410	4023	3 portas NAND de 3 entradas
7427	4002	2 portas NOR de 4 entradas
7430	74C30	1 porta NAND de 8 entradas



Algumas pastilhas SSI. *Layouts* de pinos de *The TTL Data Book for Design Engineers*, direitos reservados de Texas Instruments Incorporated, 1976.

Fonte: Imagem retirada da internet
Disponível em: <http://www.dsc.ufcg.edu.br/~joseana/IC_NA09.pdf>.
Acesso em: 05 de nov.2017

4.4 Funções e portas lógicas

As Portas lógicas são dispositivos dos circuitos digitais que implementam funções lógicas. Monteiro (2007) indica que um computador é constituído de elementos eletrônicos, como resistores, capacitores e, principalmente, transistores. Os computadores atuais são montados utilizando-se uma combinação de unidades construtivas básicas chamadas de portas lógicas. Essas unidades são construídas por combinação de transistores e dispositivos semicondutores auxiliares. Imagine um grande sistema digital "lego" de peças que se encaixam (portas lógicas - unidade base) formando uma unidade maior chamada microprocessador. A integração dessas peças básicas formam os Circuitos Integrados (chips ou CIs) com objetivo e função de realizar uma tarefa específica. A complexidade desses CIs depende do seu nível de integração conforme descrito na Figura 5.

A Álgebra de Boole indica que uma operação lógica qualquer (ex.: soma ou multiplicação de dígitos binários) resulta sempre em dois valores: 0 (falso) ou 1 (verdadeiro). Desta forma, dado um conjunto definido de valores de entrada, é possível determinar previamente todos os resultados possíveis de uma operação lógica. Esse conjunto de valores organizados previamente é definido como Tabela Verdade e, segundo Monteiro (2007), cada operação lógica possui sua própria tabela verdade.

A seguir, vamos conhecer as funções das portas lógicas:

AND	**NAND**
OR	**NOR**
NOT	**XOR**

a) A porta AND

A porta AND é um circuito que executa a função AND (E), ou seja, executa a multiplicação binária de duas ou mais variáveis de entrada. Os operadores são binários simples, 0 e 1. A saída S só será verdadeira (um) quando as entradas "A", "E", "B" também forem verdadeiras (um). De acordo com Monteiro (2007), as portas E (AND) podem ser aplicadas para compor circuitos para transferência de bits de dados da memória para a CPU, por exemplo, com o objetivo de garantir que um bit de origem seja o mesmo de destino. É representada pelo operador ".".

Figura 6 - Porta AND (E) com duas e três entradas

Símbolo

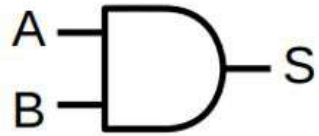


Tabela Verdade

Entrada		Saída
A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

S=A.B

Símbolo

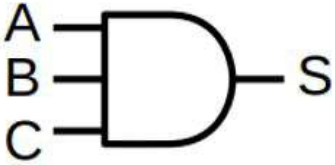


Tabela Verdade

Entrada			Saída
A	B	C	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

S=A.B.C

Fonte: Adaptada de Monteiro (2007)

b) A porta OR

A porta OR é um circuito que executa a função OR (OU), ou seja, simula a soma binária de duas ou mais variáveis de entrada. Os operadores são binários simples, 0 e 1. A saída S será verdadeira (um) sempre que pelo menos uma das entradas, "A" ou "B", for verdadeira (um). A saída S somente será falsa (zero) quando as duas entradas, "A" e "B", forem falsas (zero). Segundo Monteiro (2007), as operações lógicas 'OU' são muito empregadas em lógica digital, particularmente nos comandos de decisão de algumas linguagens de programação. É representada pelo operador "+".

Figura 7 - Porta OR (OU) com duas e três entradas

Símbolo

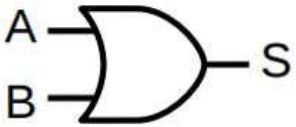


Tabela Verdade

Entrada		Saída
A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

S=A+B

Símbolo

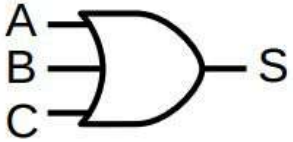


Tabela Verdade

Entrada			Saída
A	B	C	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

S=A+B+C

Fonte: Adaptada de Monteiro (2007)

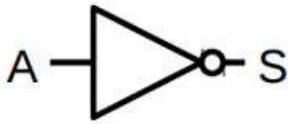
c) A porta NOT

A porta NOT é um circuito que executa a função NOT (NÃO), ou seja, é aquela que funciona como um inversor, invertendo ou complementando o estado da variável. Se a entrada A estiver falsa (zero), a saída será verdadeira (um) e vice-versa.

Os operadores são binários simples, 0 e 1. Representada pelo operador "¯" ou '. As portas inversoras possuem apenas uma entrada. Segundo Monteiro (2007), as portas inversoras têm como principal aplicação a inversão de um grupo de bits representativos de um número negativo (em complemento de 1 e complemento de 2) quando da representação de números negativos no formato binário.

Figura 8 - Porta NOT (NÃO)

Símbolo



Um círculo ( ) em um pino de algum circuito lógico indica que a entrada/saída é negada.

Tabela Verdade

Entrada	Saída
A	S
0	1
1	0

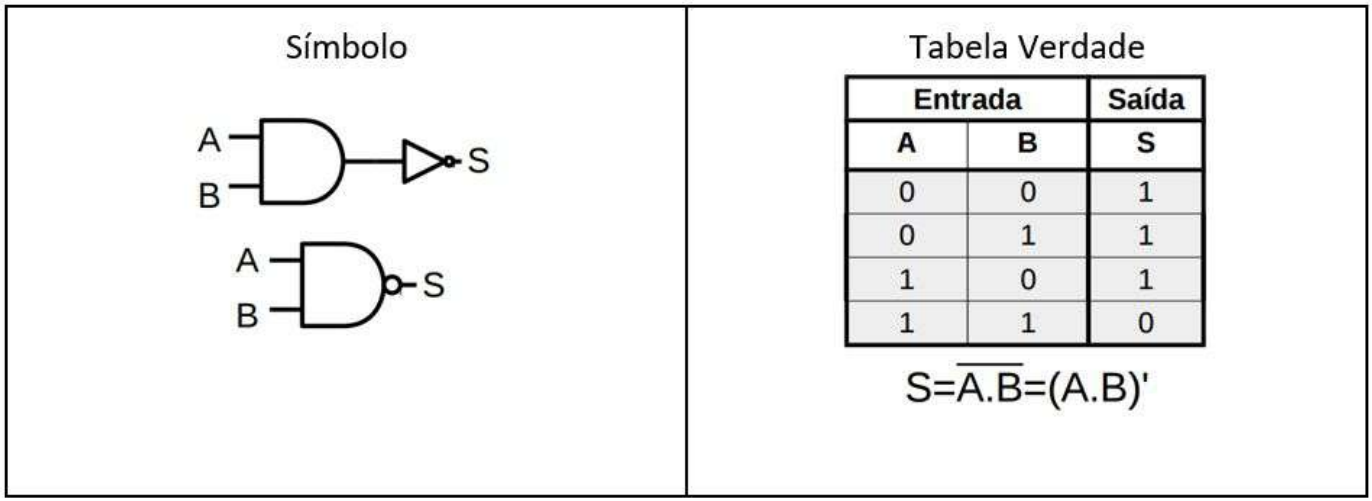
$S=\overline{A}=A'$

Fonte: Adaptada de Monteiro (2007)

d) A porta NAND

A porta NAND é um circuito derivado da porta lógica AND. Ela executa a função AND (E) e depois tem sua saída invertida, ou seja, equivale à Porta E com saída negada. As operações lógicas são realizadas em dois passos: primeiro a operação AND, em seguida, o seu resultado é invertido. Essa porta lógica possui diversas aplicações, sendo utilizada para reduzir a complexidade e a quantidade de portas lógicas necessárias a um determinado circuito lógico.

Figura 9 - Porta NÃO E (NE ou NAND)



Fonte: Adaptada de Monteiro (2007)

e) A porta NOR

A porta NOR é um circuito derivado da porta lógica OR. Ela executa a função OR (OU) e depois tem sua saída invertida, ou seja, equivale à Porta OR com saída negada. As operações lógicas são realizadas em dois passos: primeiro a operação OR, em seguida, o seu resultado é invertido. Essa porta lógica possui diversas aplicações, sendo utilizada para reduzir a complexidade e a quantidade de portas lógicas necessárias a um determinado circuito lógico.

Figura 10 - Porta NÃO OU (NOU ou NOR)

Símbolo

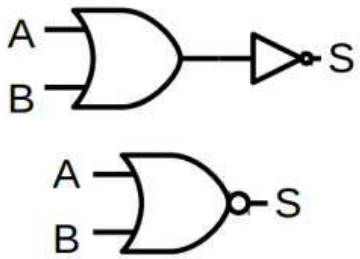


Tabela Verdade

Entrada		Saída
A	B	S
0	0	1
0	1	0
1	0	0
1	1	0

$S=\overline{A+B}=(A+B)'$

Fonte: Adaptada de Monteiro (2007)

f) A porta XOR

A porta XOR é um circuito que executa função de fornecer a saída verdadeira (um) quando as variáveis de entrada forem diferentes entre si, ou seja, a saída só será verdadeira (um) quando apenas uma das entradas também for verdadeira (um). A sua implementação necessita de diversas portas lógicas. A porta XOR só pode ter duas variáveis de entrada obrigatoriamente. Os operandos são binários simples (0 e 1).

Segundo Monteiro (2007), a porta XOR tem como principal função a verificação de igualdade, tendo inúmeras aplicações como, por exemplo, testar se duas palavras de dados são iguais. É uma porta bastante versátil. É representada pelo operador “ ”.

Figura 11 - Porta XOR (OU exclusivo)

Símbolo

Tabela Verdade

Entrada		Saída
A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

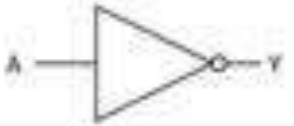
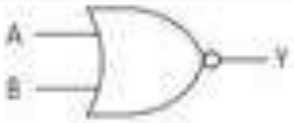
$S=A \oplus B$

Fonte: Adaptada de Monteiro (2007)

4.5 Resumo das Funções Lógicas

Considerando que um circuito lógico pode ser composto por um conjunto de diversas portas lógicas suas tabelas verdade poderão também ser compostas por inúmeras entradas e saídas, sendo estas representadas pelas suas respectivas equações booleanas.

Figura 12 - Símbolos gráficos e equações booleanas de portas lógicas

Função Lógica Básica	Símbolo Gráfico da Porta	Equação Booleana
AND		$Y = A.B$
OR		$Y = A+B$
XOR		$Y = A \oplus B$
NOT		$Y = \bar{A}$
NAND		$Y = \overline{A.B}$
NOR		$Y = \overline{A+B}$

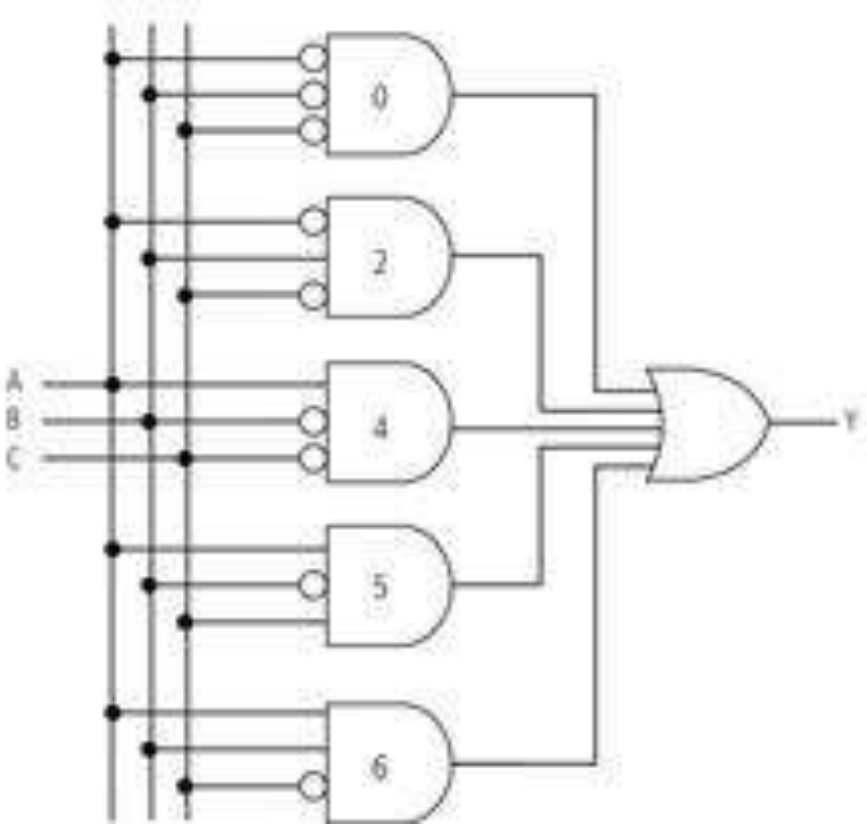
Fonte: Adaptada de Monteiro (2007)

Os circuitos combinacionais (ou combinatórios) construídos a partir do arranjo de diversas portas lógicas básicas, interligadas em diversas unidades configurando uma rede lógica, permitem construir circuitos importantes empregados na construção de computadores e vários outros sistemas digitais. Dentre eles podemos citar: somadores, subtratores, circuitos que executam prioridades, codificadores e decodificadores.

Segundo Monteiro (2007), um circuito combinacional é definido como um conjunto de portas lógicas cuja saída, em qualquer instante de tempo, é função somente das entradas, ou seja, é aquele em que a saída depende única e exclusivamente das combinações entre as variáveis de entrada.

Monteiro (2007) afirma ainda que existe outra categoria de circuitos que combina portas lógicas, denominada circuitos sequenciais, os quais, além de possuir portas, contêm elementos de armazenamento (uma espécie de memória).

Figura 13 - Circuito combinacional implementado para uma determinada função.



Fonte: Adaptada de Monteiro (2007)

Vamos refletir?

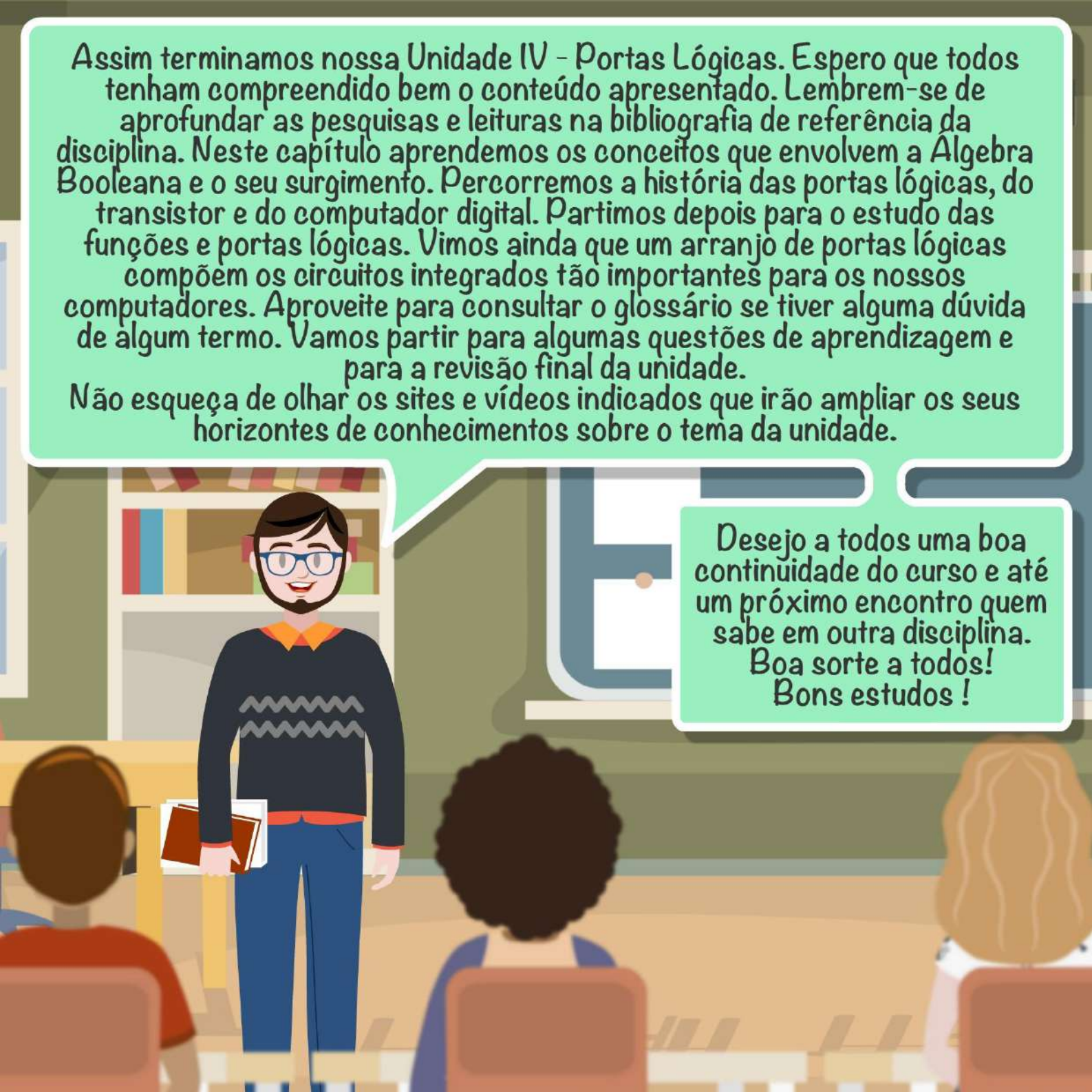
Em nossos computadores e sistema digitais temos circuitos combinacionais que desempenham funções essenciais, particularmente na realização de operações aritméticas básicas na Unidade Lógica Aritmética (ULA) destes sistemas digitais, em que uma série de portas lógicas são combinadas para adicionar, subtrair, multiplicar ou dividir números binários. No caso das operações de multiplicação e divisão, além das portas lógicas, há a necessidade de circuitos sequenciais.

Saiba mais!

O que acha de visitar o Clube do Hardware e conhecer um pouco mais sobre a temática desta unidade? Gabriel Torres apresenta, neste artigo, tudo o que você precisa saber sobre portas lógicas, os componentes básicos da eletrônica digital.

Acesse:
http://www.clubedohardware.com.br/artigos/outros/introdu%C3%A7%C3%A3o-%C3%A0s-portas-l%C3%B3gicas-r34573/

Assim terminamos nossa Unidade IV - Portas Lógicas. Espero que todos tenham compreendido bem o conteúdo apresentado. Lembrem-se de aprofundar as pesquisas e leituras na bibliografia de referência da disciplina. Neste capítulo aprendemos os conceitos que envolvem a Álgebra Booleana e o seu surgimento. Percorremos a história das portas lógicas, do transistor e do computador digital. Partimos depois para o estudo das funções e portas lógicas. Vimos ainda que um arranjo de portas lógicas compõem os circuitos integrados tão importantes para os nossos computadores. Aproveite para consultar o glossário se tiver alguma dúvida de algum termo. Vamos partir para algumas questões de aprendizagem e para a revisão final da unidade.
Não esqueça de olhar os sites e vídeos indicados que irão ampliar os seus horizontes de conhecimentos sobre o tema da unidade.



Desejo a todos uma boa continuidade do curso e até um próximo encontro quem sabe em outra disciplina. Boa sorte a todos! Bons estudos !

Vamos rever?

A lógica é uma palavra que vem do grego “logos” e pode ser compreendida como razão, discurso ou linguagem. Aristóteles foi considerado o pai da lógica.

Os silogismos eram a base da Lógica Aristotélica. A Álgebra Booleana foi desenvolvida George Boole e utilizava símbolos algébricos (conectivos). A Álgebra de Boole permite que uma afirmação (lógica) possa ser expressa matematicamente. O objetivo da lógica é modelar o raciocínio humano. A resolução de uma equação não levaria a uma resposta numérica, mas sim a formulação de uma conclusão lógica.

Nascia, então, a “álgebra do pensamento” conhecida como lógica binária.

Claude Shannon percebeu que havia uma semelhança entre álgebra booleana e circuitos de chaveamento de telefone e propôs a aplicação da Álgebra Booleana a sistemas elétricos.

Sua ideia foi colocada em prática imediatamente nos projetos de circuitos de chaveamento para telefones encerrando a era das telefonistas.

Em seu trabalho, a exploração dessa característica de interruptores elétricos criou a lógica e os conceitos mais básicos dos computadores digitais.

Vamos rever?

O computador digital é construído a partir de um conjunto convenientemente distribuído de circuitos lógicos (portas).

Tais circuitos têm funções diferenciadas: armazenar valores, permitir e controlar o fluxo de sinais entre componentes e realizar operações matemáticas.

A informação binária (0 ou 1) é representada em um sistema digital por quantidades físicas (sinais elétricos). As operações são físicas e realizadas por meio de circuitos eletrônicos denominados de circuitos digitais, compostos basicamente por portas lógicas que permitem ou não a passagem de sinais elétricos.

Assim, a lógica baseia-se no fato de um transistor realizar a função de uma chave binária (porta lógica) cujo tempo de comutação (chaveamento) é pequeno (nanossegundos). Imagine um grande sistema digital "lego" de peças que se encaixam (portas lógicas - unidade base de diversos tipos: AND, OR, NOT, NAND, NOR, XOR, etc.) formando uma unidade maior chamada microprocessador.

A integração dessas peças básicas formam os Circuitos Integrados (chips ou CIs) com objetivo e função de realizar uma tarefa específica.

Saiba mais!

Sites indicados:

1. Aplicativo para Android para Simular Lógica no Celular (Logic Simulator Pro) Disponívelem:<https://apkpure.com/br/logic-simulator-pro/com.KAJORY.Logicsimulatorpro>. Acesso em: 03 de nov.2017.

2. Aplicativo para Android para Simular Lógica no Celular (Smart Logic Simulator) Disponível em: <https://play.google.com/store/apps/details?id=com.tomaszczart.smartlogicsimulator&hl=pt_BR>. Acesso em: 03 de nov.2017.

3. Visitamos o museu da Intel! Veja como é feito um chip e conheça a história da empresa Disponível em: <https://www.youtube.com/watch?v=AuUOrrW8YOU>. Acesso em: 03 de nov.2017.

Saiba mais!

Audiovisuais indicados:

1. Expressões Booleanas, Circuitos Lógicos e Tabela Verdade
   Disponível em: <https://www.youtube.com/watch?v=aYVz0l3ZMWc>. Acesso em: 05 de nov. de 2017.

2. Lógica Boleana. Disponível em: <https://www.youtube.com/watch?v=sVSYTOO6gDA>. Acesso em: 05 de nov. de 2017.

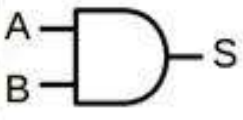
3. A História dos Microchips e Transistores. Disponível em: <https://www.youtube.com/watch?v=SelwNgqbtQ4>. Acesso em: 05 de nov. de 2017.

4. Como é feito um Microship? Disponível em: <https://www.youtube.com/watch?v=tGGM2x5BHTs>. Acesso em: 05 de nov. de 2017.

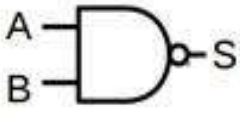
Questões de autoaprendizagem

a) Preencha o diagrama de portas abaixo completando as tabelas verdades e as identificações de cada porta e sua expressão.

A	B	S
0	0	
0	1	
1	0	
1	1	



Nome:
Expressão:

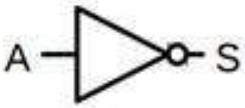


Nome:
Expressão:

A	B	S
0	0	
0	1	
1	0	
1	1	

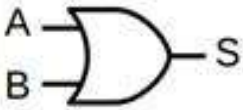
A	S
0	
1	

Nome:
Expressão:



A	B	S
0	0	
0	1	
1	0	
1	1	

Nome:
Expressão:

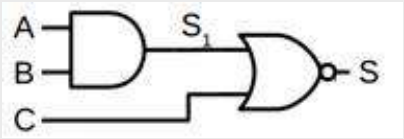


Nome:
Expressão:

A	B	S
0	0	
0	1	
1	0	
1	1	

Questões de autoaprendizagem

b) Qual expressão representa a saída S?



c) Qual seria a tabela verdade do circuito anterior, considerando as entradas abaixo?

Entrada			Saída	
A	B	C	S1	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Glossário

- **Acesso aleatório** - O acesso aleatório está diretamente relacionado à posição da memória endereçável. Dessa forma, dizemos que qualquer posição da memória pode ser selecionada, desde que endereçada corretamente.
- **Assembly** - é uma notação legível por humanos para o código de máquina que uma arquitetura de computador específica usa, utilizada para programar códigos entendidos por dispositivos computacionais, como microprocessadores e microcontroladores. O código de máquina, que é um mero padrão de bits, torna-se legível pela substituição dos valores em bruto por símbolos chamados mnemônicos.
- **Chip** - É uma pastilha de silício onde ficam os circuitos integrados compostos por milhões de componentes eletrônicos e essenciais aos computadores.
- **Dado** - é tudo aquilo que sozinho, pode não significar nada. Por exemplo: 5 m, 10 kg, 20 km/h. Mas se dissermos “uma passarela tem 5m de altura”, isso pode se constituir numa informação para um motorista de caminhão, que evitaria trafegar naquela via sabendo que seu caminhão tem uma altura maior. Já a informação subentende dados organizados segundo uma orientação específica para o atendimento ou emprego de uma pessoa ou grupo que os recebe.
- **Framework** - Abstração que une códigos comuns entre vários projetos de software, provendo uma funcionalidade genérica.

Glossário

- **Hardware** - Parte física do computador, componentes como placas de circuitos, discos de armazenamento, monitores e impressoras.
- **Informática** - A informática é a disciplina que lida com o tratamento racional e sistemático da informação por meios automáticos e eletrônicos. Representa o tratamento automático da informação. Constitui o emprego da ciência da informação através do computador. Embora não se deva confundir informática com computadores, na verdade ela existe porque estes existem. Pode ser considerada como "informação automática", ou seja, utilização de métodos e técnicas no tratamento automático da informação. Para tal, é preciso uma ferramenta adequada: o computador.
- **Interface** - Parte do software que interage com o usuário para receber comandos e transmitir informações.
- **Kernel** - Núcleo de um sistema operacional responsável por gerenciar os recursos do sistema computacional como um todo.
- **Loop** - É um termo muito utilizado em informática, principalmente, em redes e programação. Designa uma execução de uma tarefa que teoricamente pode nunca ter fim. Pode ser um aliado em processamentos que exigem volumes de dados muito grandes, como cálculos de previsão do tempo.

Glossário

- **Motherboard** - Placa-mãe: placa principal que liga todos os componentes internos do computador.
- **Neologismo** - emprego de palavras novas, derivadas ou formadas de outras já existentes, na mesma língua ou não.
- **Notebook** - Computador portátil que traz como principal característica a integração e miniaturização da maior parte dos componentes, tornando-o leve e de pequenas dimensões.
- **Pixel** - É a unidade de medida lógica utilizada para determinar a resolução de um monitor. Entretanto, a medida do pixel varia de acordo com a resolução física utilizada. Se for usada a resolução máxima do monitor, um pixel corresponde a exatamente um ponto no monitor. Caso a resolução adotada seja menor que aquela permitida pelo monitor, o pixel terá um tamanho maior de pontos, ou seja, um pixel será representado por mais de um ponto no monitor.
- **Processamento paralelo** - É aquele processamento em que uma tarefa demasiadamente grande, que consome muito tempo de processamento, pode ser dividida em várias pequenas tarefas e ser processada por dois ou mais processadores. Hoje em dia, com os computadores ligados em rede, é possível que esses processadores estejam em computadores separados, já que cada computador tem seu próprio processador.

Glossário

- **Program Counter (PC)** - Contador de Programas.
- **Semicondutores** - São sólidos cujo material possui propriedades de condução ou isolamento da corrente, dependendo do tratamento químico que é dado a ele.
- **Software** - Programa, conjunto de instruções lógicas que gerenciam as funções do computador e permitem a realização de tarefas específicas.
- **UC -** Unidade de Controle.
- **ULA -** Unidade Lógica e Aritmética.
- **Wikipédia** - projeto de enciclopédia livre, baseado na web, colaborativo e apoiado pela organização sem fins lucrativos Wikimedia Foundation. Seus 19 milhões de artigos (691 696 em português) foram escritos de forma colaborativa por voluntários ao redor do mundo e quase todos os seus verbetes podem ser editados por qualquer pessoa com acesso ao site .

Referências

AMARAL, Alan F. F. **Arquitetura de Computadores**: Curso Técnico em Informática. Colatina CEAD/IFES 2010

DELGADO, José.; RIBEIRO, Carlos. **Arquitetura de Computadores**. 2 ed. LTC, 2009.

HENESSY, John L.; PATTERSON, David A. **Arquitetura de Computadores:** uma abordagem quantitativa. 5 ed. Rio de Janeiro: CAMPUS, 2014.

MACHADO, Francis B. **Arquitetura de sistemas operacionais**. 4 Ed. Rio de Janeiro. LTC. 2007.

MORIMOTO, Carlos E. **Hardware, o Guia Definitivo**. Porto Alegre: GDH Press e Sul Editores, 2007.

MONTEIRO, Mário, A. **Introdução à organização de computadores**. 5 ed. Rio de Janeiro: LTC, 2007.

PATTERSON, David A. ; HENNESSY, John.L. **Organização e projeto de computadores** – a interface hardware software. 4 ed. Editora Campus, 2009.

STALLINGS, William. **Arquitetura e Organização de Computadores**. Rio de Janeiro: Makron Books Brasil, 2010.

Referências

STALLINGS, William. **Arquitetura e organização de computadores**: projeto para o desempenho. 8. ed. Pearson Prentice Hall, 2009.

TANENBAUM, Andrew. **Organização estruturada de computadores**. 5. ed. São Paulo: Person Education do Brasil: Prentice Hall, 2007.

VELLOSO, Fernando de Castro. **Informática**: conceitos básicos. 8 ed. Rio de Janeiro: Editora Elsevier, 2003.

WEBER, Raul Fernando. **Fundamentos de arquitetura de computadores**. Vol. 8. ed. Porto Alegre: Bookman: Instituto de Informática da UFRGS, 2008.

Lógica de Programação

Ronald Emerson Scherolt da Costa

Caro leitor,

Esta disciplina integra o currículo do Curso Técnico em Informática do Instituto Federal de Brasília, o qual pretende formar profissionais habilitados para atuar junto ao setor de informática, exercendo atividades de planejamento, execução e condução de projetos na área de TI.

O conteúdo está dividido em quatro unidades, que abordam os seguintes temas: a primeira unidade apresenta uma introdução às linguagens de programação, trata da elaboração do raciocínio lógico e da elaboração e representação de algoritmos e aborda também conceitos fundamentais da programação. Nessa unidade, vamos trabalhar também com o desenvolvimento de algoritmos utilizando o Visual G (português estruturado) apoiado na Linguagem de Programação C.

Na segunda unidade, vamos compreender como empregar as estruturas de controle e estruturas de repetição.

Na terceira unidade, trabalharemos com as estruturas de dados, vetor e matriz e aprenderemos a modularizar os algoritmos como forma de reaproveitamento de código, empregando funções e procedimentos. Realizaremos a passagem de parâmetros por valor e referência.

Por fim, na quarta unidade, aprofundaremos um pouco mais no emprego da Linguagem C. Vamos também, conhecer um paradigma de programação diferente e inovador: a Programação Orientada a Objetos, com apoio da Linguagem C++, uma extensão da Linguagem C que implementa este novo paradigma.

O nosso material está composto por exercícios dinâmicos que o auxiliarão na memorização e no aprendizado dos conteúdos estudados.

Desejo uma excelente leitura e bons estudos!


Vamos começar?
Siiiiim!
Maravilha! Hello Word!
Mas professor como podemos dominar o computador com a programação?
Programar é dar instruções claras e diretas ao computador que permitam que ele tome decisões e realize tarefas que descrevemos em nosso algoritmo. Agora é hora de ação!

Unidade 1

Introdução às Linguagens de Programação e Raciocínio Lógico

Ao término desta unidade esperamos atingir os seguintes objetivos:

- Introduzir os conceitos fundamentais e técnicas importantes para a escrita de algoritmos;
- diferenciar os diversos tipos de representações de algoritmos;
- compreender o enunciado de um problema proposto e produzir um algoritmo que leve à solução adequada do problema;
- Instalar e familiarizar-se com o VISUALg e com o Dev-C++.

1.1 Introdução às Linguagens de Programação

Neste tópico, vamos observar os aspectos que envolvem o processamento de dados e a relação existente entre o programa e o programador. Aprenderemos como a Lógica e o Raciocínio Lógico são importantes para a resolução dos problemas computacionais. Vamos desenvolver algoritmos e compreender as etapas do desenvolvimento de programas.

Vamos começar? Nosso objetivo, neste tópico, é fazer uma introdução sobre os aspectos básicos do processo de programação, abordando conceitos fundamentais e técnicas importantes para a escrita de algoritmos.

A área da tecnologia da informação promove um certo vislumbre nas pessoas que não a conhecem profundamente. Esse encanto está presente na realidade de grande parte da população mundial que constata, a cada dia, a importância dos recursos tecnológicos (comunicação e informática) como ferramentas que ampliam a nossa qualidade de vida (Figura 1).

No entanto, a quantidade de profissionais que se envolvem com esta área não consegue suprir a demanda mundial da área de TI, especialmente pelo crescente uso e evolução dos recursos tecnológicos.

Dentre as diversas subáreas existentes na aprendizagem de tecnologia, a disciplina de Lógica de Programação se destaca. Por meio dela, o estudante aprenderá a se comunicar com o computador, de forma a lhe "ensinar" a realizar tarefas complexas, que normalmente envolvem várias pessoas e processos em sua realização. O aquecimento do mercado de tecnologia da informação, particularmente o de desenvolvimento de software, tendo por base a programação, podem representar boas oportunidades para quem deseja iniciar sua carreira em TI.

Figura 1 - Recursos tecnológicos X dados e informações

Fonte: Elaborada pelo autor.

Antes de começar, todos devem estar se perguntando: por que aprender a programar? Você considera ter uma boa resposta para essa pergunta? Reflita um pouco.

É claro que você deve ter dito que é porque escolheu ser um profissional de TI ou programador! Eu entendo. Porém, agora que pensou um pouco sobre o tema, vamos assistir a um vídeo no qual e personalidades famosas respondem a essa pergunta, possibilitando que você entenda essa necessidade de maneira mais ampla!

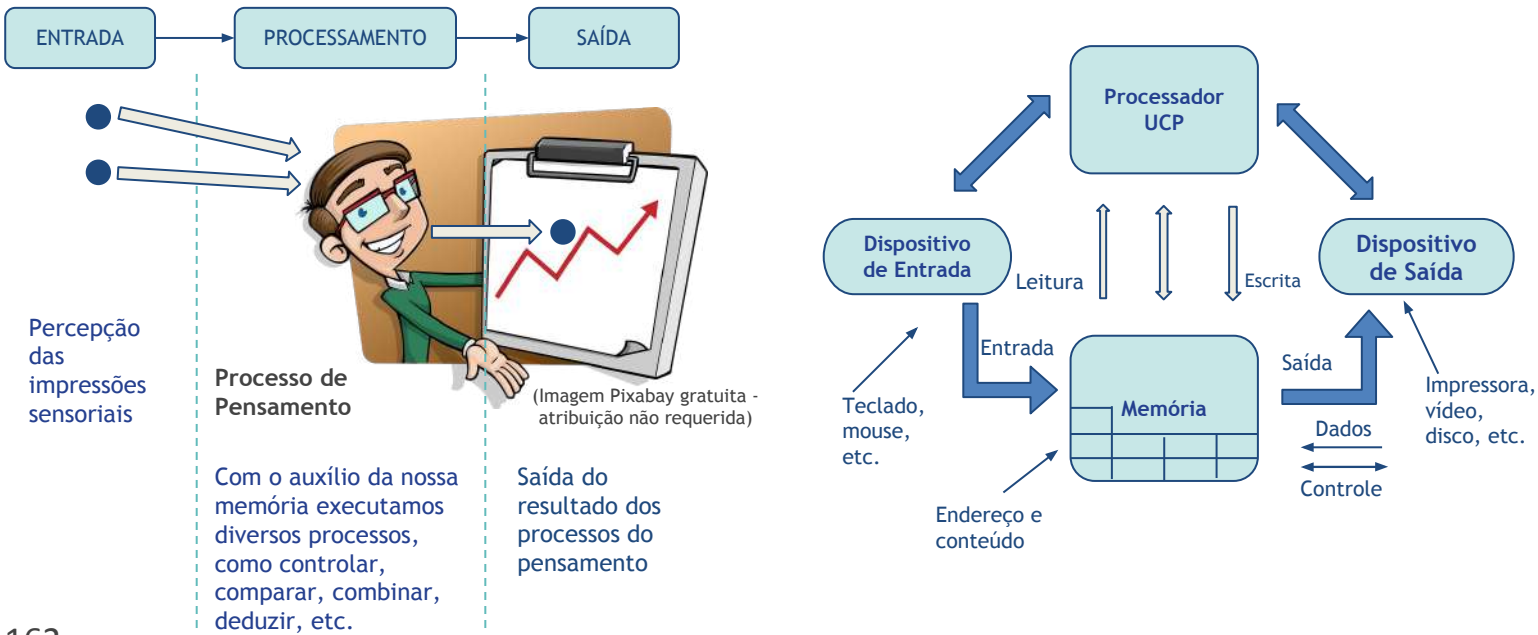
Acesse:
https://www.youtube.com/watch?v=mHW1Hsqlp6A

1.1.1 Processamento de Dados

O ato de programar um computador é análogo ao processo que realizamos para resolver um problema em nosso dia a dia. Também recebemos dados por meio de nossos "periféricos de entrada" (percepções sensoriais - olhos, ouvido, tato etc.), realizamos o processamento desses dados (auxiliados pela nossa memória e processador - cérebro) controlando, comparando ou deduzindo para chegar a um resultado que exibiremos por meio de um de nossos "periféricos de saída" (olhos, rosto - expressões, boca - fala, mãos - tato ou gestos). Observe a Figura 2.

Programar um computador, utilizando um software (programa), seguirá um modelo análogo à representação da entrada de dados, ao processamento e à saída de dados que um ser humano realiza em seu cotidiano. Cada processo desses tem uma finalidade e como vimos na arquitetura de um computador, na disciplina de introdução à informática (arquitetura), este é realizado pelos periféricos de entrada, memória, unidade central de processamento (processador) e periféricos de saída.

Figura 2 - Homem X Computador - O processamento de dados



Fonte: Elaborada pelo autor.

A velocidade de processamento de um computador é muito superior a de um ser humano. O computador pode processar dados a uma velocidade de 1 nanossegundo (1 milésimo de microssegundo), enquanto o ser humano pode levar segundos para tomar uma decisão. A automação das tarefas no computador pode acelerar esse processo ainda mais. A dificuldade está em automatizar essas tarefas de maneira que o computador possa tomar essas decisões sozinho. Como podemos fazer isso? É aqui que começamos a pensar nos programas ou algoritmos.

Vamos refletir?

Vamos entender cada etapa:

a) Entrada de dados: fornecimento de dados ao computador que os armazena em recursos eletrônicos para uso posterior.

b) Processamento: consiste na manipulação dos dados armazenados e gerados em processamentos anteriores para efetivação do processamento (transformação, cálculo, comparação etc.) desejado, de forma a gerar novos resultados esperados que, possivelmente, também serão armazenados por esses mesmos recursos eletrônicos.

c) Saída de dados: corresponde ao fornecimento dos resultados encontrados pelo processamento realizado pelo computador.

1.1.2 Programa e Programador

Os programas computacionais são sequências de instruções escritas em uma linguagem de programação a serem executadas por um computador que procura atingir um objetivo (resolver um problema) ou ajudar seu usuário a atingi-lo.

As linguagens de programação são codificações padronizadas que permitem a efetivação da comunicação desejada de forma eficiente e eficaz entre o ser humano e o computador. Porém, antes de desenvolvermos o programa, o ser humano deve encontrar a solução para o problema existente. É preciso compreender o problema para depois conseguir transpor esse processo para um algoritmo.

Por exemplo: como poderá ser resolvido o problema de venda existente em uma grande indústria de bebidas no Brasil? O que é necessário? Quais os dados de entrada? O que precisamos processar? Qual será o resultado esperado?

> Não esqueça! A primeira preocupação deve ser em conhecer muito bem o problema, pois assim será possível refletir sobre ele e, desta forma, encontrar uma solução segura e eficaz para a sua resolução.

Um dos passos fundamentais na elaboração de um programa eficiente e eficaz é o conhecimento sobre todos os detalhes possíveis que envolverão o computador e o algoritmo que será elaborado e, é claro, é necessário conhecer os detalhes da disponibilização do sistema aos seus respectivos usuários.

Portanto, quanto mais informações sobre o problema forem do conhecimento do indivíduo responsável pelo planejamento, pelo desenvolvimento e pela disponibilização do programa, menores serão as chances de processamentos incorretos ou mesmo incompleto.

O indivíduo que realiza esse tipo de atividade profissional, desenvolvendo programas ou softwares, é comumente chamado de programador.

O Desenvolvimento de sistemas de forma profissional é conhecido como o processo de Análise e Desenvolvimento de Sistemas e envolve conhecimentos de Engenharia de Software, Testes, Requisitos, Banco de Dados, Modelagem e outras disciplinas necessárias para o processo de produção de softwares em larga escala. Nesta disciplina, vamos começar pelas formas mais básicas e elementares desse processo, mas não menos importante, a programação.

1.1.3 Lógica e Raciocínio Lógico

Um raciocínio lógico "completo" para a solução de um problema envolve muitos dados, operações e informações. Não é um processo trivial e simples. A elaboração de um algoritmo pode ser algo complexo. Diante disso, torna-se essencial a organização, que permitirá a fácil representação das ações, das operações, das instruções e o reconhecimento dos dados necessários ao cumprimento adequado dos objetivos esperados. Para isso, o programador deve pensar racionalmente de forma a buscar uma lógica coerente a ser aplicada sobre o problema computacional existente, visando a encontrar uma boa solução para ele.

Podemos considerar que a Lógica é a ciência das leis ideais do pensamento e a arte de aplicá-los à pesquisa e à demonstração da verdade.

Você sabia?

"A lógica consiste em um esquema sistemático que define as interações de sinais no computador que automatiza o processamento de dados, com critério e princípios formais do raciocínio e pensamento" (MANZANO, 2003 p.3).

Para uma definição não tão técnica nesta área, a lógica poderia ser definida como "a ciência que estuda as leis e critérios de validade que regem o pensamento e a demonstração, sendo ela a ciência de princípios formais do raciocínio" (MANZANO, 2003 p.3).

Segundo o dicionário Aurélio, a lógica é a “coerência de raciocínio, de ideias”, ou, ainda, a “sequência coerente, regular e necessária de acontecimentos, de coisas”. Enfim, utiliza-se a lógica para ordenar e corrigir pensamentos ou ações voltados para a solução de problemas.

O processo de implementação na área computacional consiste, basicamente, na codificação do raciocínio humano a ser efetuado por um computador que procurará uma boa solução para o problema computacional. Esse raciocínio será descrito em uma linguagem de comunicação com o computador, denominada linguagem de programação, pois este deverá compreender as instruções necessárias ao atendimento do objetivo desejado por seu usuário e executá-las. Criar um raciocínio lógico coerente exige uma organização sobre os pensamentos a serem realizados por um indivíduo ou vários indivíduos que compõem a equipe na busca de uma solução.

A realização desse tipo de atividade demanda tempo, compreensão e cooperação dos envolvidos que precisam entender todas as situações e informações envolvidas com o problema identificado. Sendo assim, é fundamental a organização clara e simples de todos os raciocínios possíveis a serem elaborados e testados na busca da solução para um problema computacional. Para a realização desse trabalho árduo são empregados métodos e técnicas de representação do raciocínio que contribuem, significativamente, com a análise, a discussão e a averiguação das possíveis soluções apresentadas.

Vamos refletir?

O número 3 é menor que o 5
O número 7 é maior que o 5
Logo, o número 3 é menor que 5 e 7.
Todo peixe nada > Nemo é um peixe > Logo, Nemo nada.
Pela lógica deduzimos que Nemo nada.

Como vimos a lógica estuda o pensamento e, portanto, está ligada ao processo humano do raciocínio. Assim, temos lógica na linguagem falada e escrita, na matemática, nas relações humanas etc. O objetivo principal da Lógica de Programação é demonstrar técnicas para resolução de problemas e, consequentemente, para a automatização de tarefas. Como transformar um problema descrito na linguagem natural em um processo automatizado?

Vamos descobrir...

Saiba mais!

Que tal responder alguns testes de lógica?

Você está numa cela onde existem duas portas, cada uma vigiada por um guarda. Existe uma porta que dá para a liberdade e outra para a morte.

Você está livre para escolher a porta que quiser e por ela sair.

Poderá fazer apenas uma pergunta a um dos dois guardas que vigiam as portas. Um dos guardas sempre fala a verdade, e o outro sempre mente e você não sabe quem é o mentiroso e quem fala a verdade. Qual pergunta você faria e para quem?

Veja a resposta deste teste ao final desta unidade.

1.1.4 Algoritmo

Uma das principais técnicas de representação de raciocínio nesta área da Lógica é conhecida como algoritmo. Esta técnica possibilita uma representação simples e clara o suficiente para compreender a lógica racional a ser empregada na solução do problema existente.

Será possível construir algoritmos para todo tipo de problema?

Um algoritmo não é a representação de um programa de computador, mas sim os passos simples necessários para a realização de uma determinada tarefa. A execução de um algoritmo pode ser feita por um ser humano, um computador ou até mesmo um outro tipo de autômato qualquer.

Uma mesma tarefa pode ser realizada por diferentes algoritmos que utilizam conjuntos diferenciados de instruções, levando mais ou menos tempo, ocupando mais ou menos memória ou ainda com mais ou menos esforço. A diferença entre os algoritmos pode estar relacionada à complexidade computacional aplicada que depende de estruturas de dados adequadas ao algoritmo.

Você Sabia?

"Algoritmos são regras formais para a obtenção de um resultado ou da solução de um problema, englobando fórmulas de expressões aritméticas" (MANZANO, 2000, p.6).

"Algoritmo é uma sequência de passos que visa atingir um objetivo bem definido" (FORBELLONE, 2000, p.3).

"Algoritmo é a descrição de uma sequência de passos que deve ser seguida para a realização de uma tarefa" (ASCENCIO, 2002, p.2).

"Um algoritmo consiste simplesmente em uma sequência finita de regras ou instruções que especificam como determinadas operações básicas, executadas mecanicamente, devem ser combinadas para realização de uma tarefa desejada" (CAMARÃO, 2003, p.1).

Vejamos um exemplo:

O algoritmo 1 para se vestir pode especificar que você vista primeiro as meias e os sapatos antes de vestir a calça! Já o algoritmo 2 especifica que você deve primeiro vestir a calça e depois as meias e os sapatos.

Fica claro que o algoritmo 1 é mais difícil de executar que o algoritmo 2 apesar de ambos levarem ao mesmo resultado. Concorda?

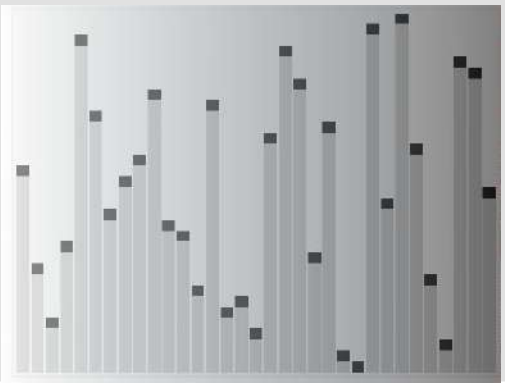
Outro aspecto importante é a complexidade associada à abstração. Problemas que possuem alto grau de abstração exigem muita complexidade na construção de um algoritmo capaz de ser automatizado por um computador.

Saiba mais!

Você já pensou em um algoritmo para ordenar números?

Acesse o link abaixo e observe a animação apresentada. Trata-se de um algoritmo para ordenação chamado QUICKSORT. Cada barra representa um número (com seu tamanho diferente). O algoritmo Quicksort é um método de ordenação muito rápido e eficiente, inventado em 1960.

Acesse:
https://pt.wikipedia.org/wiki/Algoritmo




Lembre-se: escrever um algoritmo é descrever uma sequência de passos...
Vejamos um exemplo mais prático da construção de um algoritmo para se fazer um sanduíche.

Figura 3 - Algoritmo do sanduíche

Quais são os passos para se fazer um delicioso sanduíche?

Fazer um sanduíche

1º Pegar um pão
2º Cortar o pão ao meio 3º Pegar a maionese
4º Passar maionese entre as fatias de pão
5º Pegar alface e tomate 6º Cortar alface e tomate
7º Colocar alface e tomate entre os pães
8º Pegar um hambúrguer
9º Fritar o hambúrguer
10º Colocar o hambúrguer entre as fatias
11º Juntar as duas fatias do pão

Fonte: Imagem retirada da Internet <Disponível em: https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcTRWD3n6s2mXth2viVYSzsvxemBmT2v1L5-UEeG6op14T7RY2ew>.
Acesso em: 01/02/2019.

Observando o exemplo do sanduíche descrito na Figura 3, podemos refletir sobre alguns aspectos:

a) Perceba que o algoritmo não descreve exatamente a construção do sanduíche da imagem. O que faltou? O queijo, por exemplo. Caberia então um momento de melhoria (refinamento) ao nosso algoritmo.

b) Cada um de nós poderia criar a sua receita de fazer um sanduíche adequando a sua maneira e gosto. Todos nós faríamos um sanduíche, mas a receita (algoritmo) de cada um provavelmente seria diferente. Todos nós atingiríamos o objetivo, porém com algoritmos diferentes.

c) Podem existir vários algoritmos para solucionar um problema, alguns mais rápidos, outros com mais ou menos etapas, ou ainda mais seguros. Tudo dependerá dos critérios definidos para essa avaliação de "qualidade dos algoritmos".

Vamos avaliar outros exemplos para compreender a importância de pensar logicamente. Observe a atividade a seguir na qual vamos organizar sequências lógicas para chegar a um resultado esperado.

Atividade

1) Tarefa: uma pessoa precisa colocar um carro em movimento. Quais as ações necessárias para realizar a tarefa e em qual sequência? Vamos construir esse algoritmo iniciando pela descrição da sequência de passos necessários para resolver o problema.

2) É possível perceber que o nível de abstração do problema é alto. Não temos todas as informações (requisitos) sobre o problema. Mas mesmo assim, a título de exemplo, vamos começar a imaginar o que precisamos fazer. Uma pergunta intrigante que exemplifica a falta de requisito (informações iniciais sobre o problema): estamos fora ou dentro do carro?

3) Veja um exemplo de representação de um algoritmo em linguagem natural com objetivo de colocar o carro em movimento:

Palavras reservadas são sublinhadas

Identação (alinhamento das estruturas)

início
| Entrar no carro
- Ajustar banco e espelhos retrovisores
| Colocar o cinto de segurança
- Colocar o câmbio em “ponto morto”
| Ligar o carro
- Colocar o câmbio na primeira marcha
| Pisar no acelerador
- Soltar o freio de mão
| Soltar a embreagem devagar
fim

Os algoritmos possuem ações primitivas e ações não primitivas.

As ações primitivas são aquelas que não podem ser refinadas (divididas, ou melhor, descritas) em outras ações com mais detalhes (diminuir o nível de abstração).

As ações não primitivas, pelo contrário, são aquelas que podemos detalhar mais, ou que ainda cabe alguma correção ou ajuste que melhore o nível de abstração (ações que ainda estão com o nível de abstração muito alto e podem se tornar mais claras ou exatas). Representar um algoritmo por meio da linguagem natural não é algo fácil. Na prática, ela é muito pouco usada, pois permite interpretações equivocadas, ou então, dá margem a imprecisões e ambiguidades.

Por exemplo, a instrução "colocar o câmbio em ponto morto" no algoritmo está sujeita a interpretações diferentes por pessoas distintas. Nas atividades 1 e 2, vamos ver uma instrução mais precisa.

Atividade

1) Como podemos refinar mais a ação em destaque?

Refinamento consiste em detalhar um passo...

início
| Entrar no carro
- Ajustar banco e espelhos retrovisores
| Colocar o cinto de segurança
- Colocar o câmbio em "ponto morto"
| Ligar o carro
- Colocar o câmbio na primeira marcha
| Pisar no acelerador
- Soltar o freio de mão
| Soltar a embreagem devagar
fim

Atividade

2) A ação em destaque pode ser melhor detalhada por um conjunto de passos que descrevem melhor aquela ação.

Refinamento consiste em detalhar um passo...

início
| Entrar no carro
- Ajustar banco e espelhos retrovisores
| Colocar o cinto de segurança
- **Colocar o câmbio em "ponto morto"**
| | **Pisar na embreagem**
- | - **Pegar o câmbio**
| | | **Colocar em "ponto morto"**
- | - **Soltar a embreagem**
| Ligar o carro
- Colocar o câmbio na primeira marcha
| Pisar no acelerador
- Soltar o freio de mão
| Soltar a embreagem devagar
fim

1.1.5 Etapas do Desenvolvimento de Programas Computacionais

No desenvolvimento de um algoritmo ou de um programa para computador, precisamos ter em mente um conjunto de etapas importantes:

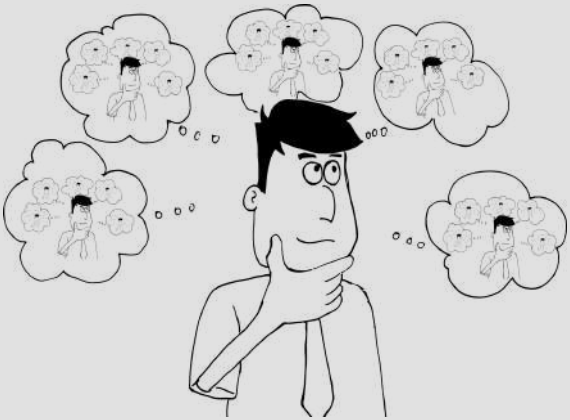
- Empenhar grande atenção sobre os dados relacionados ao problema, procurando conhecer e compreender o máximo sobre ele;
- Definir os dados que essencialmente deverão ser informados pelo usuário para que o processamento seja realizado com sucesso (dados de entrada);
- Descrever detalhadamente o processamento ou a transformação a ser executada sobre os dados de entrada em busca dos resultados desejados (como chegar ao objetivo);
- Definir quais são os dados resultantes do processamento ou transformação (dados de saída);
- Construir o algoritmo que representa a solução encontrada com o detalhamento necessário para a implementação almejada; e
- Testar o algoritmo (teste de mesa) por meio de simulações e efetuar as devidas correções que possam vir a ser necessárias na lógica proposta.

A implementação de um algoritmo é realizada no computador por meio da transcrição da lógica que está representada no algoritmo para uma linguagem de programação, que também possibilitará a compreensão e a execução das instruções pelo computador.

Essa transição também pode ser chamada de tradução, pois somente efetuará mudança na forma de descrever o raciocínio a ser implementado no computador de uma linguagem de representação para uma linguagem de programação.

Vamos refletir?

1) Processo de tradução da Lógica (pensamento)> Algoritmo (português estruturado)> Programa (linguagem de programação)

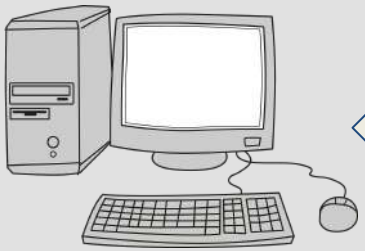


(Imagem Pixabay gratuita - atribuição não requerida)

Algoritmo

TRADUÇÃO

Linguagem de programação

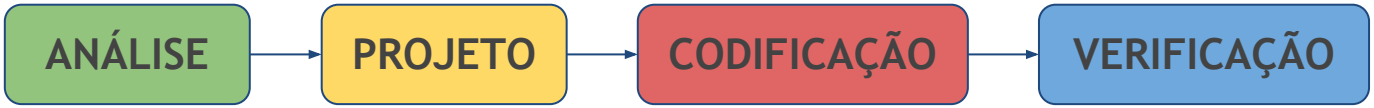


(Imagem Pixabay gratuita - atribuição não requerida)

Programa

Fonte: Elaborada pelo autor.

O uso da linguagem de programação permite ao programador elaborar programas que instruirão o computador nas operações a serem efetuadas por ele. Para a elaboração de programas, principalmente os mais complexos, é necessária a aplicação de um método sistemático de programação que contribuirá com o desenvolvimento de programas confiáveis, flexíveis e eficientes. O processo de desenvolvimento de software necessita de metodologias organizadas em algumas fases básicas fundamentais:



ANÁLISE

Nesta etapa, estuda-se o problema, buscando sua completa compreensão, por meio da correta identificação da entrada, do processamento e da saída dos dados.

PROJETO:

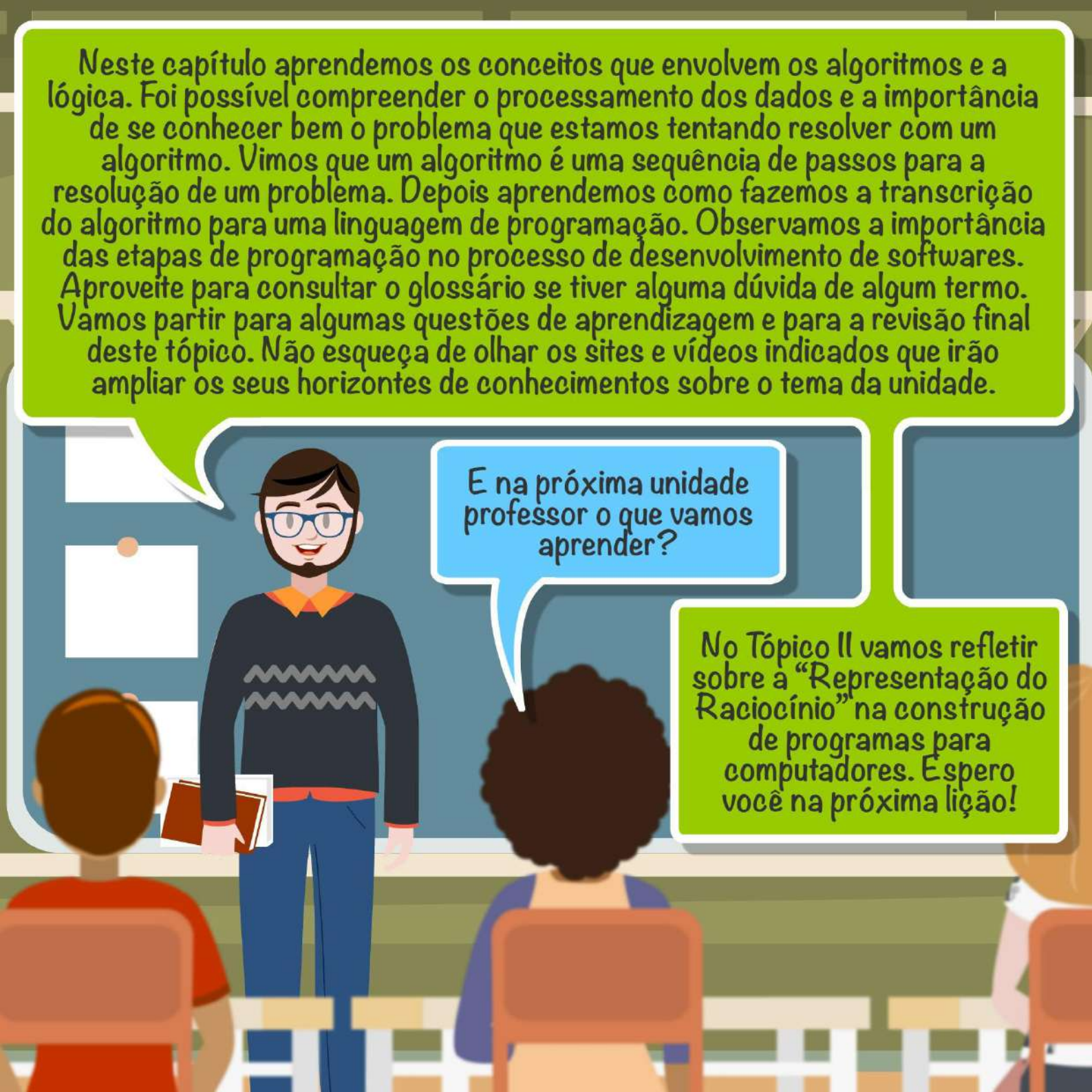
Nesta etapa, aplica-se métodos e técnicas que possibilitem a descrição necessária do problema com suas possíveis soluções. Ela consiste no projeto do programa, sendo elaborada por meio da construção do algoritmo e da definição correta das estruturas de dados necessárias para tal solução.

CODIFICAÇÃO:

Esta etapa consiste na implementação do projeto do programa, ou seja, na tradução do raciocínio, representado no algoritmo, para um programa computacional escrito na linguagem de programação desejada.

VERIFICAÇÃO

Nesta etapa é finalizado o processo de desenvolvimento do programa, que é classificado como *software*, sendo também chamado de aplicação. Simulações, testes e verificações dos resultados alcançados pelo programa são analisados, podendo resultar em alterações no código elaborado. Essas alterações procurarão o atendimento eficaz do objetivo desejado pelo programa, que é finalmente liberado aos usuários finais.


Neste capítulo aprendemos os conceitos que envolvem os algoritmos e a lógica. Foi possível compreender o processamento dos dados e a importância de se conhecer bem o problema que estamos tentando resolver com um algoritmo. Vimos que um algoritmo é uma sequência de passos para a resolução de um problema. Depois aprendemos como fazemos a transcrição do algoritmo para uma linguagem de programação. Observamos a importância das etapas de programação no processo de desenvolvimento de softwares. Aproveite para consultar o glossário se tiver alguma dúvida de algum termo. Vamos partir para algumas questões de aprendizagem e para a revisão final deste tópico. Não esqueça de olhar os sites e vídeos indicados que irão ampliar os seus horizontes de conhecimentos sobre o tema da unidade.
E na próxima unidade professor o que vamos aprender?
No Tópico II vamos refletir sobre a "Representação do Raciocínio" na construção de programas para computadores. Espero você na próxima lição!

Vamos rever?

Este tópico demonstrou que o crescimento no uso de tecnologias é um fator exponencial e que os recursos tecnológicos estão cada vez mais presentes no nosso dia a dia. Cada vez mais temos necessidade de organizar e processar dados e informações utilizando computadores. A programação é um processo fundamental para automatizar essas tarefas em computadores. O mercado de TI, particularmente o de programação, está cada vez mais aquecido. Programar está se tornando uma necessidade para qualquer cidadão. Os computadores processam dados de forma análoga a nós, seres humanos. Assim como nós, eles dependem de dados de entrada, realizam processamento com esses dados e elaboram um resultado final, a saída de dados. Para que um computador resolva um problema, precisamos inicialmente empregar a lógica para transpor os passos de resolução daquele problema para um algoritmo e depois transcrever esse algoritmo em uma linguagem de programação de forma que o computador entenda. Desenvolver programas (softwares), ou seja, ser um programador, requer a aplicação de metodologias que permitam a busca por qualidade e eficiência. Naturalmente quando desenvolvemos um algoritmo e depois codificamos um programa a partir dele, realizamos algumas fases básicas fundamentais: análise, projeto, codificação e verificação (testes). Estes processos de desenvolvimento permitem uma entrega adequada do sistema ao usuário final, passando por testes e permitindo a manutenção corretiva e evolutiva do programa (refinamentos).

Sites indicados

1) Racha a Cuca - site de jogos de raciocínio lógico
https://rachacuca.com.br/

2) Pescadores de Vidas - Desafio do Código para aprender a programar:
https://pescadoresdevidas.org.br/desafio/

3) Jogos para os programadores de amanhã:
https://blockly-games.appspot.com/?lang=pt-br

4) A hora do código:
https://code.org/
https://studio.code.org/s/20-hour

Audiovisuais indicados

1) Por que todos deveriam aprender a programar?
https://www.youtube.com/watch?v=mHW1Hsqlp6A

2) O que é algoritmo? - Curso de Algoritmo #01
https://www.youtube.com/watch?v=8mei6uVttho

3) Tudo que você precisa saber antes de estudar programação - Vídeo 01
https://www.youtube.com/watch?v=23G_vP9Mupo

Questões de autoaprendizagem

1) Construa um algoritmo em linguagem natural para “escovar os dentes”. Compare com o algoritmo do seu colega. Os dois resolvem o problema? O que eles têm de diferente?

2) Você está em uma escola que possui 5 salas de aula com lâmpadas que podem estar queimadas. Seu projeto é fazer um algoritmo para um robô ir em cada sala da escola (sala de aula de 1 a 5) e trocar a lâmpada que estiver queimada. Observe que o robô já está no corredor que dá acesso às salas e que em uma caixa ao seu lado estão as 5 lâmpadas novas. Construa um algoritmo em linguagem natural para “o robô trocar as lâmpadas queimadas da escola”. Compare com o algoritmo do seu colega. Os dois resolvem o problema? O que eles têm de diferente?

3) Pesquise cinco Linguagens de Programação mais utilizadas na atualidade. Para que tipo de problema elas se aplicam? Quais as que você já conhecia ou tinha ouvido falar?


Olá pessoal! Vamos dar mais um passo em frente em programação? Neste tópico vamos dialogar sobre as diversas representações do raciocínio na elaboração de algoritmos.
Opa! Só se for agora! Vamos logo programar!
Calma gente. Um passo de cada vez. Primeiro a base e os fundamentos, depois o código. Certo, vamos começar...

1.2 Representação do Raciocínio

Neste tópico denominado Representação do Raciocínio, vamos aprender a distinguir entre as diversas representações de algoritmos existentes e como fazemos para representar o nosso raciocínio empregando uma dessas formas. Depois vamos olhar com um pouco mais de detalhes como se elabora um algoritmo. Finalizando, vamos explorar as linguagens de programação existentes.

Bem, você deve se lembrar do filme “Harry Potter”, no qual o mágico Dumbledore, utilizando sua varinha mágica retirava pensamentos de sua cabeça e os colocava na “penseira”. Após colocar a varinha na têmpora e tirar um fio prateado, colocam-se os pensamentos na penseira e eles tomam forma. Mas elaborar um algoritmo a partir de um raciocínio lógico é assim? É claro que não! Vamos dar uma olhada em como podemos fazer isso então...

1.2.1 Representação

A correta compreensão do que cada indivíduo pensa pode consistir em um grande desafio para pessoas que precisam entender os esclarecimentos ou explicações elaboradas por este indivíduo. Muitas vezes, um especialista tem dificuldade de explicar seu raciocínio para que outras pessoas possam entendê-lo. Aqui está uma questão chave na programação. Transferir o raciocínio que elaboramos para uma forma de representação na qual outras pessoas possam compreender.

Figura 4 - Dumbledore - Mágico do filme Harry Potter retirando pensamentos

Fonte: Retirada da internet
Disponível em: https://en.wikipedia.org/wiki/Albus_Dumbledore
https://pbs.twimg.com/media/C2VI6vIWgAA80kv.jpg

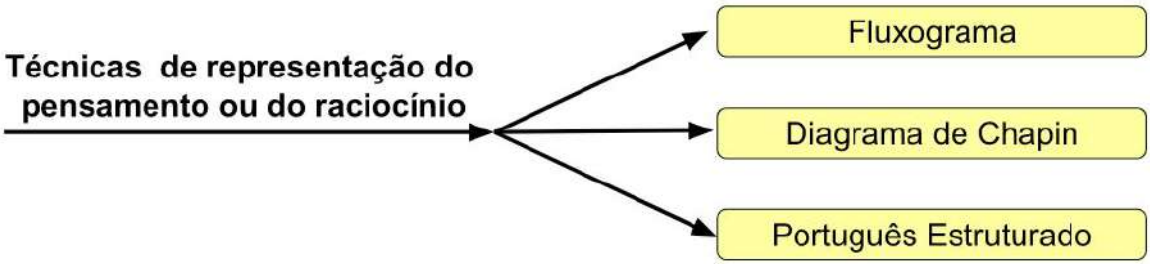
No entanto, quase todo mundo já passou por uma situação similar, em que foi necessário explicar o que se estava pensando. Com certeza, você deve ter tentado expor suas ideias da forma mais fiel possível, mas nem sempre as pessoas entendem adequadamente o que você quer apresentar. Existem várias maneiras diferentes de expor nossos pensamentos para que outras pessoas compreendam e possam refletir conosco.

A elaboração de algoritmo, para área de Tecnologia, procura justamente realizar esse tipo de compartilhamento de pensamentos e ideias antes do desenvolvimento do programa computacional (da codificação propriamente dita). Essa habilidade de exposição de ideias é exercitada por todos os envolvidos nessa área, sendo um dos principais conteúdos abordados por esta disciplina. Essa importante habilidade, fundamental aos responsáveis pelo desenvolvimento de programas, também é conhecida como Representação do Raciocínio Lógico ou de Dados.

1.2.2 Elaboração do algoritmo

A elaboração do algoritmo descreve os dados e as suas manipulações. Essa descrição pode ser feita de diversas formas e por meio de técnicas diferentes que representarão a sequência dos passos (ou etapas) a serem realizados pela execução do algoritmo. Essa representação da sequência lógica envolvida corresponde a um dos principais pontos de estudo e prática desta disciplina.

Figura 5 - Técnicas de representação do raciocínio

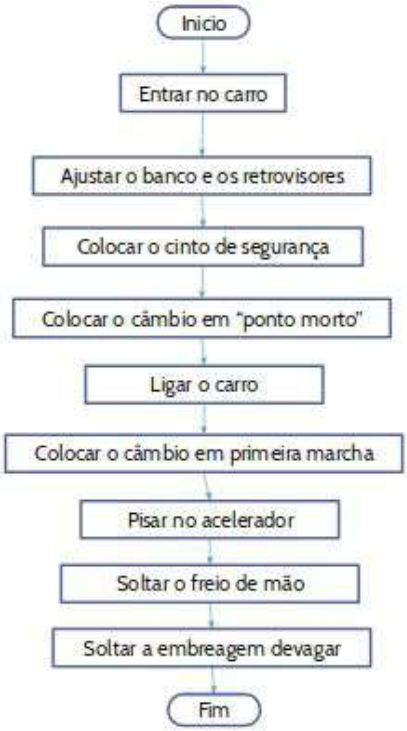


Fonte: Elaborada pelo autor.

1.2.3 Fluxograma

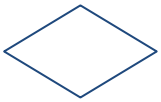
O fluxograma é uma técnica de representação que utiliza figuras geométricas pré-definidas para descrever ações (ou instruções) a serem realizadas na resolução de um problema. Ele é elaborado após a fase de análise do problema. Existem várias figuras geométricas empregadas na construção do fluxograma.

Vejamos um exemplo:

Fluxograma	Português Estruturado
	**início** Entrar no carro Ajustar banco e espelhos retrovisores Colocar o cinto de segurança Colocar o câmbio em "ponto morto" Ligar o carro Colocar o câmbio primeira marcha Pisar no acelerador Soltar o freio de mão Soltar a embreagem devagar **fim**

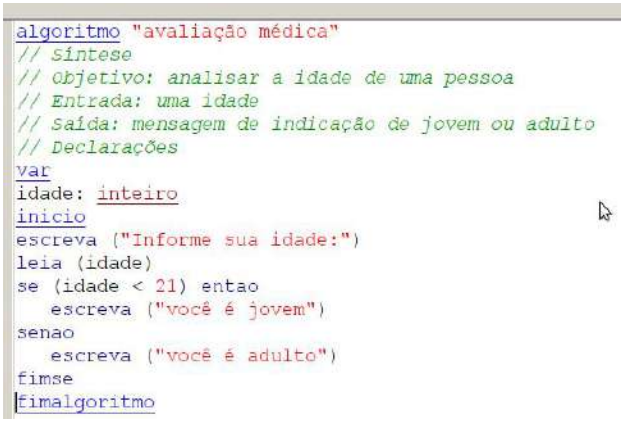
Cada figura do exemplo acima tem um significado. Perceba que a coluna da esquerda (fluxograma) representa (equivale) a coluna da direita (português estruturado). A seguir, um pequeno exemplo de um conjunto básico de figuras e seus significados para representação de algoritmos.

Figura 6 - Representações utilizadas em Fluxograma

FIGURA	SIGNIFICADO
	Figura para definir início e fim do algoritmo
	Figura usada no processamento de cálculo, atribuições e processamento de dados em geral
	Figura utilizada na representação de entrada de dados
	Figura utilizada para representação da saída de dados
	Figura que indica o processo seletivo ou condicional, possibilitando o desvio no caminho do processamento
	Símbolo geométrico usado como conector
	Símbolo que identifica o fluxo de dados, permitindo a conexão entre as outras figuras existentes

Fonte: Elaborada pelo autor.

A representação facilita a compreensão. Existem outras formas mais atuais de representação, especialmente quando estudarmos em Engenharia de Software. Imagine uma situação em que a idade de uma pessoa deva ser analisada para classificá-la em jovem ou adulta. O algoritmo a seguir é apresentado graficamente por meio do fluxograma coerente com a situação sugerida como exemplo.

Fluxograma	Português Estruturado (Visual G)
	

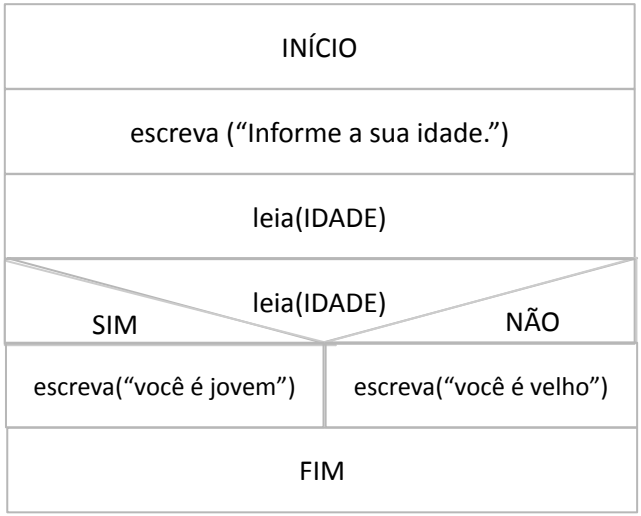

```
algoritmo "avaliação médica"
// Síntese
// Objetivo: analisar a idade de uma pessoa
// Entrada: uma idade
// Saída: mensagem de indicação de jovem ou adulto
// Declarações
var
idade: inteiro
inicio
escreva ("Informe sua idade:")
leia (idade)
se (idade < 21) entao
   escreva ("você é jovem")
senao
   escreva ("você é adulto")
fimse
fimalgoritmo
```

1.2.4 Diagrama de Chapin

Esta forma de representação foi elaborada por Nassi e Shneiderman e ampliada por Ned Chapin (apud ANZANO, 2000). O Diagrama de Chapin busca a substituição da representação tradicional (diagrama de blocos - fluxograma) por uma diagramação com quadros que ofereça uma visão hierárquica e estruturada da lógica proposta para um programa.

Entre os diversos métodos existentes para esse tipo de representação, o Diagrama de Chapin é um dos menos utilizados atualmente, principalmente porque exige bastante atenção do programador na representação do seu raciocínio.

Vejamos o exemplo a seguir:




```
//-------------------------------------------------------------------
// Exercícios - Prof. Ronald Costa
// Lógica de Programação
//-------------------------------------------------------------------
algoritmo “Idade”
// Síntese
// Objetivo: analisar a idade de uma pessoa
// Entrada: uma idade
// Saída: mensagem de indicação de jovem ou adulto
// Declarações
var
idade= inteiro
inicio
escreva (“Informe sua idade:”)
leia (idade)
    se (idade < 25) então
        escreva (”você é jovem”)
     senão
         escreva (”Você é velho”)
     fimse
fimalgoritmo
```

1.2.5 Português Estruturado

Esta forma de representação, também conhecida como portugol ou pseudocódigo, consiste na descrição estruturada, por meio de regras pré-definidas, de fases (ou passos) a serem realizadas para a resolução do problema. O Português Estruturado utiliza a linguagem natural para representar o raciocínio lógico desenvolvido pelo programador. No Brasil, a linguagem de comunicação natural é o Português. Isso facilitará muito a representação dos algoritmos empregando o português estruturado na representação dos raciocínios lógicos propostos em exercícios de aprendizagem nesta disciplina. Suponha o mesmo exemplo representado nos algoritmos anteriores por meio do fluxograma e do Diagrama de Chapin, observando suas diferentes características descritivas existentes no português estruturado, além de sua organização pré-definida.

A linguagem de comunicação natural é imprecisa e muitas vezes prolixa, dificultando a comunicação correta com o computador quando utilizada de forma bem natural.

Por isso, é tão necessária a incorporação de normas e regras que permitam a representação organizada e eficiente da solução desejada, indicando sempre, com clareza e precisão, o que realmente o computador deve fazer. Por meio dessa última representação de raciocínio (português estruturado), alia-se a facilidade no uso da linguagem natural do indivíduo com a técnica de desenvolvimento estruturado. Apesar disso, essa junção, por si só, ainda não atinge automaticamente os objetivos almejados, mas preconiza uma ajuda sistemática para o alcance desse objetivo. Sua aplicação ainda conta com o esforço e a disciplina incessante na busca da simplicidade e da clareza, que fornecerão facilidades inestimáveis à manutenção e à modificação evolutiva do algoritmo.

A elaboração de algoritmos pode envolver três estruturas lógicas fundamentais no controle do fluxo de dados e instruções. Essas três estruturas, conhecidas como estruturas de controle de dados, desempenham o papel de controle sobre a sequência de ações (ou tarefas) a serem realizadas (ou executadas) no algoritmo, que posteriormente se tornará um programa computacional, por meio de sua transcrição (ou tradução) para uma linguagem de programação. A principal característica referente ao controle na sequência de execução de cada uma dessas estruturas é descrita a seguir:

Sequencial:

As instruções existentes no algoritmo são executadas uma após a outra, respeitando sempre a sequência linear de cima para baixo.

Seletiva:

Exerce o controle sobre a sequência de instruções a serem executadas, por meio do resultado de um teste ou verificação baseada na lógica convencional, conhecida também como condicional.

Repetitiva:

Por meio de um teste, ou verificação lógica condicional, uma instrução ou um conjunto de instruções, é executado repetidamente (mais de uma vez), conforme discriminado no raciocínio lógico proposto. Essa estrutura ainda é conhecida como laço ou looping.

Um outro aspecto lógico importante, e muito aplicado na elaboração de algoritmos, é a modularização, isto é, a divisão de um raciocínio lógico maior, ou mais complexo, em vários raciocínios menores, ou mais simples, que possibilitem a realização ou o atendimento completo do raciocínio lógico maior (mais complexo).

Essa organização lógica é conhecida como modularização, pois confere a um conjunto de módulos a capacidade de resolver o mesmo problema, porém de forma mais organizada e flexível.

Isso possibilita, entre outras características positivas no desenvolvimento de programas, o reaproveitamento do código já elaborado, ou seja, o melhor aproveitamento de trechos bem definidos e já elaborados no algoritmo proposto, em diversos momentos que sejam necessários ao mesmo.

As estruturas de dados consistem em organizações eficientes sobre os recursos de armazenamento de dados, normalmente disponíveis em um computador. A utilização logicamente correta desse tipo de estrutura propicia a otimização no uso dos recursos disponibilizados com o emprego do computador na realização das mais diversas tarefas, satisfazendo as pretensões do usuário com maior segurança, eficiência e agilidade.

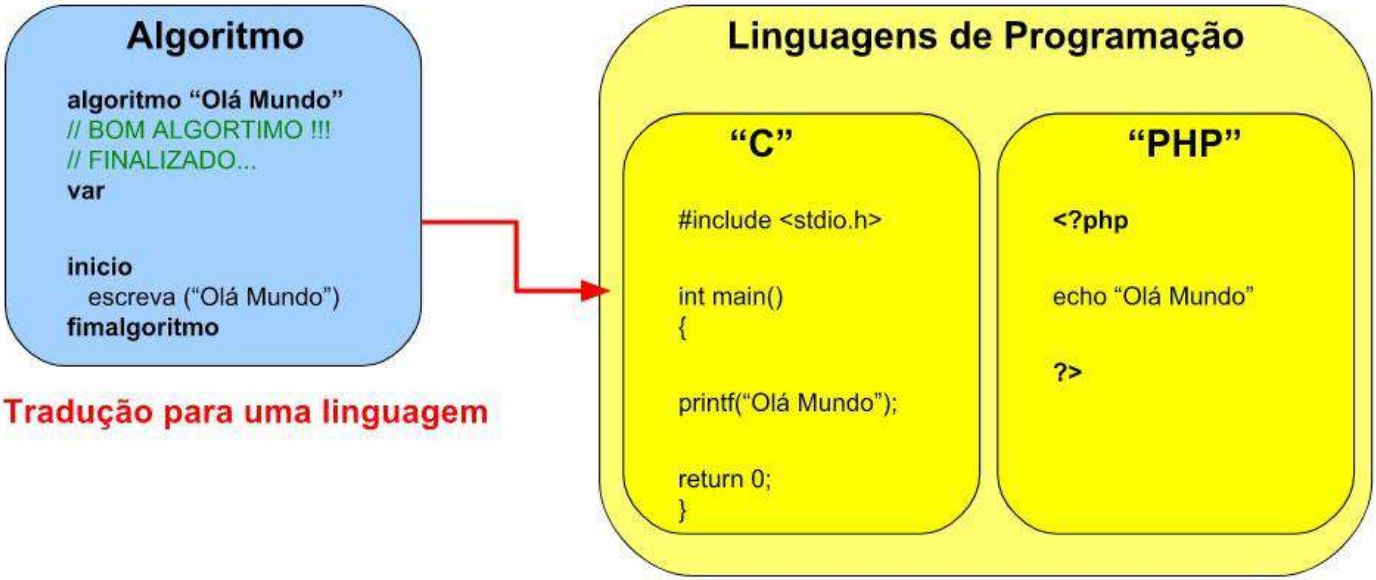
Vamos refletir?

O algoritmo é a parte mais importante no desenvolvimento de um programa computacional, pois é nele que se encontra a representação do raciocínio lógico que solucionará o problema existente.

1.2.6 Linguagem de Programação

O algoritmo auxilia o desenvolvimento dos programas, porque por meio dele os programadores representam a solução proposta para ser analisada e testada, antes de ser empregada em uma atividade. Quando um algoritmo está finalizado, ou seja, está pronto, resta apenas transformá-lo em um programa computacional, respeitando toda lógica racional elaborada e já avaliada para a solução do problema em questão. É claro que didaticamente aprenderemos primeiro em português estruturado utilizando VISUALg e depois, em um segundo momento, faremos a codificação para uma ou mais linguagens de programação.

Figura 7 - Processo de tradução (codificação) em uma linguagem



Fonte: Elaborada pelo autor

Durante a elaboração do algoritmo, estudamos e aprofundamos os conhecimentos sobre o problema, elaboramos uma proposta de solução, realizamos testes e ajustes necessários para a correta solução do problema. Depois é feita a reescrita (tradução ou codificação) da solução em uma linguagem de programação. As linguagens de programação possuem regras sintáticas e semânticas que devem ser obedecidas. Correspondem, respectivamente, à forma como seus termos, expressões e instruções são descritos no corpo do programa e ao significado que esses elementos da linguagem possuem para a execução do programa. Existem várias linguagens de programação e, em todas elas, o raciocínio lógico relacionado à coerente proposta elaborada no algoritmo pode ser implementada, a fim de atender ao objetivo desejado pelo usuário do computador (ou sistema). Lembre-se: o português estruturado é uma técnica de representação do raciocínio lógico e não uma linguagem de programação. Fazer a tradução do português estruturado para uma linguagem é relativamente fácil, desde que você conheça a sintaxe da linguagem para a qual pretende fazer a transcrição.

Veja a seguir um exemplo de um comando do português estruturado sendo codificado em diversas linguagens:

Português Estruturado	Pascal	C	Java
Escreva	writeln	printf	System.out.println

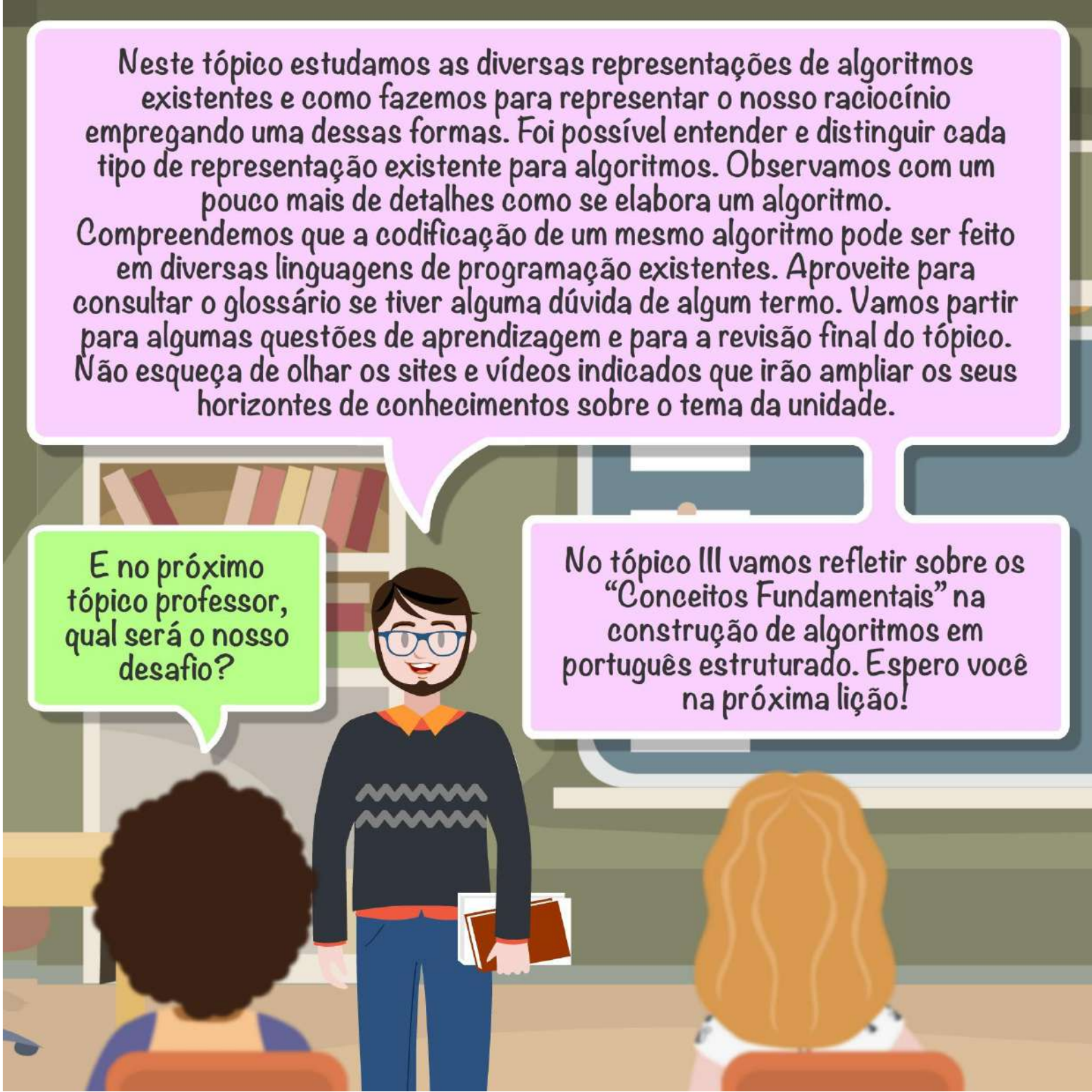
Para se tornar um bom programador é necessário envolvimento e dedicação para compreender o problema e implementar um algoritmo que o resolva com a lógica adequada. Não se esqueça: quem pensa é o ser humano e não o computador. Como programador você dará ao computador uma receita (algoritmo ou conjunto de passos) para resolver o problema.

Saiba mais!

Quais são as 15 principais linguagens de programação do mundo?

Descubra quais são as principais linguagens de programação segundo os rankings mais conceituados do mercado: IEEE Spectrum, TIOBE e Redmonk! Aprofunde um pouco mais lendo esse conteúdo interessante.

Acesse: https://becode.com.br/principais-linguagens-de-programacao/


Neste tópico estudamos as diversas representações de algoritmos existentes e como fazemos para representar o nosso raciocínio empregando uma dessas formas. Foi possível entender e distinguir cada tipo de representação existente para algoritmos. Observamos com um pouco mais de detalhes como se elabora um algoritmo. Compreendemos que a codificação de um mesmo algoritmo pode ser feito em diversas linguagens de programação existentes. Aproveite para consultar o glossário se tiver alguma dúvida de algum termo. Vamos partir para algumas questões de aprendizagem e para a revisão final do tópico. Não esqueça de olhar os sites e vídeos indicados que irão ampliar os seus horizontes de conhecimentos sobre o tema da unidade.
E no próximo tópico professor, qual será o nosso desafio?
No tópico III vamos refletir sobre os "Conceitos Fundamentais" na construção de algoritmos em português estruturado. Espero você na próxima lição!

Vamos rever?

A correta compreensão do que cada indivíduo pensa pode consistir em um grande desafio para pessoas que precisam entender os esclarecimentos ou explicações elaboradas por este indivíduo. Muitas vezes, um especialista tem dificuldade de explicar seu raciocínio para que outras pessoas possam entendê-lo. Aqui está uma questão chave na programação. Transferir o raciocínio que elaboramos para uma forma de representação que outras pessoas possam compreender. A elaboração de algoritmo, para área de Tecnologia, procura justamente realizar esse tipo de compartilhamento de pensamentos e ideias antes do desenvolvimento do programa computacional (da codificação propriamente dita). A elaboração do algoritmo descreve os dados e as suas manipulações. Essa descrição pode ser feita de diversas formas e por meio de técnicas diferentes que representarão a sequência dos passos (ou etapas) a serem realizados pela execução do algoritmo. Podemos representar o raciocínio lógico utilizando Fluxograma, Diagramas e Português Estruturado. O fluxograma é uma técnica de representação que utiliza figuras geométricas pré-definidas para descrever ações (ou instruções) a serem realizadas na resolução de um problema. O Diagrama de Chapin busca a substituição da representação tradicional (diagrama de blocos - fluxograma) por uma diagramação com quadros que ofereça uma visão hierárquica e estruturada da lógica proposta para um programa. O português estruturado (portugol ou pseudocódigo) consiste na descrição estruturada, por meio de regras pré-definidas, de fases (ou passos) a serem realizadas para a resolução do problema. O Português Estruturado utiliza a linguagem natural para representar o raciocínio lógico desenvolvido pelo programador. A elaboração de algoritmos pode envolver três estruturas lógicas fundamentais no controle do fluxo de dados e instruções (sequencial, seletiva e repetitiva). Essas três estruturas, conhecidas como estruturas de controle de dados, desempenham o papel de controle sobre a sequência de ações (ou tarefas) a serem realizadas (ou executadas) no algoritmo, que posteriormente se tornará um programa computacional, por meio de sua transcrição (ou tradução) para uma linguagem de programação.

Sites indicados

1) VISUALg - Apoio Informática
http://www.apoioinformatica.inf.br/produtos/visualg

2) Revista Eixo - IFB - Cartilha de Lógica de Programação
http://revistaeixo.ifb.edu.br/index.php/editoraifb/article/view/179

3) Lógica de Programação - Fluxograma e Portugol
http://academicotech.blogspot.com.br/2014/02/v-behaviorurldefaultvmlo.html

4) Fluxogramas, diagrama de blocos e de Chapin no desenvolvimento de algoritmos
https://www.devmedia.com.br/fluxogramas-diagrama-de-blocos-e-de-chapin-no-desenvolvimento-de-algoritmos/28550

Audiovisuais indicados

1) Curso de Lógica de Programação - Gustavo Guanabara
https://www.youtube.com/watch?v=M2Af7gkbbro&list=PLHz_AreHm4dmSj0MHol_aoNYCSGFqvfXV&index=2

2) Lógica de programação - Aula 01 - Introdução
https://www.youtube.com/watch?v=Ds1n6aHchRU

3) Lógica de Programação | Vídeo Aula COMPLETA
https://www.youtube.com/watch?v=PbRkAwZnQCU

Questões de autoaprendizagem

Crie um fluxograma que representa o algoritmo para os seguintes problemas:

1) Ler três números e mostrar o resultado da multiplicação desses números.
2) Calcular a área de um círculo.
3) Ler 5 números e mostrar a média aritmética desses números.


Olá pessoal! Sejam bem vindos ao Tópico III onde vamos estudar os "Conceitos Fundamentais" de algoritmos.
Como começamos a criar um algoritmo professor? Existe alguma regra para escrever um algoritmo de maneira que o computador entenda?
No tópico III vamos refletir sobre os "Conceitos Fundamentais" na construção de algoritmos em português estruturado. Espero você na próxima lição!

Neste tópico, denominado Conceitos Fundamentais, vamos compreender as regras básicas para a construção de algoritmos. Precisamos conhecer a estrutura básica de um algoritmo, os tipos de dados que podemos empregar, aprender a definir identificadores de variáveis e constantes. Depois vamos trabalhar os comandos mais básicos de entrada e saída, empregar operadores de atribuição, aritméticos, relacionais e lógicos. É hora de colocar a mão no código!

1.3 Representação de Algoritmos

1.3.1 Estrutura do Algoritmo em Português Estruturado

Compreender algoritmos é fundamental para a formação de um programador, quer seja mobile, web ou desktop, afinal qualquer programa de computador é baseado em algoritmos. O primeiro elemento que devemos conhecer no processo de criação de um algoritmo em Português estruturado é a sua estrutura geral. A elaboração de um algoritmo deve respeitar as normas estabelecidas em cada método ou técnica de representação de raciocínio.

Cuidado com os alinhamentos das suas principais palavras reservadas e pontuações existentes em cada linha descritiva, pois, no algoritmo em português estruturado, essa endentação correta é obrigatória.

Vejamos um exemplo:

Algoritmo exemplo

Todo algoritmo deve ter um início e um fim definidos.

```
algoritmo
        ----------
        ----------
        ----------
fimalgoritmo
```

```
algoritmo "modelo geral"
  // Síntese
  // Objetivo: <descrição resumida do que o algoritmo descrito abaixo realiza>
  // Entrada: <valores lidos ou coletados pelos recursos de entrada de dados>
  // Saída: <valores apresentados pelos recursos de saídas de dados>
  //
   Declarações
   var
      <constantes>
      <variáveis>
      <subprogramas>
   inicio
      <bloco de instruções>
fimalgoritmo
```

Vamos identificar três blocos importantes da estrutura geral de um algoritmo. Observe:

```
algoritmo "modelo geral"

   // Síntese
   // Objetivo: <descrição resumida do que o algoritmo descrito
abaixo realiza>
   // Entrada: <valores lidos ou coletados pelos recursos de
entrada de dados>
   // Saída: <valores apresentados pelos recursos de saídas de
dados>

   // Declarações
    var
      <constantes>
      <variáveis>
      <subprogramas>

   inicio
      <bloco de instruções>
fimalgoritmo
```

1 Algoritmo "modelo geral"

- Identificação do nome do algoritmo proposto;
- Consiste em uma palavra ou expressão significativa para a finalidade de sua aplicação;
- Obrigatório o uso de aspas para delimitar o nome.

1 Síntese do problema

```
   // Síntese
   // Objetivo: <descrição resumida do que o algoritmo descrito
abaixo realiza>
   // Entrada: <valores lidos ou coletados pelos recursos de
entrada de dados>
   // Saída: <valores apresentados pelos recursos de saídas de
dados>
```

- Sempre iniciado pela palavra reservada Síntese como comentário, ou seja, contendo // (barra barra) antes e iniciando este primeiro bloco de representação em português estruturado.

- A Síntese é formada por três itens relevantes ao processamento de dados, sendo eles o Objetivo (descreve, resumidamente, o que o algoritmo faz), a Entrada (indica, resumidamente, todas as entradas de dados para este algoritmo) e a Saída (indica, sucintamente, todas as saídas de dados fornecidas por este algoritmo).

2 Bloco de declarações

```
// Declarações
 var
    <constantes>
    <variáveis>
    <subprogramas>
```

- Sempre iniciado pela palavra reservada Declarações como comentário, ou seja, // Declarações e seguido da palavra reservada var um pouco mais a sua direita (atenção com a endentação correta é obrigatória no português estruturado).

- Neste bloco são criados e definidos os recursos ou objetos computacionais (variável, constante e subprogramas) que serão utilizados pelo algoritmo em sua execução.

- Este bloco termina na palavra reservada início e pode não conter nenhuma instrução de acordo com a necessidade da solução proposta no algoritmo.

3 Bloco de instruções

```
inicio
  <bloco de instruções>
fimalgoritmo
```

- Inicia a partir da palavra reservada início, sendo encerrado somente com a palavra reservada fimalgoritmo.
- Contém todas as instruções ou comandos que descrevem o que o algoritmo deve fazer (ações ou processamentos realizados pelo algoritmo proposto).
- Este bloco terá no mínimo uma instrução, caso contrário, o algoritmo não existiria, pois não teria finalidade alguma.

Toda linguagem de programação possui palavras reservadas que fazem parte de sua sintaxe. Assim em português estruturado temos algumas palavras que não podem ser usadas para outro propósito em um algoritmo que não seja aquele previsto nas regras de sintaxe. A palavra fimalgoritmo, por exemplo, é uma palavra reservada e aparece sublinhada e destacada em azul no VISUALg.

Os comentários são linhas iniciadas por duas barras (//) que são ignoradas pelo interpretador, ou seja, elas podem conter qualquer informação depois das duas barras até o final da linha que não serão executadas pelo programa. Utilizamos os comentários para adicionar ao algoritmo uma referência, documentação ou explicação adicional que ajude a facilitar o entendimento de quem estiver lendo o código.

No exemplo abaixo as linhas de comentários, iniciadas com duas barras, estão descritas na cor verde:

```
algoritmo "semnome"
// Função :
// Autor :
// Data : 19/11/2017
// Seção de Declarações
```

1.3.2 Tipos de Dados

Para que um computador possa realizar um determinado processamento, normalmente, é necessário que ele tenha acesso a informações, ou seja, dados para realizar o processamento. Os dados, ou informações, em algoritmos estão delimitados em quatro tipos básicos de dados: inteiros, reais (com casas decimais), caracteres e dados lógicos.

O tipo de dado representa a especificação do grau de complexidade e o escopo de valores possíveis para o dado em questão.

Tipo de dado	Definição	Exemplos
Inteiro	Corresponde ao conjunto matemático dos valores numéricos inteiros positivos ou negativos (não fracionários)	32 0 -89
Real	Corresponde ao conjunto matemático dos valores numéricos reais positivos ou negativos (incluindo os fracionários - decimais)	1 -2.2 78.9876 -100.9
Caractere	Corresponde aos valores alfanuméricos, contendo letras, números e símbolos especiais, estando sempre descritos entre aspas (“ “). O conjunto de caracteres também é chamado de cadeia de caracteres ou string, quando estiver definido com um único ou mais carácter.	“M” (um carácter) “Taguatinga - DF” “356-9025” “Taxa=10%”
Lógico	Corresponde ao conjunto de valores possíveis na lógica convencional (booleana), ou seja, FALSO ou VERDADEIRO. Esse tipo de dado só apresenta um entre esses dois valores, sendo estes mutuamente exclusivos.	verdadeiro/true (T) falso/false (F)

1.3.3 Identificadores

Identificador é o nome fornecido a um recurso computacional que o identifica distintamente para acesso e manipulação do computador na execução ou realização de um algoritmo. Todos os recursos de armazenamento e manipulação de dados devem possuir nomes para sua correta identificação e uso.

Para criação de identificadores é necessário observar algumas regras convencionadas:

- todo identificador deve começar com o primeiro carácter sendo alfabético (pertence ao alfabeto) em letra minúscula;
- os demais caracteres do identificador podem ser letras do alfabeto (“a, b, c,...”), caracteres numéricos (“1, 2, 3,...”) ou o caractere underline (linha baixa “_”);
- nenhum identificador pode ser igual a uma palavra reservada;
- Na sintaxe do Português Estruturado, não há diferença entre letras maiúsculas de
- minúsculas (NOME é o mesmo que noMe);
- nomes de variáveis devem ter no máximo 127 caracteres;
- nenhum caractere especial pode fazer parte do identificador, inclusive o espaço em branco entre palavras, sendo a única exceção o underline;
- identificadores compostos devem possuir o primeiro carácter da segunda palavra ou expressão que compõe o identificador em letras maiúsculas, sendo as demais sempre em minúsculas (valorTotal ou receitaArrecadadaMensal).

Exemplos:

Identificadores válidos:
IDADE, FONE, TIPO_FILHO, IdadeAluno, SOMA1, Estado_Civil

Identificadores inválidos:
3Pais, Tipo Telefone, PARA, algoritmo, end/bairro

Observe que as palavras “PARA e algoritmo” são identificadores inválidos. Mas porque isso ocorre? É simples! Os dois são inválidos, pois são palavras reservadas da linguagem! Veja a seguir outras palavras que você não deve utilizar como identificadores de variáveis ou constantes.

PALAVRAS RESERVADAS						
aleatorio	caso	enquanto	fimfuncao	literal	ou	raizq
abs	compr	entao	fimprocedimento	log	outrocaso	rand
algoritmo	copia	escolha	fimrepita	logico	para	randi
arccos	cos	escreva	fimse	logn	passo	repita
arcsen	cotan	exp	funcao	maiusc	pausa	se
arctan	cronometro	faca	grauprad	mensagem	pi	sen
arquivo	debug	falso	inicio	minusc	pos	senao
asc	declare	fimalgoritmo	int	nao	procedimento	timer
ate	e	fimequanto	interrompa	numerico	quad	verdadeiro
caracter	eco	fimescolha	leia	numpcarac	radpgrau	xou

1.3.4 Variáveis

A variável é o recurso ou objeto computacional de armazenamento de dados sujeito à variação em seus valores. Uma variável é um espaço reservado na área de memória do computador (um endereço de memória) que armazenará os dados de acordo com o tipo de dado definido em sua declaração. O nome da variável é o identificador associado a essa área de memória (endereço), que será utilizada pelo algoritmo para guardar valores empregados no seu processamento. O valor de uma variável pode ser modificado ao longo da execução do algoritmo. Toda variável é definida no bloco de declarações.

Para a declaração de uma variável, utilizamos as regras dos identificadores. Observe, a seguir, o exemplo de criação (declaração) da variável denominada idade:

```
idade : inteiro // declaração da variável idade
```

Regra de Sintaxe Geral

```
<identificador>: <tipo de dado>
```

Outro detalhe importante é que podem ser criadas mais de uma variável na mesma instrução (separadas por vírgula). Esta forma de declaração múltipla se aplica apenas para um mesmo tipo de variável.

```
// declaração ou criação de 2 variáveis do tipo real na mesma instrução
salario , mensalidade : real
```

1.3.5 Constantes

Uma constante consiste no recurso ou objeto computacional que pode ser criado pelo programador para armazenar um único dado de um tipo definido. Uma constante é um espaço reservado na área de memória do computador (um endereço de memória) que armazenará os dados de acordo com o tipo de dado definido em sua declaração. O valor guardado em uma constante não sofre alteração ao longo do algoritmo. Seu valor só poderá ser atribuído uma única vez no início do bloco de instruções do algoritmo, não podendo ser mais alterado durante toda execução.

As constantes empregam os mesmos tipos de dados utilizados para as variáveis.

- Numérica: representada por valores reais ou inteiros, em que o ponto '.' separa a parte inteira da decimal, por exemplo: -0.59; 2.0; 0; -34; 597; 10.
- Lógica: representada pelas palavras VERDADEIRO ou FALSO como verdadeiro ou falso da lógica convencional, sendo denominadas ainda como constantes lógicas ou booleanas.

- Literal (caractere): formada por um caractere ou uma sequência de caracteres aceitos na simbologia da linguagem (símbolos especiais, letras e números). Essas constantes são representadas pelos caracteres correspondentes e válidos, sempre envolvidos por aspas (“ “).

A declaração de uma constante é semelhante ao de uma variável.

Regra de Sintaxe Geral

```
<identificador> : <tipo de dado> // CONSTANTE

// declaração da constante
equipe : inteiro // CONSTANTE

// único valor inicialmente atribuído a Constante
inicio
equipe  <-  25 // CONSTANTE
```

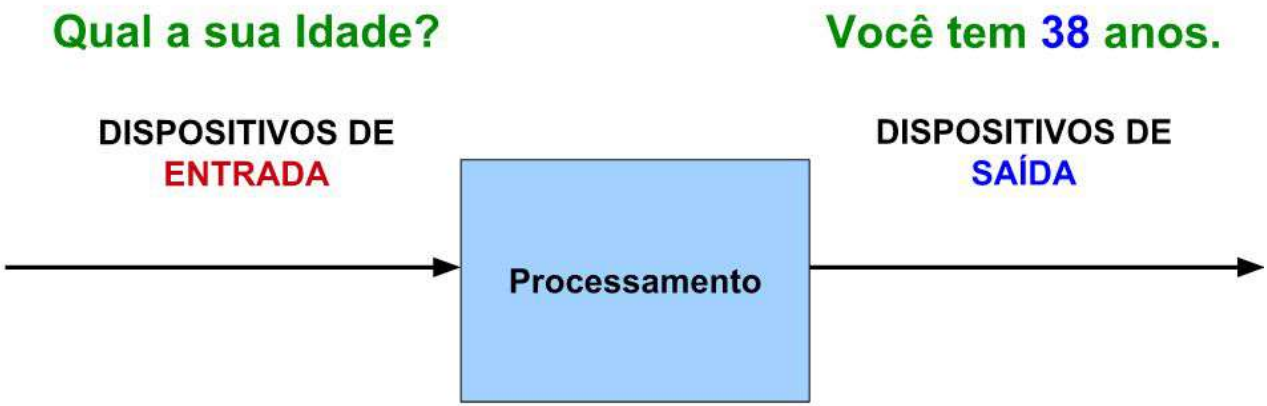
A constante equipe é criada como uma variável, porém, recebe a indicação de que é uma constante em seu algoritmo (é uma convenção adotada). Isso acontece por meio da descrição // CONSTANTE para indicar que esse recurso computacional é uma constante e que só receberá um único valor no início do algoritmo. Isso é uma convenção que indicará ao programador que o valor da constante não deverá mudar durante toda sua execução, pois o VISUALg que vamos utilizar não implementa constantes. Assim, esse recurso computacional se mantém constante durante todo processo de execução do seu algoritmo.

1.3.6 Instruções de Entrada e Saída de Dados

Como nosso algoritmo pode se comunicar com o usuário? Como podemos receber e exibir dados? Para essa finalidade, empregaremos as instruções de entrada e saída.

Observe a situação descrita pela Figura 8. Um programa de computador normalmente depende de dados de entrada para realizar um processamento e depois exibir isso ao usuário de alguma forma por meio de uma saída de dados. Se ele não tiver como exibir seus resultados, será inútil. Todas as linguagens de programação têm comandos para esse fim: receber e exibir resultados.

Figura 8 - Entrada, processamento e saída.



Fonte: Elaborada pelo autor

Instrução Leia

Para receber os dados, por meio do teclado (um dispositivo de entrada), vamos utilizar a instrução “leia”. Ela recebe os dados do usuário e armazena em uma variável (em um endereço de memória) para que possamos depois utilizar para processamentos. No exemplo abaixo, o algoritmo declara duas variáveis e depois usa um comando “leia” para receber do usuário os dois valores. Observe que a leitura dos dados acontece somente depois das duas variáveis serem declaradas.

Exemplo: leia (temperatura)

Regra de Sintaxe Geral
leia (<identificador da variável>)

```
algoritmo "Leia"

var
temperatura: real
dia: inteiro

inicio
leia (temperatura , dia)

fimalgoritmo
```

Instrução Escreva

Mas como exibir um resultado recebido ou mostrar um cálculo realizado (mostrar uma saída depois de um processamento)? Simples! Vamos empregar a instrução "escreva". Essa instrução exibe na tela do computador uma mensagem definida pelo USUÁRIO ou o valor armazenado em uma variável.

Exemplo: escreva (temperatura)

Regra de Sintaxe Geral
escreva (<mensagem ao usuário> ou <identificador da variável>)

Exemplos:
escreva ("Temperatura:" , temperatura) // mensagem e variável escreva (temperatura) // somente uma variável

A instrução escreva por exibir uma mensagem ao usuário concatenada (agrupada) com o valor da variável e, para isso, utilizamos a mensagem entre " " seguida por uma vírgula e o identificador da variável.

Observe alguns exemplos válidos dessa concatenação:

escreva (temperatura , “ graus no dia “ , dia) // variáveis intercaladas com mensagem
escreva (“Temperatura:” , temperatura , “ - Dia: “ , dia) // misturadas

É importante lembrar que sempre que utilizamos uma variável em um algoritmo, ela precisa inicialmente ser declarada, ou, as instruções leia e escreva vão gerar um erro de execução.

Vamos refletir sobre o exemplo abaixo carregado e executado no VISUALg.

```
algoritmo "Leia"

var
//Declaracao de variaveis
temperatura: real
dia: inteiro

inicio
// Recebe o valor da temperatura
escreva ("Informe a temperatura (graus):")
leia (temperatura)

// Recebe o dia
escreva ("Informe o dia (número):")
leia (dia)

// Exibe para o usuario a frase com os valores das variaveis
// "dia"  e "temperatura"
escreva ("No dia ", dia , " temos a temperatura em " , temperatura, " graus!")

fimalgoritmo
```

Escopo	Nome	Tipo	Valor
GLOBAL	TEMPERATURA	R	20
GLOBAL	DIA	I	11

```
Inicio da execução
Informe a temperatura (graus):20
Informe o dia (número):11
No dia  11 temos a temperatura em  20 graus!
Fim da execução.
```

Observe que temos três quadros na imagem anterior. O primeiro contém o código do algoritmo. Os dois abaixo indicam as variáveis e seus valores durante a execução do código e a tela de execução do algoritmo. Vamos entender a execução do algoritmo. O algoritmo começa com a declaração das variáveis temperatura e dia.

(1)A primeira instrução escreva orienta a ação do usuário na operação do algoritmo, por meio da exibição de uma mensagem fixa (texto constante - “Informe a temperatura (graus):”), sempre descrita entre aspas (“ “).

(2)A primeira instrução leia recebe do usuário o valor da temperatura digitado que será o lido e armazenado na memória do computador por meio da variável temperatura. Para a temperatura, o usuário digitou o valor 20.

(3)A segunda instrução escreva orienta a ação do usuário na operação do algoritmo, por meio da exibição de uma mensagem fixa (texto constante - “Informe o dia (numero):”), sempre descrita entre aspas (“ “).

(4)A segunda instrução leia recebe do usuário o valor do dia digitado que será o lido e armazenado na memória do computador por meio da variável dia. Para o dia, o usuário digitou o valor 11.

(5)A terceira instrução escreva devolve uma mensagem para o usuário concatenada com as duas variáveis dia e temperatura, por meio da exibição de uma mensagem fixa entre aspas (texto constante) intercalado com as variáveis por meio da vírgula. Como resultado final da execução temos:

No dia 11 temos a temperatura em 20 graus!
Fim da execução.

1.3.7 Operadores

Para que um algoritmo possa indicar ao computador como processar dados, é necessário utilizar operadores para os cálculos, relações e decisões.

Operadores Aritméticos

As operações aritméticas mais comuns são representadas pelos símbolos (ou caracteres) relacionados na tabela a seguir.

OPERAÇÃO	SÍMBOLO
ADIÇÃO	**+**
SUBTRAÇÃO	**-**
MULTIPLICAÇÃO	*
DIVISÃO	**/**
RAIZ QUADRADA	**raizq(**<valor>**)**
EXPONENCIAÇÃO	**^**
DIVISÃO INTEIRA (quociente)	**Div** ou ****
RESTO DA DIVISÃO INTEIRA	**Mod** ou **%**

Vejamos alguns exemplos:

```
2 + 3 // realiza a operação de adição entre 2 e 3 resultando no valor 5
valor * numero // efetua a multiplicação entre duas variáveis (valor e número)

raizq(4) // calcula a raiz quadrada do número 4 que teria resultado 2

5 ^ 2 // eleva 5 ao quadrado
```

Operadores Relacionais

A utilização dos operadores relacionais é muito importante ao processamento realizado por um computador. Os operadores relacionais realizam a comparação entre dois valores ou duas expressões (entradas) e resultam em valores lógicos VERDADEIRO (V) ou FALSO (F) (saídas). É preciso observar a tabela verdade. Os símbolos (ou caracteres) empregados na indicação dessas operações são apresentados na tabela a seguir:

OPERAÇÃO RELACIONAL	SÍMBOLO
IGUALDADE	=
DIFERENÇA	<>
MAIOR	>
MAIOR ou IGUAL	>=
MENOR	<
MENOR ou IGUAL	<=

Exemplos:

5 < 7 // compara o valor 5 verificando se este é menor que 7 e resulta em VERDADEIRO

idade = 1 // verifica se a variável idade possui o valor igual a 1 e se for, resulta em VERDADEIRO, caso contrário, resulta em FALSO.

4 <> -4 // verifica se 4 é diferente de -4 (quatro negativo). O resultado é FALSO.

Operadores Lógicos

Os operadores lógicos são empregados para avaliar expressões e também resultam em valores lógicos VERDADEIRO (V) ou FALSO (F).

OPERAÇÃO LÓGICA	SÍMBOLO
CONJUNÇÃO (e lógico)	**E**
DISJUNÇÃO (ou lógico)	**OU**
NEGAÇÃO (não lógico)	**NÃO**

Exemplos:

// conforme a lógica matemática, falso e falso resulta em falso
falso E falso

// conforme a matemática, a negação de falso é verdadeiro
NAO falso

// expressão com operadores relacionais e lógicos
((valor = ano) ou (5 < 7))

Vamos refletir?

Sempre que utilizarmos os operadores lógicos ou relacionais será necessário relembrar a tabela verdade. Ela apresenta os resultados da aplicação dos operadores lógicos (saídas) conforme os valores dos operadores envolvidos (entradas).

Tabela Verdade

Entrada (A)	Entrada (B)	(A) e (B)	(A) ou (B)	NÃO (A)	NÃO (B)
V	V	V	V	F	F
V	F	F	V	F	V
F	V	F	V	V	F
F	F	F	F	V	V

Os parênteses em uma expressão definem os blocos (subexpressões) e indicam o que será executado primeiro da esquerda para a direita. Além dos parênteses, existe uma ordem de prioridades estabelecida entre os operadores utilizados na expressão.

OPERADOR ARITMÉTICO	PRIORIDADE
Exponenciação	3 (maior)
Multiplicação	2
Divisão	2
Adição	1
Subtração	1 (menor)

OPERADOR LÓGICO	PRIORIDADE
e	3
ou	2
nao	1

A tabela abaixo apresenta a ordem de prioridade entre as categorias de operadores.

OPERADOR	PRIORIDADE
Operadores aritméticos	3
Operadores relacionais	2
Operadores lógicos	1

Exemplos:

```
// modularização é a divisão de uma expressão em partes com o uso de parênteses
(2+5>4) e (3<>3)     //resulta FALSO, pois VERDADEIRO e FALSO resulta FALSO.
(2 + 2)/2       // resulta 2
2 + 2/2         // resulta 3
```

No VISUALg, interface de desenvolvimento de algoritmos que utilizaremos nesta disciplina, não possui relacionamento de categorias. É preciso utilizar parênteses para organizar os operadores. Veja o exemplo abaixo:

```
2*5>3 ou 5+1<2 e 2<7-2     // resulta em erro.
(2*5>3) ou (5+1<2) e (2<7-2)       // certo seria assim.
```

Operador de Atribuição

A operação de atribuição consiste na possibilidade de um recurso ou objeto computacional poder receber e armazenar, em uma área previamente reservada de memória, um valor condizente com seu tipo de dado definido no momento de sua criação (declaração). Esta notação (<-) simboliza o operador de atribuição.

OPERAÇÃO	SÍMBOLO
ATRIBUIÇÃO	←

Exemplos:

```
meta <-  10   // valor 10 atribuído a variável ou constante meta
taxa  <- 3,4 +  0,2   // resultado da expressão (3,6) atribuído a variável taxa
```

Vamos refletir?

Perceba que o Português Estruturado é uma forma de representação de algoritmos que possui regras simples e bastante flexíveis.

Ao começar a programar em alguma linguagem de programação qualquer, verá que as ações (ler e escrever) e os operadores (atribuição e operador aritmético "+") serão substituídos por seus respectivos códigos na linguagem, mas a estrutura de criação e desenvolvimento do algoritmo terá processo semelhante aos diversos exemplos didáticos que veremos nesta disciplina.

Saiba mais!

DevCast: Programação - Por onde começar?

Dar os primeiros passos em uma nova área é sempre um momento difícil. Quando essa área é a programação, devido à quantidade de opções, termos e conceitos, essa etapa se torna ainda mais complexa. Afinal, por onde começar na programação?

Acesse: https://www.devmedia.com.br/programacao-por-onde-comecar/37391

Neste tópico aprendemos os conceitos que envolvem a estrutura básica de um algoritmo, suas palavras reservas e como identificar e atribuir variáveis e constantes. Aprendemos a utilizar instruções de entrada e saída e os operadores que irão permitir a realização de cálculos e a interação com o usuário por meio do recebimento, processamento e exibição de resultados de um algoritmo. Aproveite para consultar o glossário se tiver alguma dúvida de algum termo. Agora vamos realizar as atividades propostas e acompanhar as ações presenciais da disciplina. Não esqueça de olhar os sites e vídeos indicados que irão ampliar os seus horizontes de conhecimentos sobre o tema da unidade.

Vamos rever?

Compreender algoritmos é fundamental para a formação de um programador, quer seja mobile, web ou desktop, afinal qualquer programa de computador é baseado em algoritmos. A elaboração de um algoritmo deve respeitar as normas estabelecidas em cada método ou técnica de representação de raciocínio. Em português estruturado, marcamos o início e o fim pelas palavras reservadas “algoritmo” e “fimalgoritmo”. Toda linguagem de programação possui palavras reservadas que fazem parte de sua sintaxe. A palavra fimalgoritmo, por exemplo, é uma palavra reservada. Os comentários são linhas iniciadas por duas barras (//) que são ignoradas pelo interpretador, ou seja, elas podem conter qualquer informação depois das duas barras até o final da linha que não serão executadas pelo programa. Para que um computador possa realizar um determinado processamento, normalmente, é necessário que ele tenha acesso a informações, ou seja, dados para realizar o processamento. Os dados, ou informações, em algoritmos estão delimitados em quatro tipos básicos de dados: inteiros, reais (com casas decimais), caracteres e dados lógicos. Identificador é o nome fornecido a um recurso computacional que o identifica distintamente para acesso e manipulação do computador na execução ou realização de um algoritmo. A variável é o recurso ou objeto computacional de armazenamento de dados sujeito à variação em seus valores. Uma variável é um espaço reservado na área de memória do computador (um endereço de memória) que armazenará os dados de acordo com o tipo de dado definido em sua declaração. O valor guardado em uma constante não sofre alteração ao longo do algoritmo. Para receber os dados, por meio do teclado (um dispositivo de entrada), vamos utilizar a instrução “leia”. Ela recebe os dados do usuário e os armazena em uma variável (em um endereço de memória). A instrução “escreva” exibe na tela do computador uma mensagem definida pelo USUÁRIO ou o valor armazenado em uma variável. Para que um algoritmo possa indicar ao computador como processar dados, é necessário utilizar operadores para os cálculos, as relações e as decisões.

Sites indicados

1) BLOG DO SMITH
https://andreysmith.wordpress.com/category/informatica/introducao-a-ciencia-da-computacao/

2) Noções básicas de algoritmo
https://www.devmedia.com.br/nocoes-basicas-de-algoritmo/26405

3) Lógica de programação: introdução a algoritmos e pseudocódigo
https://www.devmedia.com.br/logica-de-programacao-introducao-a-algoritmos-e-pseudocodigo/37918
xogramas-diagrama-de-blocos-e-de-chapin-no-desenvolvimento-de-algoritmos/28550

Audiovisuais indicados

1) Programação: O que é uma “variável”?
https://www.youtube.com/watch?v=ubI8qb_F12o

2) Programação: O que é Algoritmo?
https://www.youtube.com/watch?v=68ZbfArHKw8

Questões de autoaprendizagem

1) A partir do que estudamos neste tópico elabore um algoritmo “Olá mundo” em VISUALg. Como ele seria? Tente fazer sozinho primeiro e, somente depois, clique no *link* da resposta.

2) O que é um algoritmo? Para que ele serve?

3) Observe o texto a seguir: “O aprendizado de qualquer nova ciência se inicia com o domínio de conceitos fundamentais. Para a programação, compreender o que é uma variável é essencial, pois é através delas que…”.

   O que é uma variável e qual a sua importância para um algoritmo (programa)? De que forma um computador armazena uma variável?

4) Algoritmo da soma de dois números. A partir da descrição das linhas abaixo, implemente em VISUALg o algoritmo que realiza a soma de dois números. Tente fazer sozinho primeiro e, somente depois, clique no link da resposta.

Questões de autoaprendizagem

Linha 01: Indicação do início do algoritmo em Portugol e seu nome;

Linha 02: Região de definição de variáveis.

Linha 03: Declaração das variáveis inteiras “a” e “b” que são utilizadas para guardar os valores inseridos pelo usuário, enquanto o “resultado”, como o nome sugere, irá receber a soma deles;

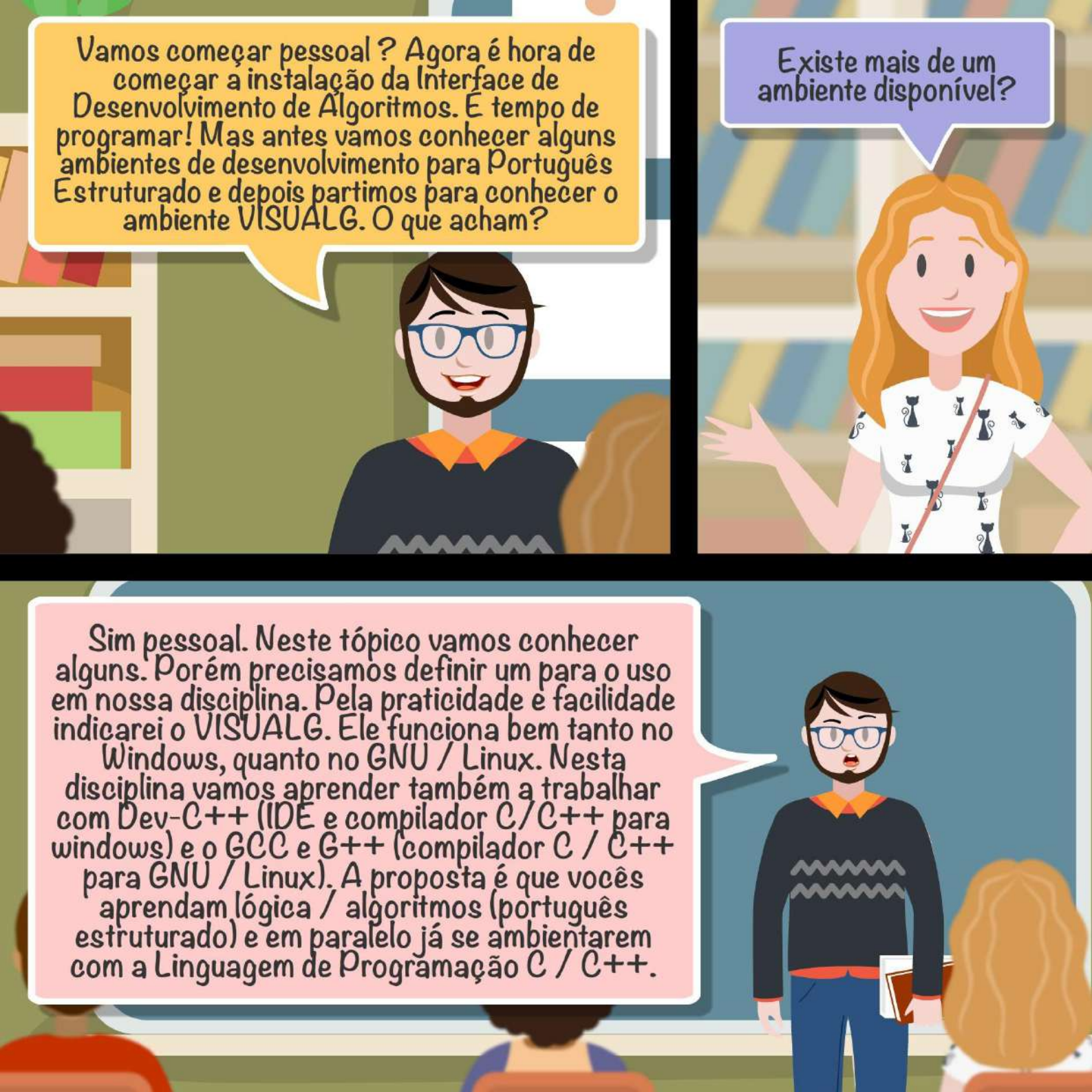
Linha 04: Início do bloco de código (comandos)

Linhas 05 e 06: Lê os valores inseridos pelo usuário e os guarda em “a” e “b”. Note que, no Portugol, as ações são sempre definidas por verbos no infinitivo. Ler serve para indicar a leitura de informações do usuário (entrada de dados), e Escrever, para mostrar o resultado final para o mesmo (saída de dados);

Linha 07: Utilizamos o operador de atribuição <-. Esse operador atribui a soma de “a” e “b” à variável “resultado”. Essa é a fase de processamento dos dados de entrada;

Linha 08: Para finalizar o algoritmo, temos a amostragem dos dados para o usuário. Nesse caso, utilizamos o verbo Escrever para mostrar essa ação (exibir na tela o resultado);

Linha 09: Indicação do fim do algoritmo em Portugol.


Vamos começar pessoal ? Agora é hora de começar a instalação da Interface de Desenvolvimento de Algoritmos. É tempo de programar! Mas antes vamos conhecer alguns ambientes de desenvolvimento para Português Estruturado e depois partimos para conhecer o ambiente VISUALG. O que acham?
Existe mais de um ambiente disponível?
Sim pessoal. Neste tópico vamos conhecer alguns. Porém precisamos definir um para o uso em nossa disciplina. Pela praticidade e facilidade indicarei o VISUALG. Ele funciona bem tanto no Windows, quanto no GNU / Linux. Nesta disciplina vamos aprender também a trabalhar com Dev-C++ (IDE e compilador C/C++ para windows) e o GCC e G++ (compilador C / C++ para GNU / Linux). A proposta é que vocês aprendam lógica / algoritmos (português estruturado) e em paralelo já se ambientarem com a Linguagem de Programação C / C++.

1.4 Conceitos Fundamentais de Programação

1.4.1 Ambiente de Desenvolvimento de Algoritmo

O que é um Integrated Development Environment (IDE)? Em português, falamos Ambiente de Desenvolvimento Integrado e trata-se de um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo. Ou seja, é um conjunto de ferramentas computacionais usadas no desenvolvimento de programas ou aplicações. Esses ambientes, normalmente, contam com editor, compilador e outras ferramentas de apoio ao desenvolvimento na linguagem pretendida. Podem ser somente um sistema de arquivos, um editor de texto, um linkeditor e um compilador ou podem consistir em uma interface uniforme (coleção de ferramentas convenientemente integradas).

Um computador só compreende linguagem de máquina. Já o ser humano se expressa em linguagem natural (português, por exemplo). As linguagens de programação funcionam como interfaces entre o ser humano e o computador. As linguagens de alto nível (mais próximas da linguagem natural) facilitam a compreensão do ser humano para a realização do problema e podem ser interpretadas ou compiladas para linguagem de máquina permitindo que o computador as execute.

Figura 9 - Interpretação ou compilação de uma linguagem de programação

Fonte: Elaborada pelo autor.

Existe uma diferença fundamental nos dois processos e ela é fundamental para o processo de desenvolvimento de programas e algoritmos. Essa diferença tem um resultado relevante na alocação de recursos computacionais para a execução do programa ou algoritmo. Veja as principais diferenças no quadro a seguir:

Interpretador (VISUALg)	Compilador (C/C++)
• ler uma instrução no algoritmo. • depurar somente a instrução lida. • traduzir a instrução lida para linguagem de máquina. • executar a instrução traduzida (Não gera arquivo executável. Lê e executa linha a linha o programa). • seguir para próxima instrução na sequência do algoritmo. • manter o ciclo acima em execução até fim do algoritmo.	• ler uma instrução no algoritmo. • depurar somente a instrução lida. • traduzir a instrução lida para linguagem de máquina. • seguir para próxima instrução na sequência do algoritmo. • manter ciclo acima em execução até fim do algoritmo. • executar algoritmo totalmente traduzido e em linguagem de máquina (gera um arquivo executável).

Para a interpretação de um arquivo em linguagem de alto nível (como português estruturado) sempre será necessário ter instalado o interpretador no computador no qual vamos executar o código. Assim, para interpretar um algoritmo em português estruturado precisaremos sempre ter instalado o VISUALg. No caso da Linguagem de programação C, uma vez que tenhamos compilado o arquivo do código fonte não precisaremos mais do Dev-C++ para executar o programa. Utilizaremos dois IDEs para desenvolvimento: VISUALg e Dev-C++.

Inicialmente, vamos começar trabalhando com a “Programação Estruturada”. Essa aprendizagem é fundamental para o avanço e a compreensão dos demais paradigmas existentes nessa área (orientação aos objetos, ao funcional, ao lógico, ao paralelo, entre outros).

1.4.2 Características do Ambiente VISUALg 2.0

O ambiente VISUALg possui ferramentas de edição de texto, depurador de erros sobre o algoritmo fonte, os recursos de manipulação de arquivos e um simulador para acompanhamento detalhado da situação da memória a cada execução de instruções no algoritmo elaborado.

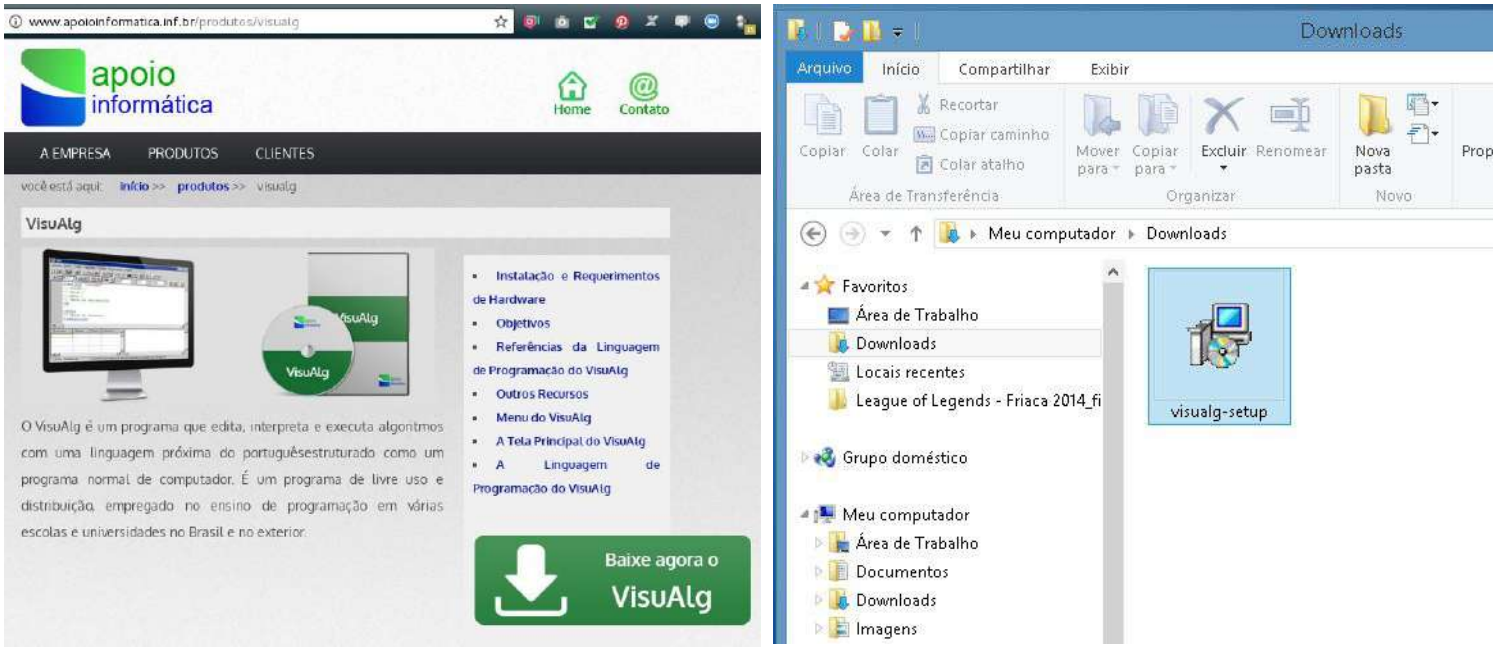
Este ambiente será responsável pela interpretação e execução do pseudocódigo (algoritmo). Ele fará a tradução de uma linguagem específica em um outro código em linguagem que o computador compreenda.

O VISUALg é um programa simples que não depende de DLLs, OCXs ou outros componentes e roda em qualquer Windows (desde o 95). Sua tela é simples e tem melhor aparência com resolução de vídeo de 800x600 ou maior. A instalação não copia arquivos para nenhuma outra pasta a não ser aquela em que for instalado e ocupa cerca de apenas 1 MB de espaço em disco.

1.4.3 Instalando VISUALg 2.0

Para iniciar o processo de instalação, precisamos obter o arquivo do instalador do VISUALg. Esse arquivo pode ser obtido no site Apoio Informática, na opção Produtos, ou diretamente no link: https://www.apoioinformatica.inf.br/produtos/visualg.

Figura 10 - Download e início da instalação

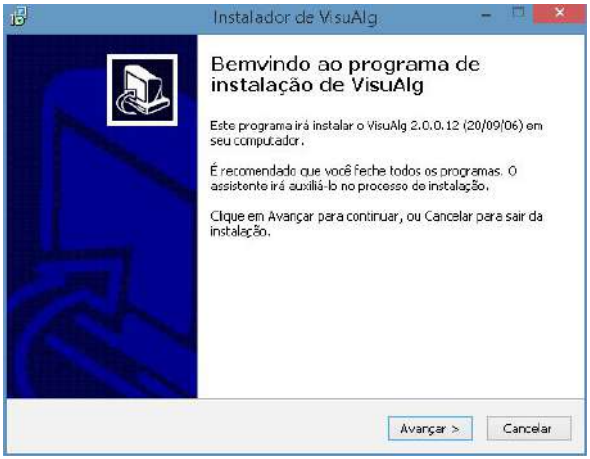


Fonte: Elaborada pelo autor.

Uma vez realizado o download do arquivo, conforme a Figura 11, vamos clicar duas vezes para executar o arquivo e iniciar sua instalação.

Uma vez carregado o instalador, obteremos a janela demonstrada na Figura 12.

Figura 11 - Instalador do VISUALg

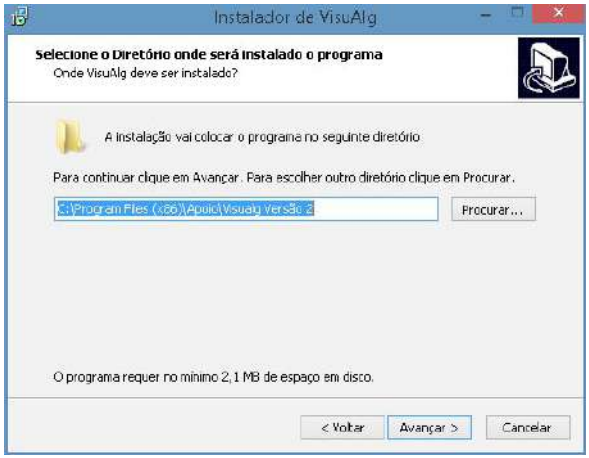


Clique em avançar para iniciar a instalação.

Fonte: Elaborada pelo autor.

Na próxima tela, conforme a Figura 12, vamos indicar o local de instalação do VISUALg. O instalador indicará o diretório padrão da instalação.

Figura 12 - Diretório de instalação

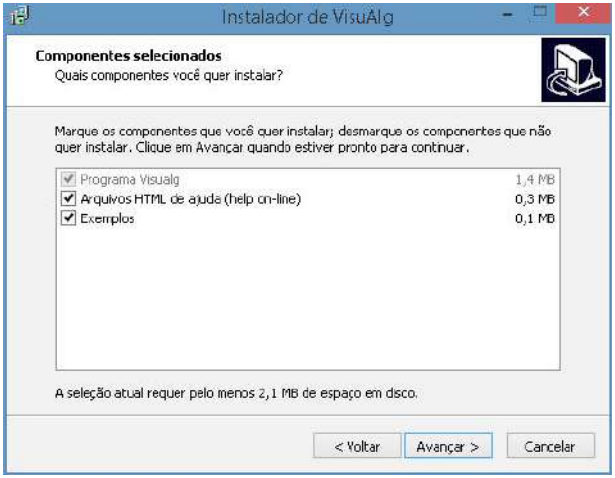


Clique em avançar para iniciar a instalação.

Fonte: Elaborada pelo autor.

Na próxima tela, conforme a Figura 13, vamos definir os componentes a serem instalados no sistema. Manteremos a configuração padrão, a qual instala os exemplos de códigos do VISUALg.

Figura 13 - Componentes da instalação

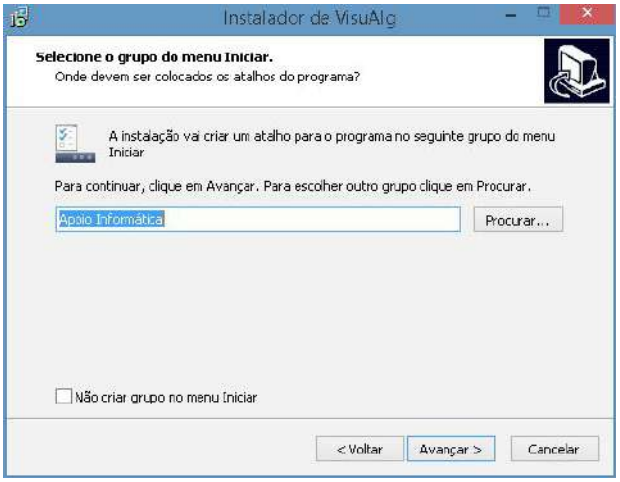


Clique em avançar para iniciar a instalação.

Fonte: Elaborada pelo autor.

Na próxima tela, conforme a Figura 14, vamos definir o grupo do menu iniciar para o atalho do programa VISUALg. Manteremos o padrão.

Figura 14 - Diretório de instalação

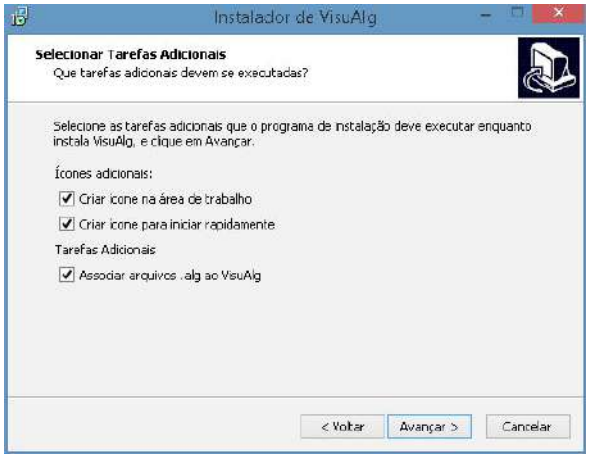


Clique em avançar para iniciar a instalação.

Fonte: Elaborada pelo autor.

Na próxima tela, conforme a Figura 15, vamos selecionar que tarefas adicionais serão executadas na instalação. Selecione “Associar arquivos .alg ao VISUALg”.

Figura 15 - Tarefas Adicionais

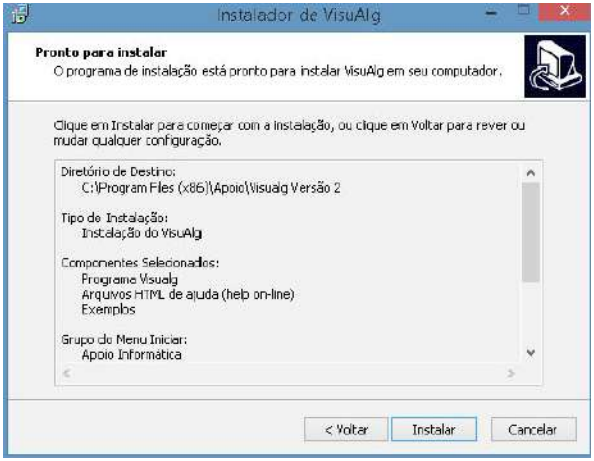


Clique em avançar para iniciar a instalação.

Fonte: Elaborada pelo autor.

Na próxima tela, conforme a Figura 16, o sistema indica que está pronto para instalar o VISUALg.

Figura 16 - Pronto para instalar

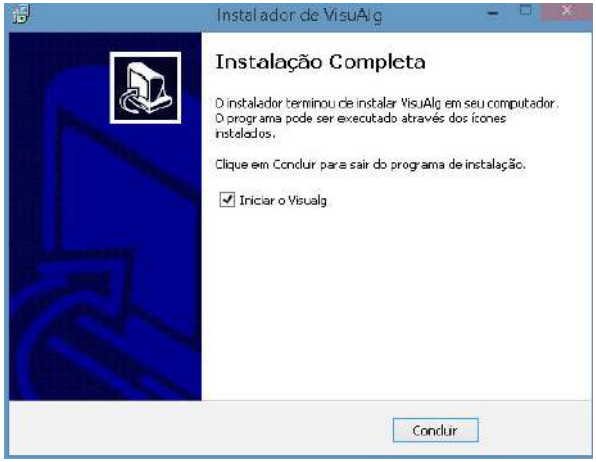


Clique em avançar para continuar a instalação.
A instalação será realizada e uma barra indicadora da evolução da instalação será exibida.

Fonte: Elaborada pelo autor.

Na próxima tela, conforme a Figura 17, o sistema indicará que terminou a instalação com sucesso.

Figura 17 - Instalação completa

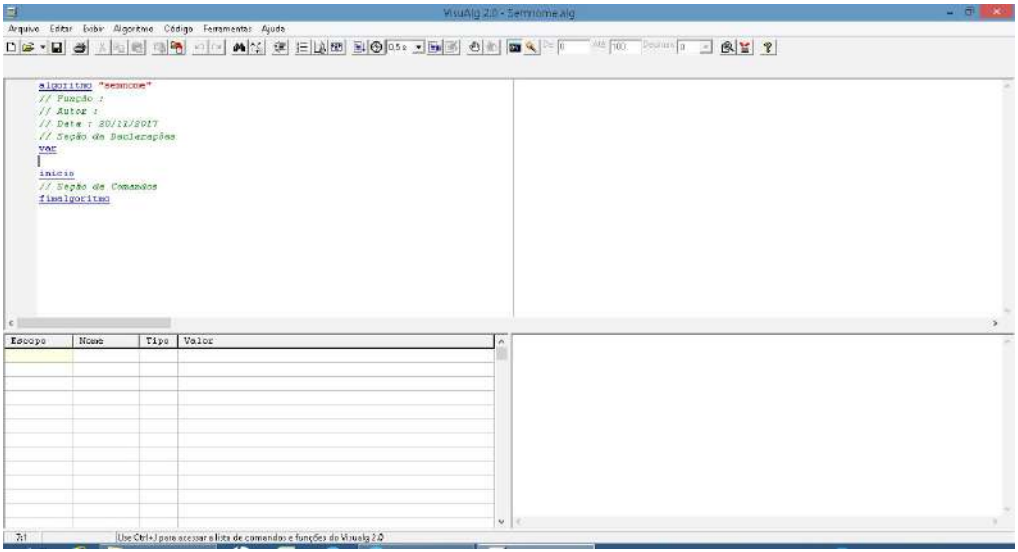


Você poderá manter o item "Iniciar o VISUALg" selecionado e ao clicar em CONCLUIR o sistema iniciará o VISUALg automaticamente.

Fonte: Elaborada pelo autor.

Considerando que tenha mantido o item "Iniciar o VISUALg" selecionado, na próxima tela, conforme a Figura 18, teremos a execução do VISUALg, pronto para ser utilizado.

Figura 18 - Execução do VISUALg

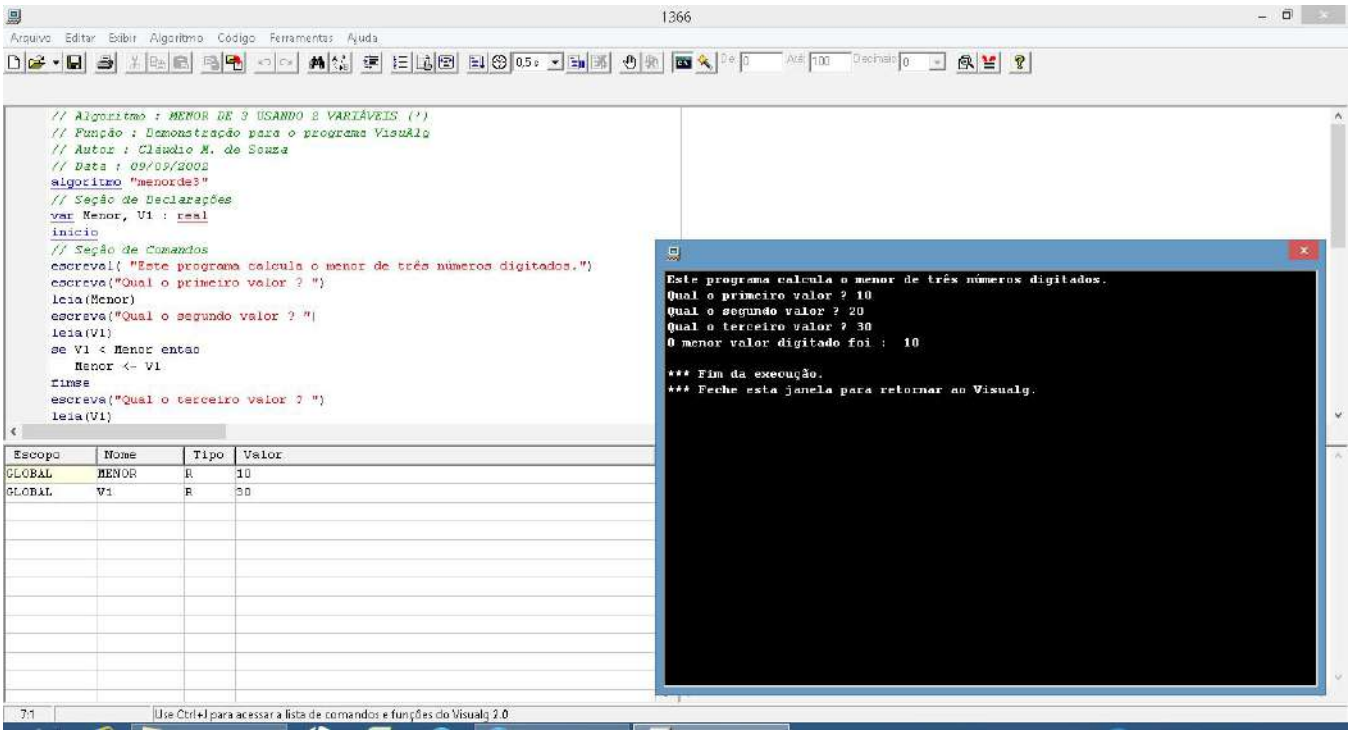


Fonte: Elaborada pelo autor.

Agora, vamos aprender a utilizar a interface do VISUALg e como criar, salvar, executar e carregar programas com ele.

Para testar o sistema, você pode clicar em ARQUIVO>>ABRIR>>EXEMPLOS e carregar um exemplo de algoritmo qualquer para testar seu VISUALg. Selecione o exemplo e depois pressione (F9) para executar o algoritmo. Se tudo funcionar corretamente você verá uma tela semelhante a Figura 19 que demonstra a execução de um algoritmo chamado “MENOR de 3”.

Figura 19 - Execução de um algoritmo

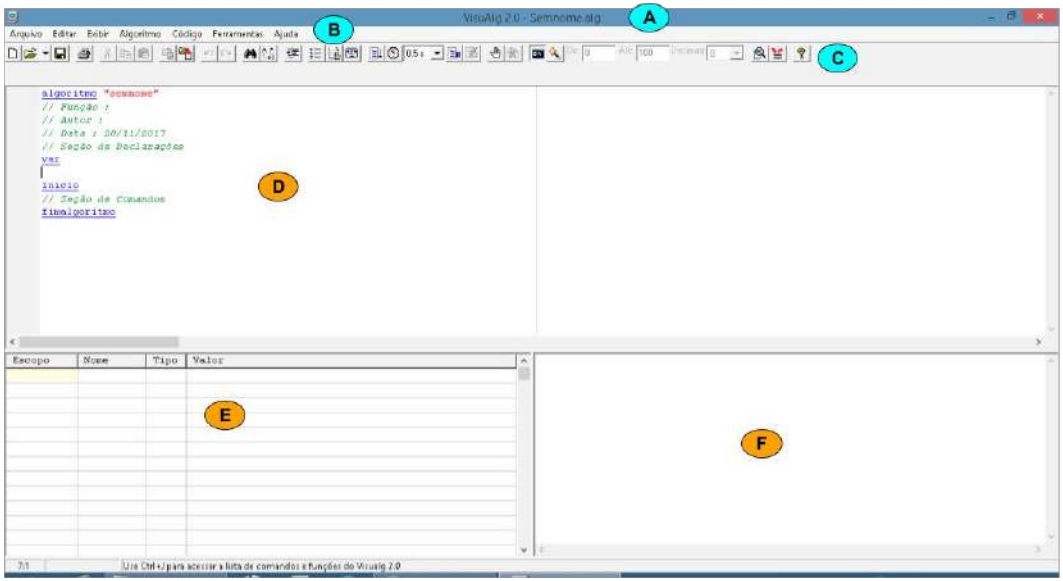


Fonte: Elaborada pelo autor.

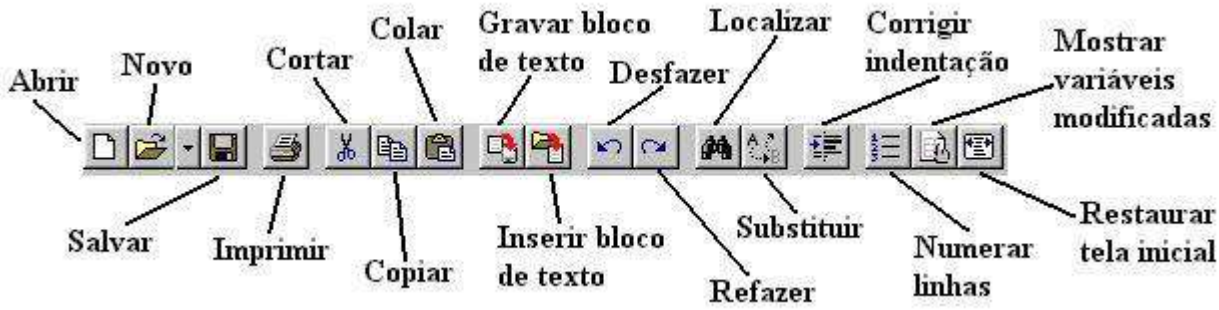
Agora vamos nos familiarizar um pouco com a tela do VISUALg. Uma vez que o programa tenha carregado, podemos identificar na sua tela, conforme a Figura 20 os seguintes itens:

(A)	Barra de título
(B)	Barra de menu (comandos)
(C)	Barra de botões (atalhos)
(D)	Editor de texto padrão
(E)	Área de acompanhamento de memória / variáveis
(F)	Quadro de execução (semelhante à janela de execução)

Figura 20 - Identificação de elementos da interface do VISUALg



Fonte: Elaborada pelo autor.



Abrir (Ctrl-A)	Abre um arquivo anteriormente gravado, substituindo o texto presente no editor.
Novo (Ctrl-N):	Cria um novo “esqueleto”, substituindo o texto presente no editor.
Salvar (Ctrl-S)	Grava o texto presente no editor; na primeira vez que um novo texto é gravado, é requisitado um nome e a localização para salvar o arquivo.
Imprimir(Ctrl+P)	Imprime na impressora padrão o texto presente no editor. Para configurar a impressão, use o comando Imprimir do menu.
Cortar (Ctrl-X)	Apaga texto selecionado, armazenando-o em uma área de transferência.
Copiar (Ctrl-C)	Copia o texto selecionado para a área de transferência.
Colar (Ctrl-V)	Copia o texto da área de transferência para o local em que está o cursor.
Gravar bloco de texto	Permite a gravação em arquivo de um texto selecionado no editor. A extensão sugerida para o nome do arquivo é .inc.
Inserir bloco de texto	Permite a inserção do conteúdo de um arquivo. A extensão sugerida para o nome do arquivo é .inc.

Salvar (Ctrl-S)	Desfaz último comando efetuado.
Refazer (Shift-Ctrl-Z)	Refaz último comando desfeito.
Localizar (Ctrl-L)	Localiza no texto presente no editor determinada palavra especificada.
Substituir (Ctrl-U)	Localiza no texto presente no editor determinada palavra especificada, substituindo-a por outra.
Corrigir Indentação (Ctrl-G)	Corrige automaticamente a indentação (formatação) do pseudocódigo, tabulando.
Numerar linhas	Ativa ou desativa a exibição dos números das linhas na área à esquerda do editor. A linha e a coluna do editor em que o cursor está em um determinado momento também são mostradas na barra de status.
Mostrar variáveis modificadas	Ativa ou desativa a exibição da variável que está sendo modificada.
Mostrar variáveis modificadas	Retorna a divisão da tela ao formato inicial, caso você tenha mudado o tamanho da área do editor de texto, quadro de variáveis ou simulador de saída.

Executar (F9)	Inicia (ou continua) a execução automática do pseudocódigo.
Executar com timer (Shift-F9)	Insere um atraso que pode ser especificado no intervalo ao lado antes da execução de cada linha.
Intervalo do timer	Atrasa cada linha quando deseja executar o pseudocódigo com timer.
Passo (F8)	Inicia ou continua a execução linha por linha do pseudocódigo, dando ao usuário a oportunidade de acompanhar o fluxo de execução, os valores das variáveis e a pilha de ativação dos subprogramas.
Parar (Ctrl-F2)	Termina a execução do pseudocódigo.
Liga/desliga breakpoint (F5)	Insere/remove um ponto de parada na linha em que esteja o cursor. Estes pontos de parada são úteis para a depuração e acompanhamento da execução dos pseudocódigos.
Desmarcar todos os breakpoints (Ctrl-F5)	Desativa todos os breakpoints que estejam ativados naquele momento.
Executar em modo DOS	Com esta opção ativada, tanto a entrada como a saída padrão passam a ser uma janela que imita o DOS, simulando a execução de um programa neste ambiente.
Gerar valores aleatórios	Ativa a geração de valores aleatórios que substituem a digitação de dados. A faixa padrão de valores gerados é de 0 a 100, mas pode ser modificada, basta alterar intervalo ao lado.
Intervalo dos valores aleatórios	Faixa de valores que serão gerados automaticamente quando esta opção estiver ativada.

Perfil (F7)	Após a execução de um pseudocódigo, exibe o número de vezes que cada uma das suas linhas foi executada. É útil para a análise de eficiência (por exemplo, nos métodos de ordenação).
Mostrar pilha de ativação (Ctrl-F3)	Exibe a pilha de subprogramas ativados num dado momento. Convém utilizar este comando em conjunto com breakpoints ou com a execução passo a passo.
Ajuda (F1)	Possibilita acesso às páginas de ajuda e às informações sobre o VISUALg.

1.4.4 Elaborando o Primeiro Algoritmo no VISUALg 2.0

Para começar a elaborar o seu primeiro algoritmo, vamos primeiro ABRIR o VISUALg. Depois de carregado, o VISUALg será mostrado conforme a Figura 21.

Agora, vamos escrever nosso primeiro algoritmo. Primeiro vamos fazer a síntese do algoritmo (colocar comentários). Digite o seguinte:

```
algoritmo "semnome"
// Função : Olá mundo
// Autor : Professor Ronald Costa
// Data : 20/11/2017
// Seção de Declarações
```

Depois vamos elaborar o corpo do algoritmo, com base no objetivo, na entrada e na saída. Vamos completar o resto do algoritmo. Esse primeiro algoritmo não realiza processamentos complexos. Salve o algoritmo em seu computador.

```
algoritmo "Olá Mundo"
// Função : Olá mundo
// Autor :  Professor Ronald Costa
// Data : 20/11/2017
// Seção de Declarações
var
inicio
// Seção de Comandos
    escreval("Alô Mundo!")
fimalgoritmo
```

Para executar algoritmos no VISUALg selecione a opção do menu superior Algoritmo e depois Executar (F9).

Figura 21 - Resultado do algoritmo

Fonte: Elaborada pelo autor.

Para consolidar o conhecimento e destreza no uso do VISUALg e sua interface vamos praticar com alguns exercícios simples de forma bem didática.

Faça um algoritmo que imprima o seu nome completo e o bairro onde você mora... O algoritmo terá uma saída simples, muito semelhante ao exemplo anterior.

Vamos colocar a mão no código!

```
1  algoritmo "boas vindas"
2  // Aluno: Tux
3  // Síntese
4  // Objetivo: realizar saudação de boas vindas para este programador
5  // Entrada: não tem
6  // Saída: mensagem de boas vindas com o nome do programador
7  // Declarações
8  var
9  inicio
10   Escreval ("Meu nome é RONALD COSTA")
11   Escreval ("Moro em Brasília na Asa Norte")
12 fimalgoritmo
```

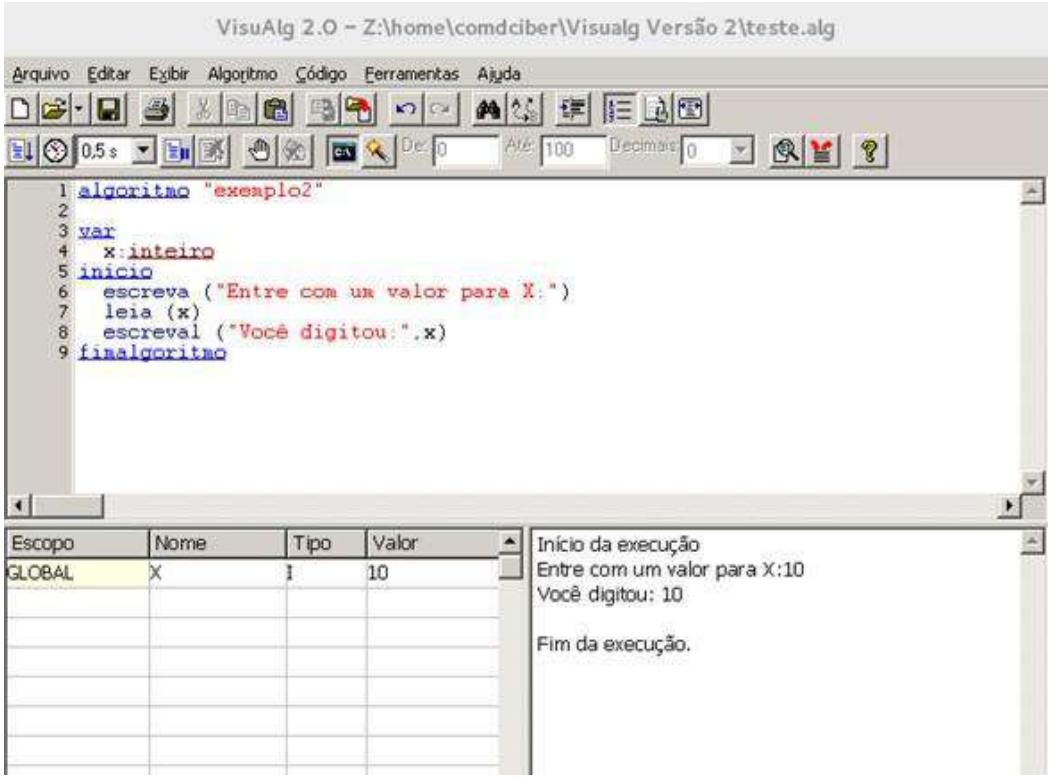
A saída após a execução deverá ser:

```
Meu nome é RONALD COSTA
Moro em Brasília na Asa Norte

*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.
```

Como fazemos para receber valores e imprimir os valores recebidos na tela? Vamos criar um algoritmo para receber um número inteiro e armazenar em uma variável. Depois faremos o algoritmo exibir esse número na tela.

Vamos colocar a mão no código!

Figura 22



Fonte: Elaborada pelo autor.

1.4.5 Recursos de Ajuda no VISUALg 2.0

O VISUALg possui integrado um sistema de ajuda. Podemos acessar um menu de ajuda de comandos pressionando a tecla CTRL + J conforme a Figura 23.

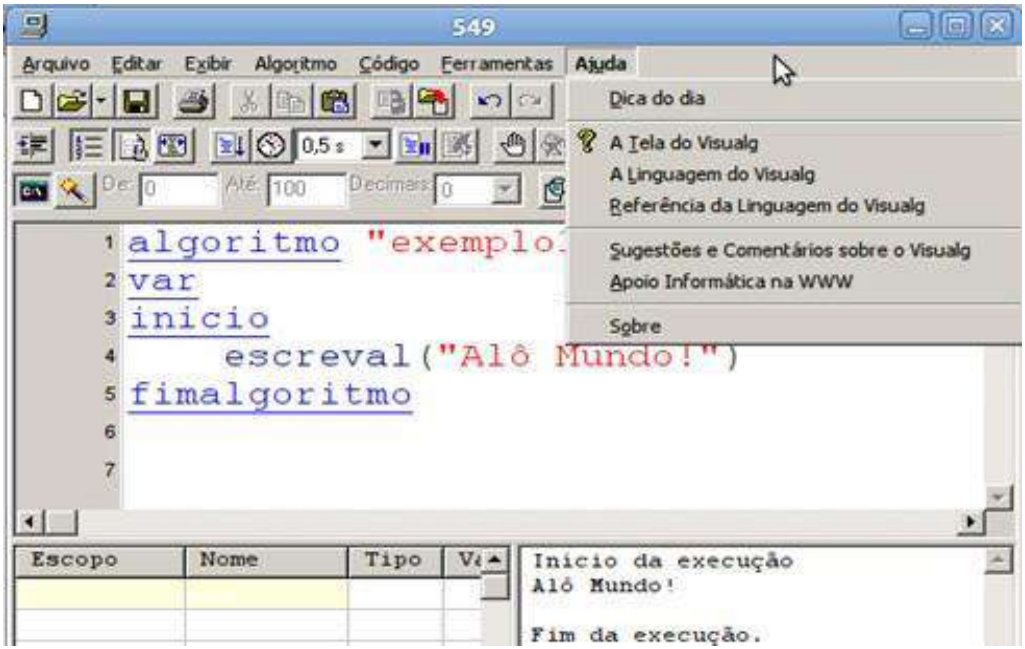
Figura 23 - Recursos de ajuda no VISUALg

Fonte: Elaborada pelo autor.

Além dessa ajuda rápida, podemos acessar o menu de AJUDA, conforme a Figura 24, no qual é possível observar as seguinte opções:

- Dica do dia
- ? A Tela do VISUALg
- A Linguagem do VISUALg
- Referência da Linguagem do VISUALg

Figura 24 - Recursos de ajuda no VISUALg



Fonte: Elaborada pelo autor.

Para ajudar no processo de aproximação ao Português Estruturado e também do VISUALg apresento um conjunto de documentações para consulta que poderão ser utilizados como material complementar:

a) Guia de Referência - VISUALg
http://www.inf.ufsc.br/~bosco.sobral/ensino/ine5201/Visualg2_manual.pdf
b) Software VISUALg 2.0 Software VISUALg 2.0 - Bruno Tonet - UCS
http://www.dainf.ct.utfpr.edu.br/~eteocles/visualg.pdf
c)Manual do VISUALg https://manual.visualg3.com.br/doku.php?id=manual

1.4.6 Realizando testes de mesa (método chinês)

O teste de mesa ou método chinês consiste na depuração manual, amplamente utilizada no meio acadêmico. Esse processo é especialmente útil para testar e avaliar um algoritmo, particularmente a erros de lógica. Essa técnica consiste no acompanhamento e registro do estado das variáveis no papel, durante a execução mental, linha a linha do algoritmo.

O objetivo de um teste de mesa é verificar se o código de um algoritmo leva ao resultado esperado, simulando seu funcionamento (execução) sem usar o computador, empregando apenas papel e caneta, assim como valores de teste.

É uma forma de estudar e compreender profundamente o algoritmo particularmente quando os alunos de TI estão tendo o primeiro contato com algoritmos e códigos. Pode e deve ser aplicado quando estamos realizando provas escritas em concursos/processos seletivos.

Para a realização de um teste de mesa, devemos observar os seguintes passos:

1)Observe as variáveis que está utilizando em seu algoritmo. Identifique-as.

2) Faça uma tabela, no papel, com linhas e colunas. A primeira coluna deverá conter o número das linhas da execução de instruções que estamos observando. As colunas seguintes da tabela deverão conter as variáveis que vamos observar. Uma coluna para cada variável.

3) Em cada linha da tabela vamos colocar o número da linha de código que estamos executando (primeira coluna). Nas colunas seguintes dessa mesma linha, os valores de cada variável no momento de execução do algoritmo.

4) Ao acompanharmos a execução, iremos registrando a linha (número na primeira coluna) e os valores de variáveis para aquele momento de execução nas outras colunas.

5) Quando for indicar que uma variável recebeu um valor lido, coloque entre parênteses. Ex: (18)

6) Se foi escrito pela instrução (alterado), coloque entre chaves. Ex: {20}

7) Quando uma variável, naquele momento de execução, ainda não tem valor definido, coloque uma interrogação. Ex: ?

8) Se o valor de uma variável for exibido na tela (saída), coloque na coluna o valor em negrito e vermelho para destacar.

Dessa forma, conseguiremos entender a execução do algoritmo. Vamos acompanhar a realização de um teste de mesa (chinesinho) de um algoritmo de exemplo.

Imagine um algoritmo para ler dois números (a e b) e apresentar o resultado das 4 operações básicas (+ , - , * e /).

Para construir o algoritmo pense no que precisamos:

a) Quais os resultados a fornecer? Exibir o valor da soma, da subtração, da multiplicação e da divisão.

b) Quais as entradas? O que preciso para obter a saída? Os dois números a e b.

c) Como transformar as entradas em saídas, ou seja, como realizar o processamento? Realizar as operações com a e b.

```
1  algoritmo "operações"
2
3  var
4  // Declaração de variáveis
5  a,b,soma,subtracao,produto,divisao : real
6
7  inicio
8  // Seção de Comandos
9  escreva ("Digite o valor de a:")
10 leia (a)
11 escreva ("Digite o valor de b:")
12 leia (b)
13
14 soma <- a + b
15 subtracao <- a - b
16 produto <- a * b
17 divisao <- a / b
18 escreval ("Soma:",soma)
19 escreval ("Subtracao:",subtracao)
20 escreval ("Produto:",produto)
21 escreval ("Divisao:",divisao)
22
23 fimalgoritmo
```

Inserindo na execução do algoritmo os valores 10 e 20 para "a" e "b", respectivamente, temos o seguinte resultado:

```
Digite o valor de a:10
Digite o valor de b:20
Soma: 30
Subtracao: -10
Produto: 200
Divisao: 0.5

*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.
```

E como testar isso no TESTE DE MESA?

De forma didática vamos colocar na coluna da esquerda da tabela, a seguir, a execução do algoritmo (linha a linha) e, na coluna da direita, a situação do teste de mesa de forma a entender o teste de mesa.

<table>
<tr><td>Execução mental do algoritmo</td><td>Inicia (ou continua) a execução automática do pseudocódigo.</td></tr>
<tr><td>algoritmo "operações"
var
// Declaração de variáveis</td><td>Não temos nada para o teste de mesa ainda...</td></tr>
<tr><td>a,b,soma,subtracao,produto,
divisao : real</td><td>LINHA 5) Todas as variáveis recebem "?" pois foram apenas declaradas (não tem valor ainda nesta linha 5)
<table>
<tr><th>Linha</th><th>a</th><th>b</th><th>soma</th><th>subtração</th><th>divisão</th><th>produto</th></tr>
<tr><td>5</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>inicio
// Seção de Comandos
escreva (“Digite o valor de a:”)</td><td>Não temos nada para o teste de mesa aqui nestas linhas...</td></tr>
<tr><td>leia (a)</td><td>LINHA 10) A variável “a” recebe o valor digitado pelo usuário que foi “10”. Colocamos (10) na coluna da variável “a”. As outras permanecem com “?”.

<table><tr><th>Linha</th><th>a</th><th>b</th><th>soma</th><th>subtração</th><th>divisão</th><th>produto</th></tr><tr><td>5</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td></tr><tr><td>10</td><td>(10)</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></td></tr>
<tr><td>escreva (“Digite o valor de b:”)</td><td>Não temos nada para o teste de mesa aqui na linha 11, pois não envolve nenhuma variável.</td></tr>
<tr><td>leia (b)</td><td>LINHA 12) A variável “b” recebe o valor digitado pelo usuário que foi “20”. Colocamos (20) na coluna da variável “b”. As outras permanecem com “?” e “a” permanece com seu valor inalterado

<table><tr><th>Linha</th><th>a</th><th>b</th><th>soma</th><th>subtração</th><th>divisão</th><th>produto</th></tr><tr><td>5</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td></tr><tr><td>10</td><td>(10)</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td></tr><tr><td>12</td><td>10</td><td>(20)</td><td>?</td><td>?</td><td>?</td><td>?</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></td></tr>
<tr><td>soma <- a + b</td><td>LINHA14)Avariável “soma” recebe o valor do processamento da soma que é “30”. Colocamos {30} na coluna da variável “soma”. As outras permanecem com seus valores anteriores (inalteradas).

<table><tr><th>Linha</th><th>a</th><th>b</th><th>soma</th><th>subtração</th><th>divisão</th><th>produto</th></tr><tr><td>5</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td></tr><tr><td>10</td><td>(10)</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td></tr><tr><td>12</td><td>10</td><td>(20)</td><td>?</td><td>?</td><td>?</td><td>?</td></tr><tr><td>14</td><td>10</td><td>20</td><td>{30}</td><td>?</td><td>?</td><td>?</td></tr></table></td></tr>
</table>

subtracao <- a - b

LINHA 15) A variável "subtracao" recebe o valor do processamento da subtração que é "-10". Colocamos {-10} na coluna da variável "subtracao". As outras permanecem com seus valores anteriores (inalteradas).

Linha	a	b	soma	subtração	divisão	produto
5	?	?	?	?	?	?
10	(10)	?	?	?	?	?
12	10	(20)	?	?	?	?
14	10	20	{30}	?	?	?
15	10	20	30	{-10}	?	?

produto <- a * b

LINHA 16) A variável "produto" recebe o valor do processamento do produto que é "200". Colocamos {200} na coluna da variável "produto". As outras permanecem com seus valores anteriores (inalteradas).

Linha	a	b	soma	subtração	divisão	produto
5	?	?	?	?	?	?
10	(10)	?	?	?	?	?
12	10	(20)	?	?	?	?
14	10	20	{30}	?	?	?
15	10	20	30	{-10}	?	?
16	10	20	30	-10	{200}	?

divisão <- a / b

LINHA 17) A variável "divisao" recebe o valor do processamento da divisão que é "0.5". Colocamos {0.5} na coluna da variável "divisao". As outras permanecem com seus valores anteriores (inalteradas).

Linha	a	b	soma	subtração	divisão	produto
5	?	?	?	?	?	?
10	(10)	?	?	?	?	?
12	10	(20)	?	?	?	?
14	10	20	{30}	?	?	?
15	10	20	30	{-10}	?	?
16	10	20	30	-10	{200}	?
17	10	20	30	-10	200	{0.5}

```
escreval ("Soma:",soma)
escreval ("Subtracao:",subtracao)
escreval ("Produto:",produto)
escreval ("Divisao:",divisao)
```

LINHAS 18,19,20 e 21) Vamos destacar os valores das variáveis envolvidas que estão sendo exibidas na tela.

Linha	a	b	soma	subtração	divisão	produto
17	10	20	30	-10	200	{0.5}
18	10	20	{30}	-10	200	0.5
19	10	20	30	{-10}	200	0.5
20	10	20	30	-10	{200}	0.5
21	10	20	30	-10	200	{0.5}

Tomando por base o processo de teste apresentado neste exemplo, você poderá criar outras variações do teste de mesa. Isso servirá até que você tenha maior firmeza para usar métodos mais simples de depuração e teste dos algoritmos.

Essa técnica é um excelente método de estudo e ajuda muito no entendimento da lógica de funcionamento dos algoritmos. A depuração manual é a base do aprendizado de algoritmos! Não esqueça!

1.4.7 Instalação do Dev-C++

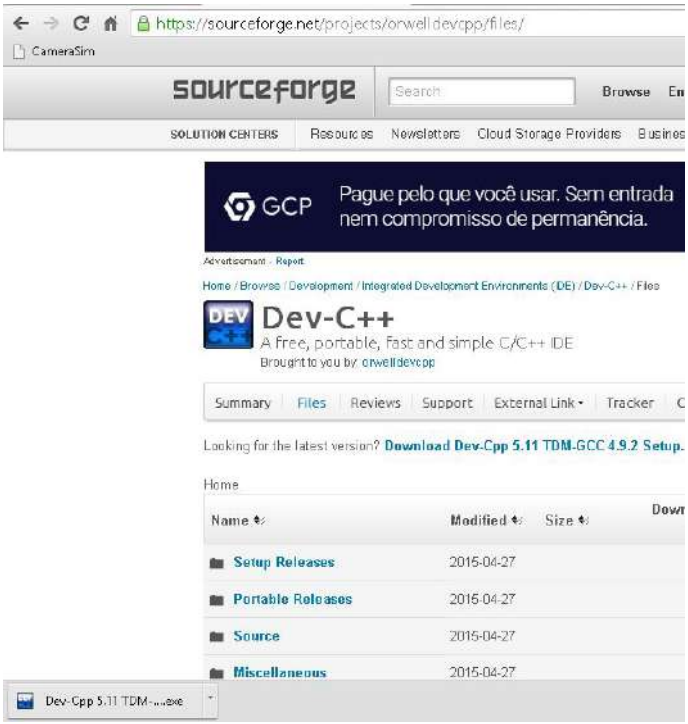
O Dev-C++ é um Ambiente de Desenvolvimento Integrado (IDE - Integrated Development Environment) para programação na linguagem C/C++, gratuito e de código aberto. Ele usa o GCC (GNU Compiler Collection) como seu compilador.

Pode ser obtido em: https://sourceforge.net/projects/orwelldevcpp/. Esta interface é a ferramenta mais utilizada nos cursos universitários para disciplinas de introdução à programação com linguagem C por ser leve e fácil de utilizar para desenvolvimento de programas tanto em linguagem C como em C++.

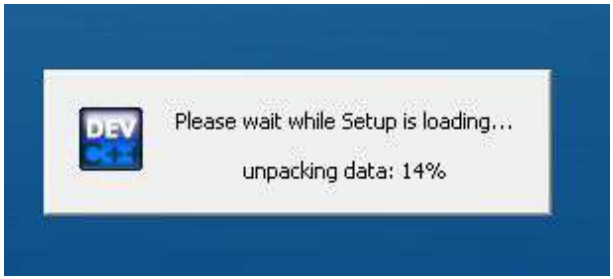
Vamos começar o roteiro de instalação da ferramenta.

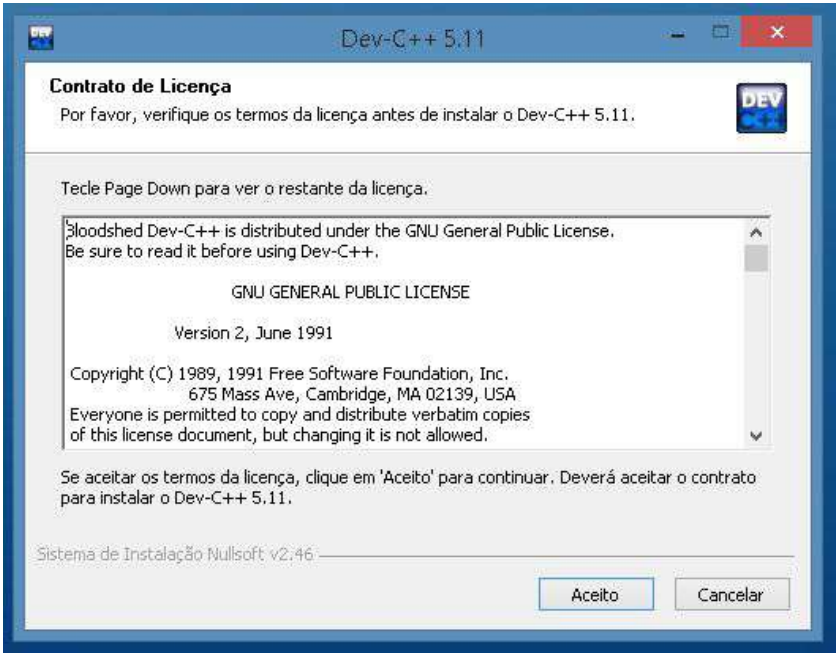
Observação: as figuras apresentadas nos itens de 1 a 14, correspondem ao print de telas extraídas durante a instalação do Dev-C++, realizada pelo autor.

1) A primeira etapa é providenciar o download do instalador do Dev-C++. Acesse o link, a seguir, para fazer o download do executável gratuito no endereço: https://sourceforge.net/projects/orwelldevcpp/

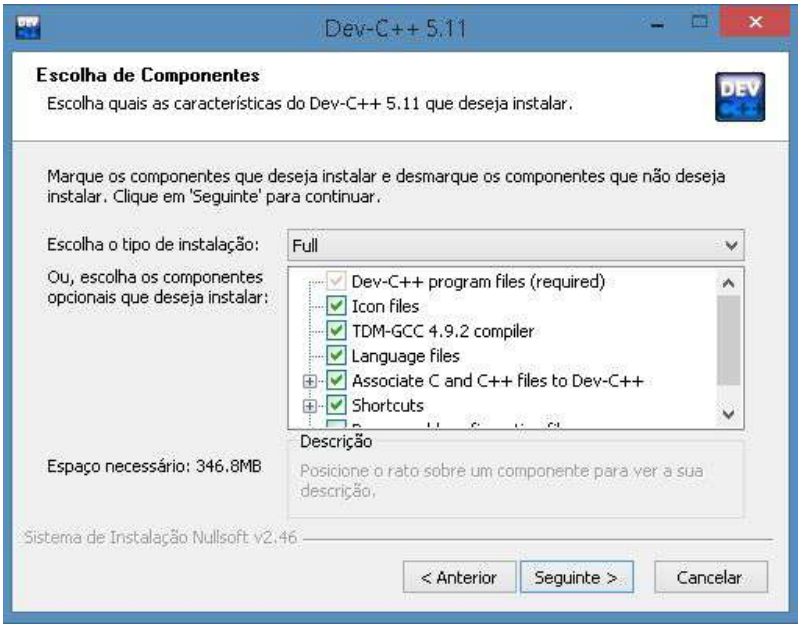


2) Faça o download para o seu computador. Acesse o local no qual fez o download e dê dois cliques para iniciar o processo de instalação. A instalação irá iniciar.

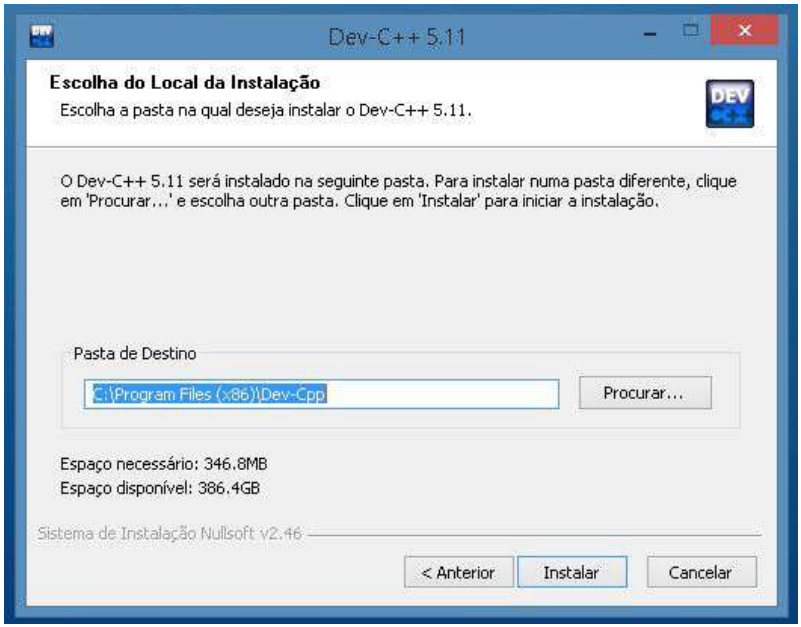


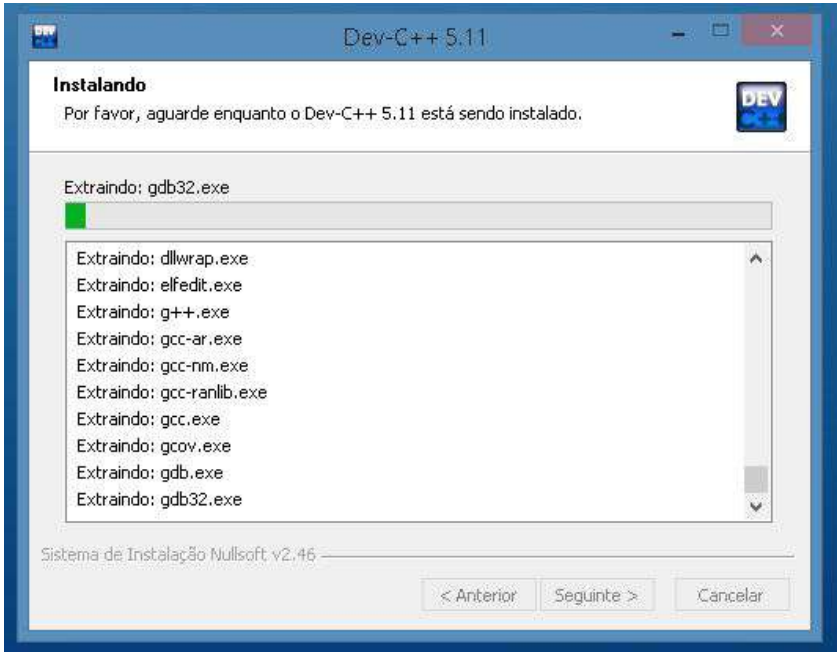
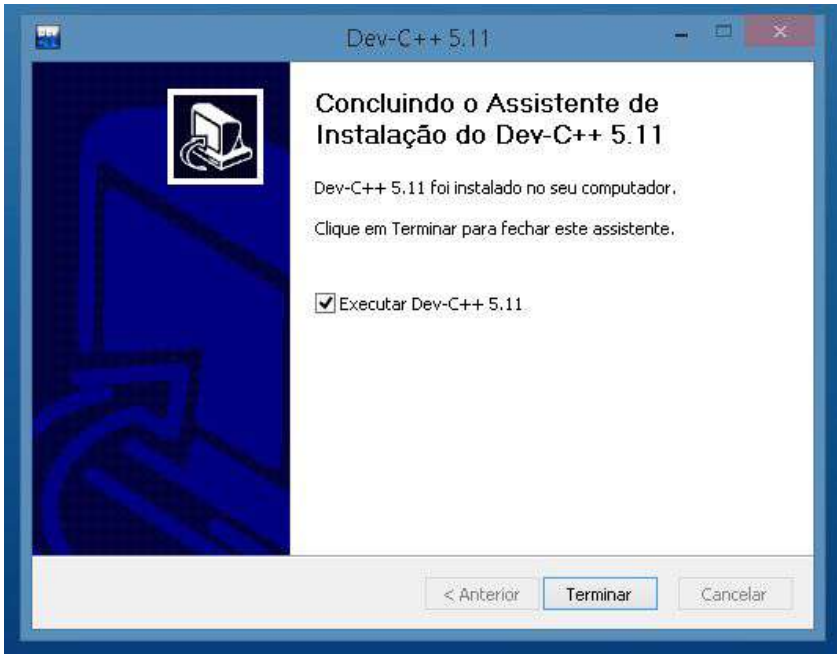
3) Selecione a linguagem “Português” e clique OK.	
4) Aceite o contrato de licença do Dev-C ++. Clique em “Aceito” para continuar a instalação.	

5) Mantenha a configuração padrão na tela de escolha dos componentes. Clique em “Seguinte” para continuar.

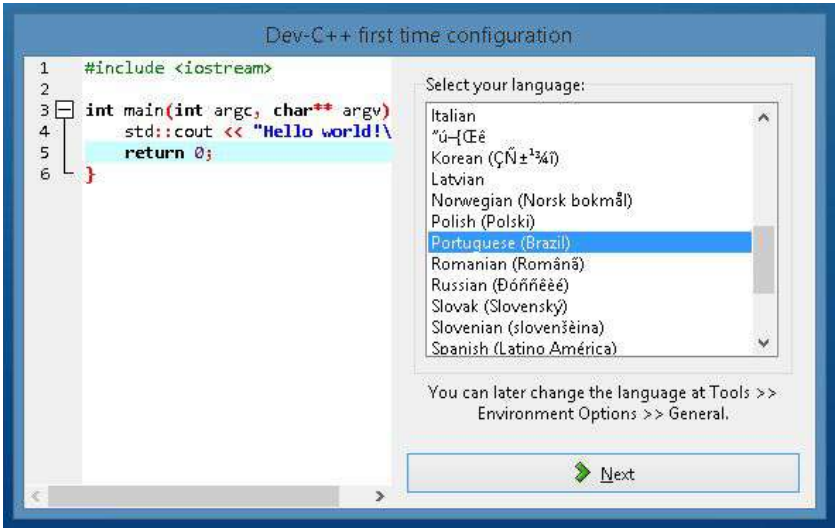


6) Escolha o local de instalação do DV-C++. Apenas modifique o local se realmente for necessário. Clique em “Instalar” para continuar.

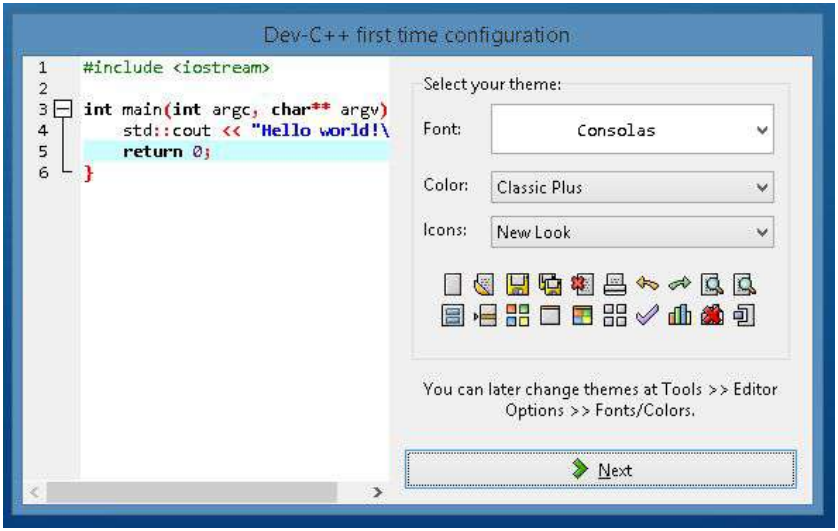


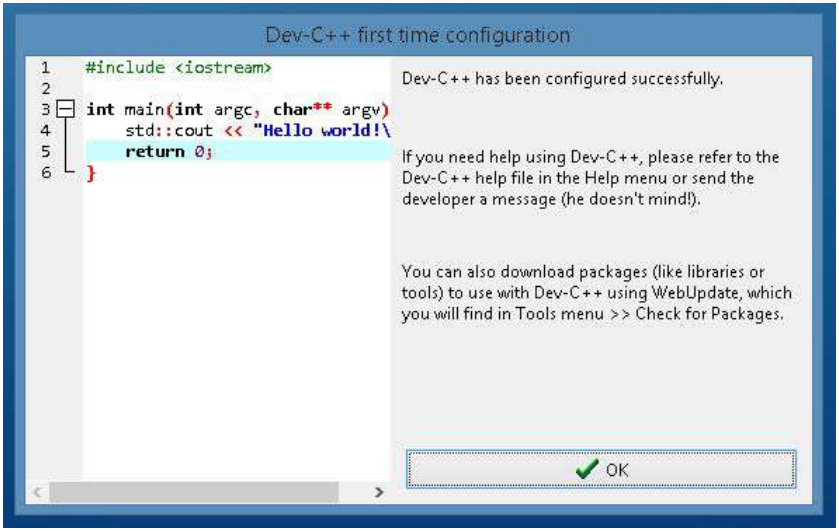
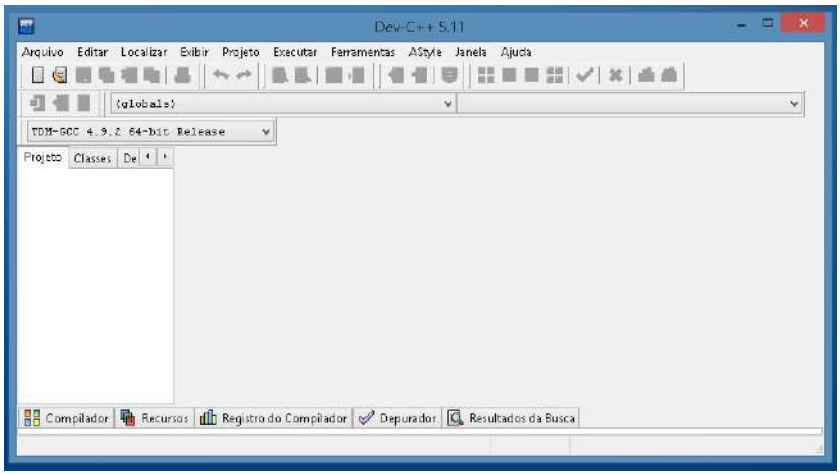
7) O sistema começará a instalação do Dev-C++. Aguarde o processo de instalação terminar.	
8) Pronto, a instalação foi realizada. Agora vamos clicar em “Terminar” para executar a primeira vez o Dev-C++.	

9) Apenas na primeira execução do Dev-C++ devemos definir a linguagem padrão da interface. Selecione “Portuguese (Brazil)” e depois clique em “Next” para continuar.



10) Manteremos a configuração padrão na tela de seleção do tema da interface. Altere apenas se desejar realmente. Clique em “Next” para continuar.

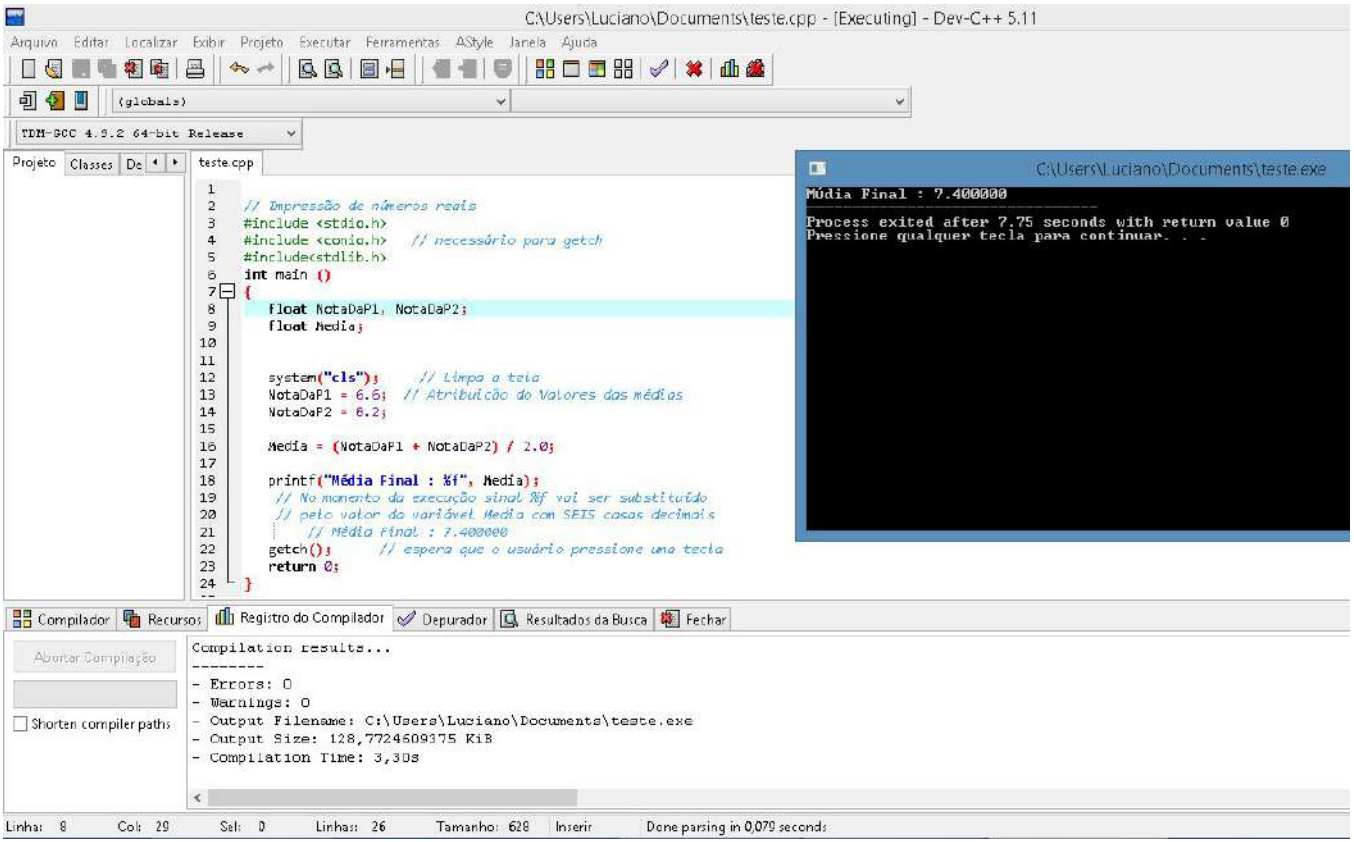


11) Pronto. O Dev-C++ foi instalado corretamente. Concorda que nem doeu? Brincadeira. Vamos clicar em "OK" para começar a programar!	
12) Essa é a interface do Dev-C++. Selecione o ícone NOVO e depois ARQUIVO FONTE (ou pelo atalho CTRL+N) para que possamos digitar um primeiro código.	

13) Digite o código abaixo na interface do Dev-C ++.

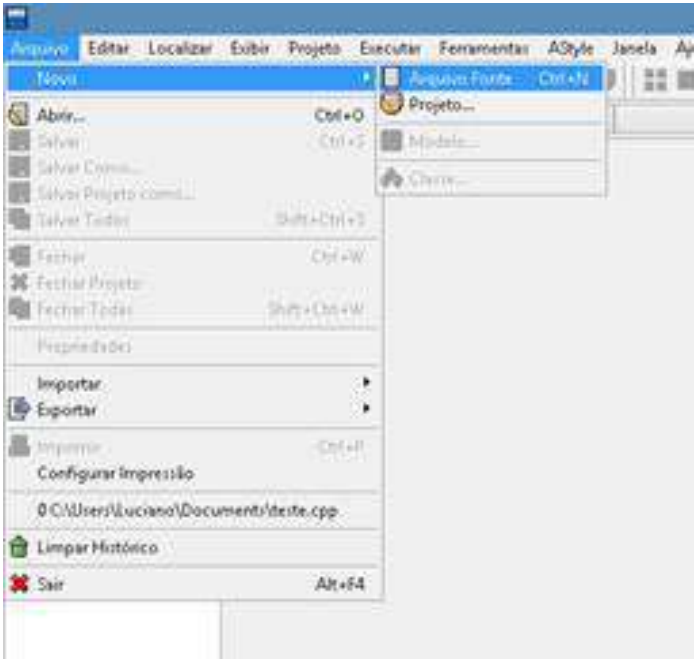
```
// Impressão de números reais
#include <stdio.h>
#include <conio.h>        // necessário para getch
#include<stdlib.h>
int main ()
{
  float NotaDaP1,
  NotaDaP2;  float Media;

  system("cls");   // Limpa a tela
  NotaDaP1 = 6.6; // Atribuição do Valores das médias
  NotaDaP2 = 8.2;
  Media = (NotaDaP1 + NotaDaP2) / 2.0;

  printf("Média Final : %f", Media);

 // No momento da execução sinal %f vai ser substituído
 // pelo valor da variável Media com SEIS casas decimais
       // Média Final : 7.400000
  getch();  // espera que o usuário pressione uma tecla
  return 0;
}
```

14) Agora vamos executar o código. Para executar vamos usar o atalho (F11). Salve o arquivo. Se você fez tudo certo verá o resultado abaixo. Perceba que C compila o arquivo do código fonte. Olhe na pasta na qual salvou o código e verá que depois da execução do código, o Dev-C ++ gerou (compilou) um arquivo executável (binário) a partir do seu código fonte. Essa é a diferença entre compilação e interpretação. Não esqueça.



Quando acessamos a interface do DEV-C++ o sistema apresenta por padrão a tela da Figura 25. Podemos, a partir do MENU ARQUIVO, começar um arquivo fonte diretamente ou um projeto. Se desejar começar um arquivo fonte rapidamente digite CTRL + N (atalho).

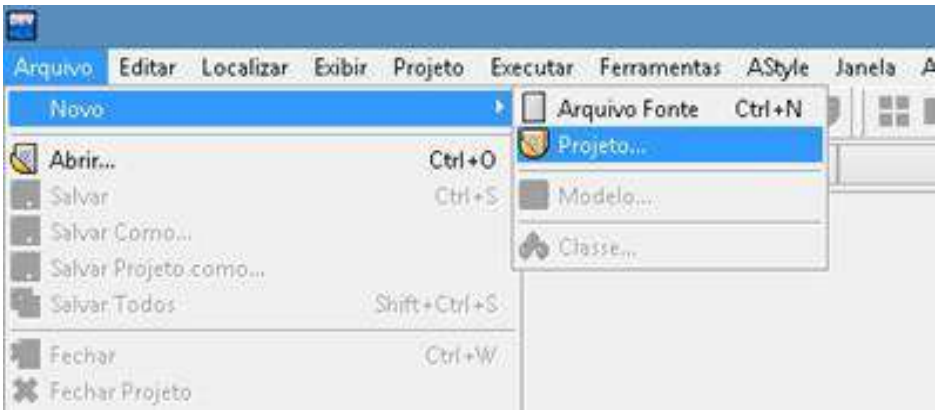
Figura 25 - Interface do Dev-C++



Fonte: Elaborada pelo autor

Acompanhe a Figura 26. Vamos testar começar um PROJETO de código em C. Para isso vamos acessar ARQUIVO >> NOVO >> PROJETO.

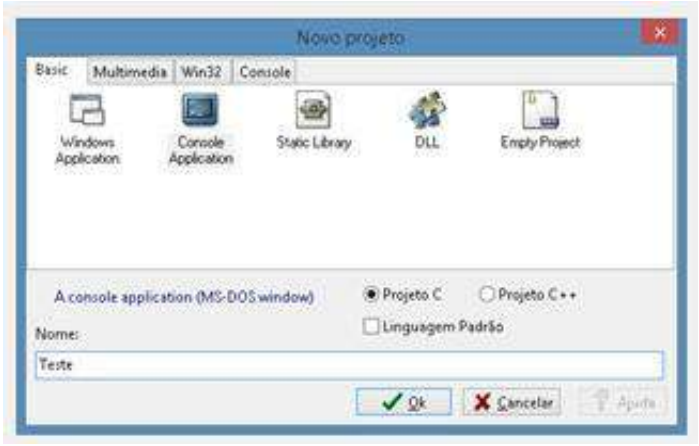
Figura 26 - Começando um projeto no Dev-C++



Fonte: Elaborada pelo autor

Observe a Figura 27. Nessa próxima tela de definição do projeto, vamos escolher “Console Application”, depois selecione o item “Projeto C” para indicar que trabalharemos com a Linguagem C especificamente. Indique um NOME para o projeto. Neste exemplo, colocamos “Teste”.

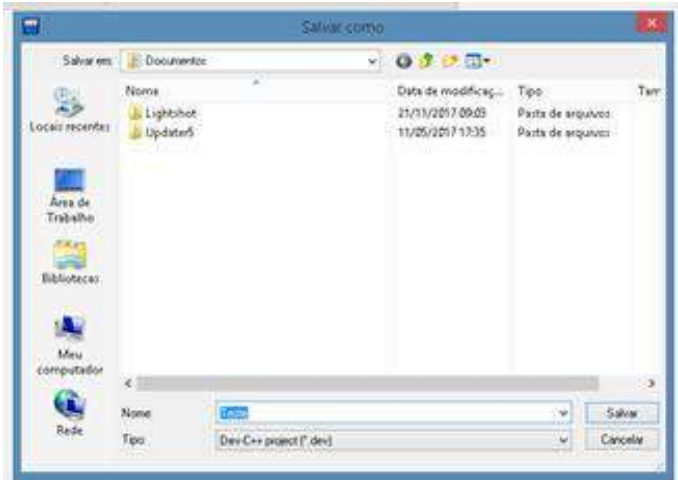
Figura 27 - Definindo o projeto no Dev-C++



Fonte: Elaborada pelo autor

Agora, na Figura 28, vamos indicar no nome para salvar o projeto (.dev). Colocamos “Teste” neste exemplo.

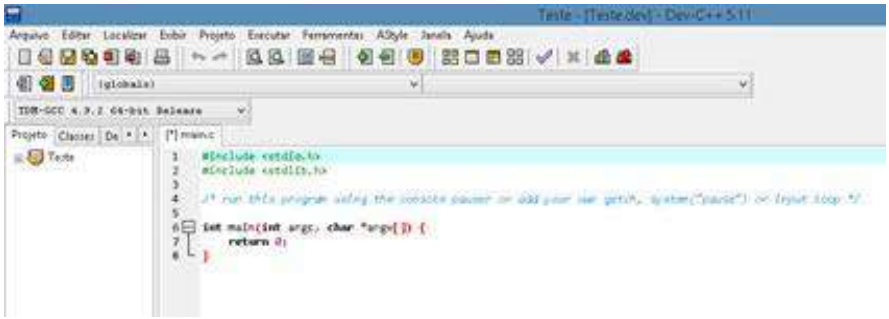
Figura 28 - Definindo o projeto no Dev-C++



Fonte: Elaborada pelo autor

Na próxima etapa, conforme a Figura 29, o Dev-C++ apresenta um “esqueleto” padrão de um arquivo (função principal) em Linguagem C. A partir desse código básico vamos começar a inserir nossos comandos de programação em C.

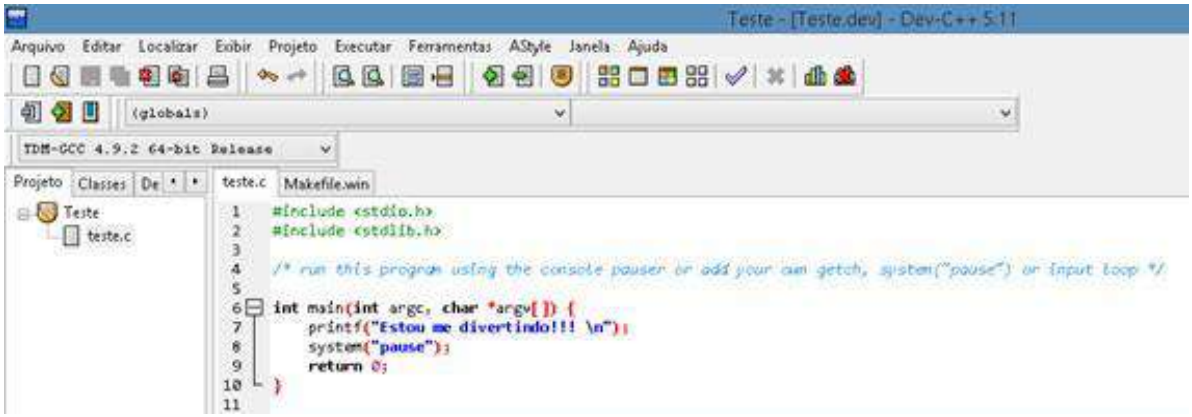
Figura 29 - Começando a programar o projeto no Dev-C++



Fonte: Elaborada pelo autor.

Adicionaremos duas instruções em C. Uma função de saída (semelhante ao “escreva”) chamada “printf” e outra função do sistema “system” para “pausar” a execução do programa em C. Acompanhe na Figura 30.

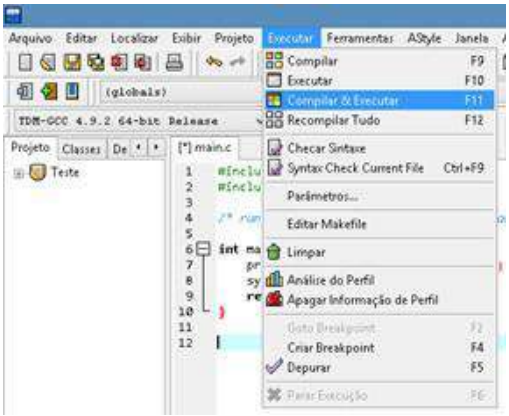
Figura 30 - Começando a programar o projeto no Dev-C++



Fonte: Elaborada pelo autor.

O passo seguinte será compilar e executar o nosso código fonte em C. Vamos acessar o menu EXECUTAR conforme a Figura 31. Observe que temos a opção de somente compilar (gerar novo executável com as últimas modificações no código), somente executar (a última versão já compilada) ou então "compilar & executar", em sequência (atalho F11). Vamos utilizar o F11 para compilar e executar.

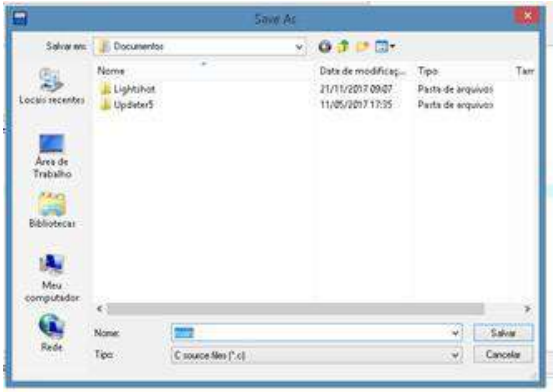
Figura 31 - Executando e compilando o projeto no Dev-C++



Fonte: Elaborada pelo autor.

Antes de compilar e executar, o Dev-C++ (a primeira vez que compilamos) solicitará o local onde vamos colocar o arquivo.c (source files). Acompanhe na Figura 32. Definimos o nome Teste.c para o arquivo de código fonte.

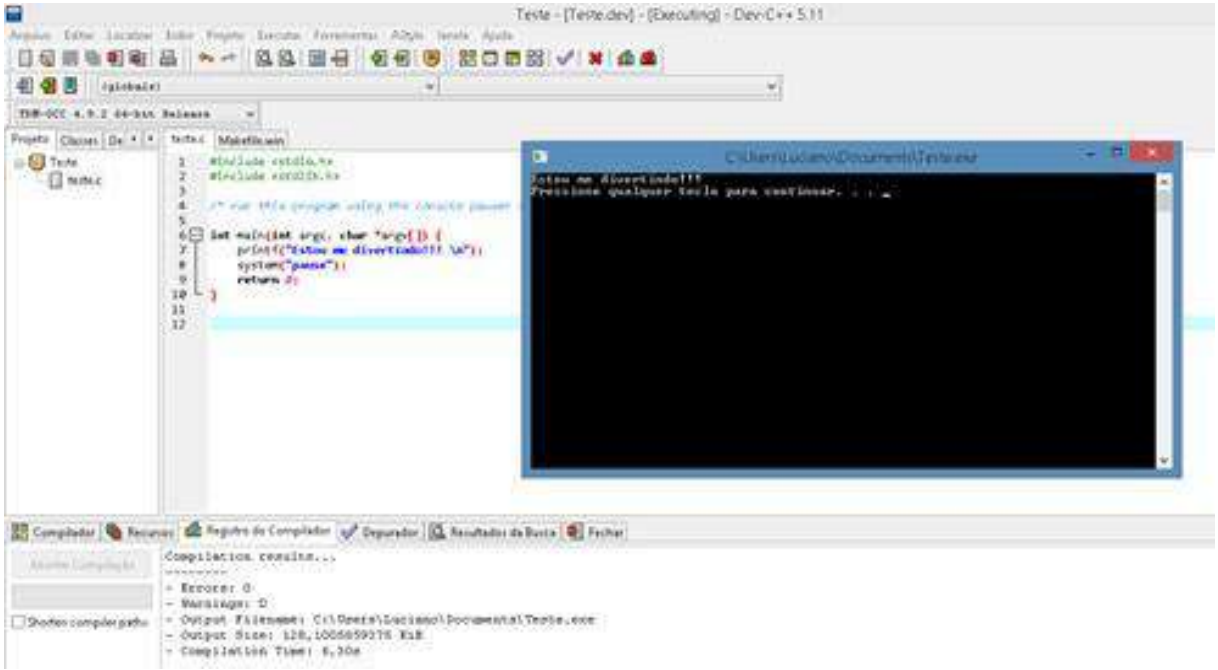
Figura 32 - Salvando o source (.c) do projeto no Dev-C++



Fonte: Elaborada pelo autor.

Finalmente temos a execução do código. Acompanhe na Figura 33. Reitero que o C compila o arquivo do código fonte. Olhe na pasta onde salvou o código e verá que depois da execução do código, o Dev-C ++ gerou (compilou) um arquivo executável (binário) a partir do seu código fonte. Essa é a diferença entre compilação e interpretação. Não esqueça!

Figura 33 - Execução do projeto no Dev-C++



Fonte: Elaborada pelo autor.

Caso tenha alguma dificuldade de instalação procure apoio do tutor presencial. Aproveite para assistir ao vídeo a seguir.

Saiba mais!

Que tal dar uma espiada em como se instala o Dev-C++ ?

Achou complicado? Vamos espiar um vídeo para entender melhor?

Por enquanto não vamos usá-lo ainda, mas no próximo tópico vamos iniciar o seu uso. Que tal adiantar um pouquinho e já ficar preparado para aprender um pouco de Linguagem C em paralelo com Português Estruturado?

Não desista. Seja persistente! Instale o VISUALg e o Dev-C++.

Assista ao vídeo da instalação do Dev-C++.

Acesse: https://www.youtube.com/watch?v=V6jWIzVcqow


Chegamos ao fim do tópico IV - Ambiente de Desenvolvimento de Algoritmo. A instalação do VISUALG e do Dev-C++ são muito importantes para a disciplina. Neste tópico, trabalhamos o ambiente de desenvolvimento VISUALG e suas características. Aprendemos como instalar e como utilizar o VISUALG para criar algoritmos. Também trabalhamos os testes de mesa para verificação do funcionamento dos algoritmos. Finalizando o tópico IV, realizamos a instalação do DEV-C++. Aproveite para consultar o glossário se tiver a dúvida de algum termo. Vamos partir para algumas questões de aprendizagem e para a revisão final do tópico. Não se esqueça de olhar os sites e vídeos indicados que irão ampliar os seus horizontes de conhecimentos sobre o tema da unidade. Vamos para os próximos tópicos nos quais colocaremos em prática o desenvolvimento de algoritmos utilizando o VISUALG e o Dev-C++ !

Vamos rever?

Um Ambiente de Desenvolvimento Integrado é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo.

Ou seja, é um conjunto de ferramentas computacionais usadas no desenvolvimento de programas ou aplicações. Normalmente contam com editor, compilador e outras ferramentas de apoio ao desenvolvimento na linguagem pretendida. Um computador só compreende linguagem de máquina. Já o ser humano se expressa em linguagem natural (português, por exemplo).

As linguagens de programação funcionam como interfaces entre o ser humano e o computador. As linguagens de alto nível (mais próximas da linguagem natural) facilitam a compreensão do ser humano para a realização do problema e podem ser interpretadas ou compiladas para linguagem de máquina permitindo que o computador as execute. Para a interpretação de um arquivo em linguagem de alto nível (como português estruturado), sempre será necessário ter instalado o interpretador no computador no qual vamos executar o código. Assim, para interpretar um algoritmo em português estruturado, precisaremos sempre ter instalado o VISUALg.

No caso da Linguagem de programação C, uma vez que tenhamos compilado o arquivo do código fonte não precisaremos mais do Dev-C++ para executar o programa. Utilizaremos dois IDEs para desenvolvimento: VISUALg e Dev-C++.

Sites indicados

1) Melhores IDEs para Portugol
http://www.galirows.com.br/meublog/blog/melhores-portugol

2) Apoio Informática - VISUALg
http://www.apoioinformatica.inf.br/produtos/visualg

3) Manual de Referência - VISUALg
http://www.inf.ufsc.br/~bosco.sobral/ensino/ine5201/Visualg2_manual.pdf

4) Software VISUALg 2.0 Software VISUALg 2.0 - Bruno Tonet - UCS
http://www.dainf.ct.utfpr.edu.br/~eteocles/visualg.pdf

Audiovisuais indicados

1) Como Baixar - Instalar e Configurar o Dev C++ no Windows 10 ?
https://www.youtube.com/watch?v=V6jWIzVcqow

2) Como criar um programa simples no Dev C++ ?
https://www.youtube.com/watch?v=fB5q97HtwMU

3) Apresentação do Dev C++ - Linguagem C
https://www.youtube.com/watch?v=vAeBYwaYcqE

4) Programando em C utilizando GNU/Linux! Como é isso?
https://www.youtube.com/watch?v=oZeezrNHxVo

Questões de autoaprendizagem

a) Vamos testar o VISUALg? Ainda não aprendemos todos os comandos desse exemplo que vamos testar, mas isso não importa. Vamos matar a curiosidade. Digite no VISUALg o código abaixo e execute. Qual foi a saída?

```
algoritmo “salario”
// Seção de Declarações  var sal:real
i,contador:inteiro

inicio
// Seção de Comandos  i<-0;
contador<-0;
para i de 1 ate 5 passo 1 faca
   escreva(“Digite o salario do funcionário “,i, “: “)  leia (sal)
   se sal>500 entao
      contador<-contador+1  fimse
fimpara
escreval(contador, “ Funcionários recebem salários superiores a R$ 300,00.” )  fimalgoritmo
```

Questões de autoaprendizagem

b) Vamos testar outro? Experimente esse agora. Digite no VISUALg o código abaixo e execute. Qual foi a saída?

```
algoritmo "Média"  var
Media, P1, P2, P3, P4: real

inicio
    Escreva("Digite a nota 1: ")
    Leia (P1)
    Escreva("Digite a nota 2: ")
     Leia (P2)
    Escreva("Digite a nota 3: ")
     Leia (P3)
    Escreva("Digite a nota 4:")
    Leia (P4)


    Media <- (P1 + P2 + P3 + P4) / 4
    Se (Media >= 7) entao
    Escreval (Media," Aluno Aprovado!")
    Senao
    Escreval (Media," Aluno Reprovado!")
    Fimse
fimalgoritmo
```

Questões de autoaprendizagem

c) Vamos testar o Dev-C++? Não sabemos nada da Linguagem C, mas isso também não importa. Agora o que importa é experimentar. Digite no Dev-C++ o código abaixo e execute. Qual foi a saída?

```
#include <stdio.h>
int main(void)
{
        printf("Meu primeiro programa - Olá Mundo!\n");
        return 0;
}
```

Resposta do Teste de Lógica

Confira, abaixo, a resposta do teste de lógica da página 11 desta unidade!

Resposta: Pergunte a qualquer um deles: Qual a porta que o seu companheiro apontaria como sendo a porta da liberdade?

Explicação: O mentiroso apontaria a porta da morte como sendo a porta que o seu companheiro apontaria, e o sincero, sabendo que seu companheiro sempre mente, diria que ele apontaria a porta da morte como sendo a porta da liberdade. Conclusão: Os dois apontariam a porta da morte. Portanto, é só seguir pela outra porta.


Opa! Tá na hora de começarmos mais um tópico ! Aqui começa a melhor parte! A programação para valer! Todo mundo empolgado?
Legal professor vamos arrebentar!
Sim pessoal. Neste tópico vamos utilizar as estruturas sequenciais e de seleção. Destaco que entender a sequência de execução e tomar decisões em programação é super importante. Vamos aprender a indicar ao computador como avaliar condições e tomar decisões a partir dela. É realmente ensinar o computador a decidir e resolver problemas.
Agora sim hein professor? Show!

Unidade 2

Estruturas de Controle e de Repetição

Ao término desta unidade, esperamos atingir o seguinte objetivo:

- Elaborar algoritmos que façam uso, em conjunto, de estruturas de controle do tipo sequenciais, de repetição e de seleção.

Cadê meu VISUALg? Opa, achei! Carregando...

Não se esqueça de baixar e instalar o VISUALg em seu computador.

Se tiver alguma dúvida de como fazer isso, retorne ao tópico 1.4.3 da unidade 1 e reveja as informações.

Para baixar o VISUALg, clique aqui!

As estruturas de controle de dados são responsáveis pela manipulação dos dados. Essas estruturas possuem uma lógica de operação e estabelecem uma sequência de ações a serem efetuadas ou verificadas.

Em um algoritmo, podemos considerar o seguinte conjunto de estruturas básicas:

- **Sequencial**: sequência de comandos executada em uma ordem linear, de cima para baixo, da esquerda para a direita, respeitando a pontuação e o alinhamento (identação) especificado. É semelhante ao processo de leitura.

- **Condicional ou de Seleção**: a partir de um teste condicional, com base em lógica convencional, uma instrução ou bloco de instruções será executado ou não, dependendo se o resultado foi falso ou verdadeiro. Pode ser SIMPLES, COMPOSTA, ENCADEADA e de MÚLTIPLA ESCOLHA.

- **Repetição**: uma estrutura determina que certas ações deverão ser repetidas, de acordo com o resultado de um teste condicional.

2.1 Estruturas de Controle

2.1.1 Estrutura sequencial

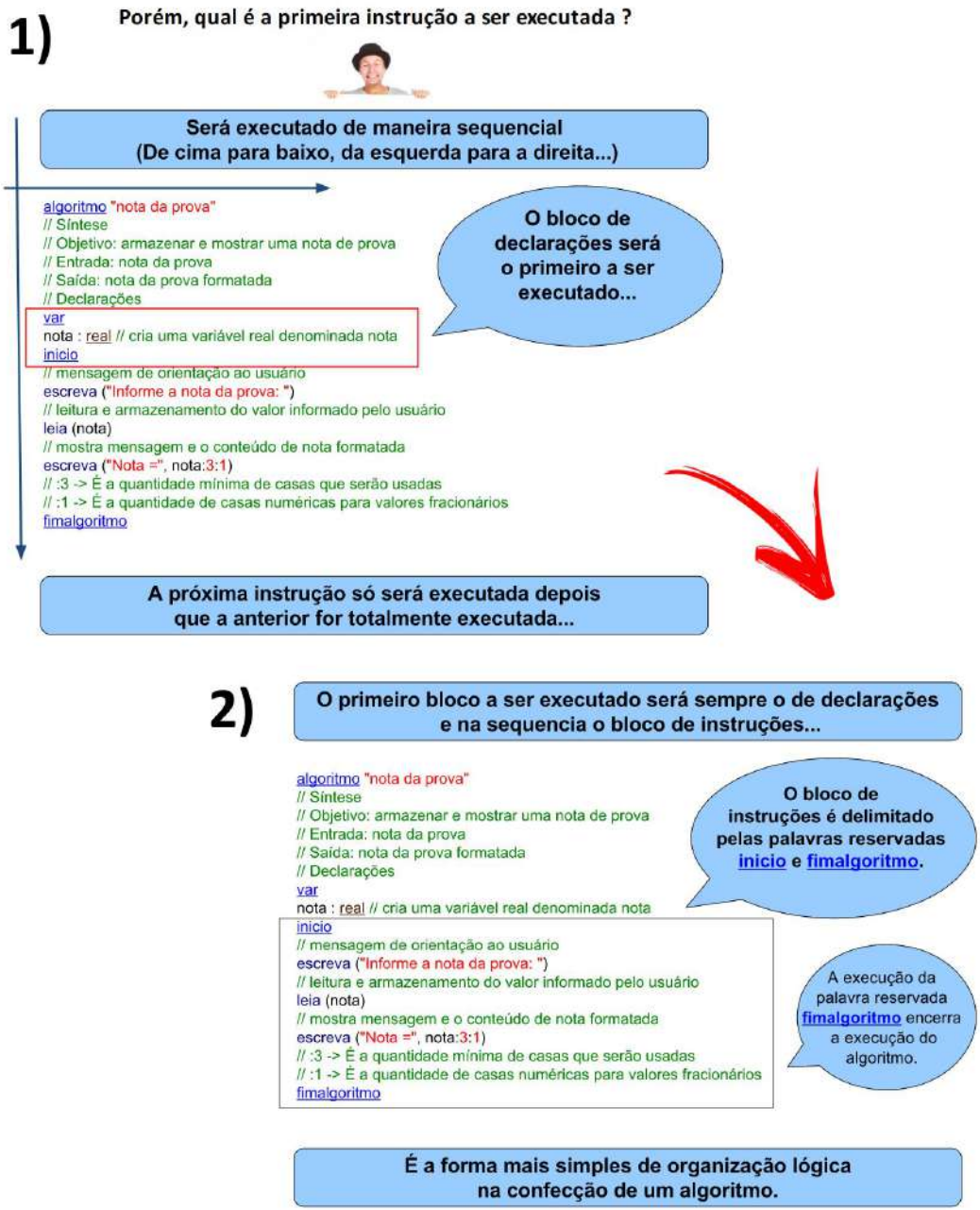
A estrutura sequencial é a mais convencional entre todas as estruturas, pois ela executa uma instrução de cada vez: o encerramento da primeira instrução permite o acionamento da instrução seguinte, respeitando a ordem de cima para baixo, linha por linha. Dentro da linha selecionada, sempre da esquerda para a direita, semelhante ao nosso procedimento de leitura de textos.

No próximo exemplo, um algoritmo proposto é descrito e respeita as regras de organização e sintaxe do português estruturado.

```
algoritmo "nota da prova"
  // Síntese
  // Objetivo: armazenar e mostrar uma nota de prova
  // Entrada: nota da prova
  // Saída: nota da prova formatada
  // Declarações
var
  nota : real // cria uma variável real denominada nota
  inicio
  // mensagem de orientação ao usuário
 escreva ("Informe a nota da prova: ")
  // leitura e armazenamento do valor informado pelo usuário
  leia (nota)
  // mostra mensagem e o conteúdo de nota formatada
  escreva ("Nota =", nota:3:1)
  // :3 -> É a quantidade mínima de casas que serão usadas
  // :1 -> É a quantidade de casas numéricas para valores fracionários
  fimalgoritmo
```

Agora observe a Figura 1. O algoritmo é o mesmo apresentado na página anterior.

Figura 1 - Exemplo de Estrutura Sequencial



Fonte: Elaborada pelo autor.

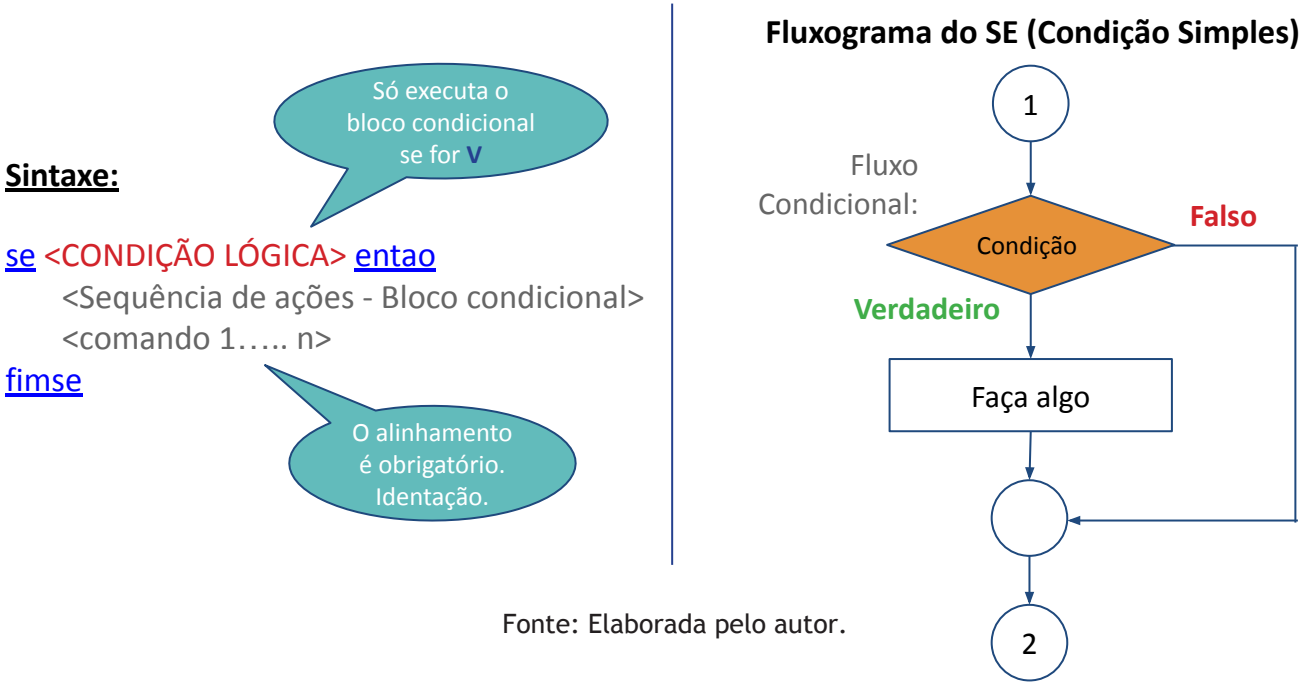
2.1.2 Estrutura de Seleção

A estrutura de seleção é também conhecida como uma estrutura condicional. Ela permite a escolha ou a definição de um caminho sequencial a ser executado. Porém, a escolha é definida de acordo com o resultado de uma verificação ou teste condicional.
Estes testes (avaliações de condições) utilizam a lógica convencional e têm resultados do tipo lógico (V ou F). Essas expressões que são avaliadas empregam operadores relacionais, lógicos e aritméticos.

2.1.3 Seleção Simples

A primeira estrutura condicional de seleção simples é o SE (if). Nesta estrutura, avalia-se um teste condicional, que sendo verdadeiro, executa uma ou mais instruções em um bloco de instruções, denominado bloco condicional.

Figura 2 - Sintaxe da Seleção Simples (SE)



Fonte: Elaborada pelo autor.

Vamos refletir?

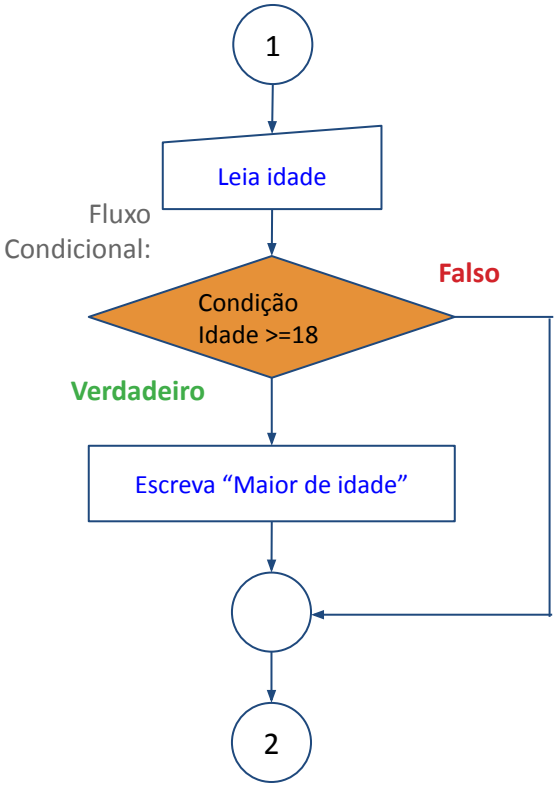
Vejamos um exemplo detalhado da estrutura de Seleção Simples (SE) de acordo com a Figura 3. Inicialmente, temos a declaração de uma variável IDADE. Depois, iniciando o algoritmo temos o comando leia, que recebe e armazena um valor na variável IDADE. O próximo comando é a estrutura SE (Seleção Simples) que avalia a expressão (IDADE>=18).

Vamos considerar que o usuário tenha digitado o valor 15 para IDADE. Neste caso, o resultado da comparação (15>=18) será **FALSO**, pois 15 não é maior ou igual a 18. A estrutura de seleção simples somente executa seu BLOCO CONDICIONAL (conjunto de instruções interno a estrutura SE - comandos identados dentro da estrutura) se o resultado da expressão (condição) que ela avalia for VERDADEIRO. Assim, a estrutura desvia a execução do algoritmo diretamente para o **fimse**, pulando todos os comando do BLOCO CONDICIONAL (observe o fluxograma da Figura 3) e depois termina o algoritmo no **fimalgoritmo**.

Agora, vamos imaginar que o usuário tenha digitado o valor 20 para a variável IDADE. Neste caso, o resultado da comparação **(20>=18)** será **VERDADEIRO**, pois 20 é maior ou igual a 18. A estrutura de seleção simples irá executar seu BLOCO CONDICIONAL (conjunto de instruções interno a estrutura SE - comandos identados dentro da estrutura), pois, desta vez, o resultado da expressão (condição) que ela avaliou foi VERDADEIRO (observe o fluxograma da Figura 3). Neste caso, o comando **escreva** fará a impressão na tela da **mensagem "Maior de idade"** e depois o algoritmo chega ao **fimse** e ao **fimalgoritmo** terminando a sua execução.

Figura 3 - Exemplo de Seleção Simples (SE)

```
algoritmo “Maior de idade”
var
IDADE : numérico
inicio
  leia (IDADE)
  se (IDADE >= 18) entao
      escreva (“Maior de idade”)
  fimse
fimalgoritmo
```



Fonte: Elaborada pelo autor.

2.1.4 Seleção composta (SE ENTÃO SENÃO)

A seleção composta sempre executa um dos blocos condicionais. De acordo com o resultado da avaliação de sua condição (expressão), se o resultado for VERDADEIRO, executa o bloco "verdadeiro" e, se for FALSO, executa o bloco "falso". Nunca irá executar os dois blocos! Ela executa um ou outro, pois sempre ocorre a seleção do bloco a ser executado de acordo com o resultado lógico.

Figura 4 - Sintaxe da Seleção Composta (SE ENTÃO SENÃO)

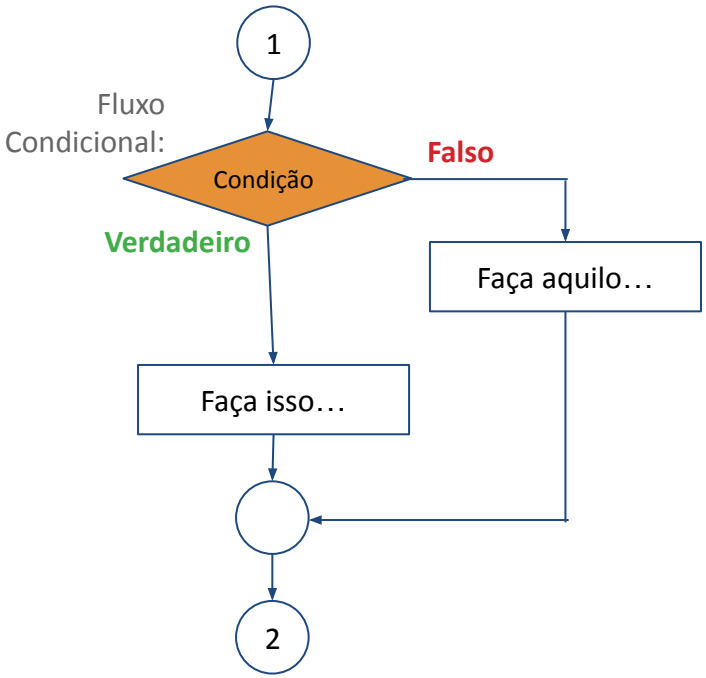
Sintaxe:

Só executa se for **V**

```
se <CONDIÇÃO LÓGICA> entao
    <Sequência de ações para VERDADEIRO>
    senao
    <Sequência de ações para FALSO>
fimse
```

Só executa se for **F**

Fluxograma do SE ENTÃO... SENÃO...



Fonte: Elaborada pelo autor.

Vamos refletir?

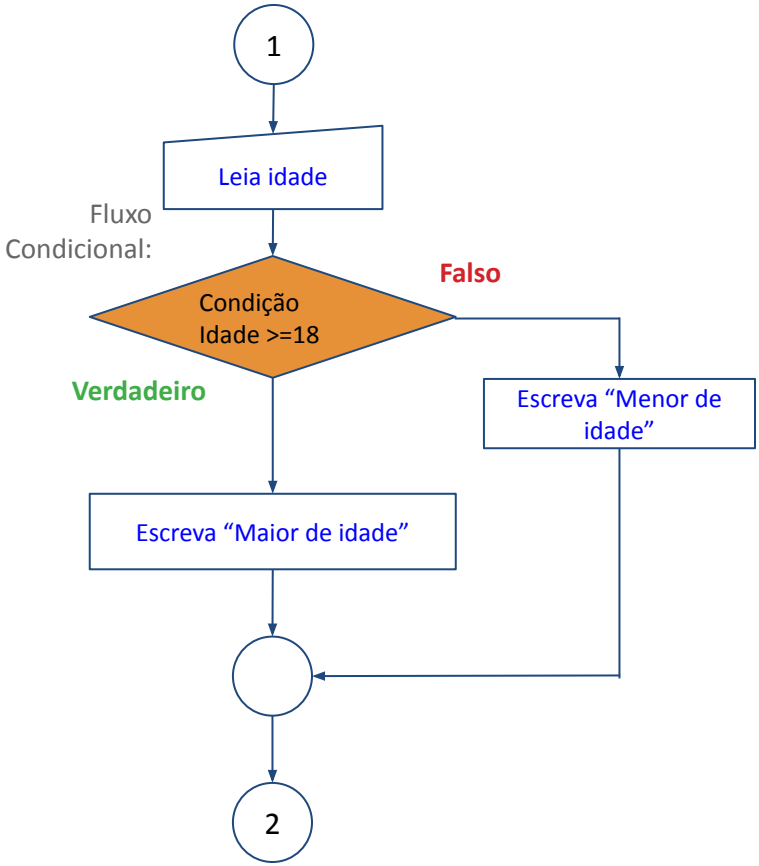
Para compreender melhor a estrutura de Seleção composta (SE ENTÃO SENÃO) vamos verificar um exemplo de acordo com a Figura 5. O primeiro passo é a **declaração de uma variável IDADE**. Depois, iniciando o algoritmo, temos o comando leia que recebe e armazena um valor na variável IDADE. O próximo comando é a estrutura de Seleção composta (SE ENTÃO SENÃO) que avalia a **expressão (IDADE>=18)**. Considere que o usuário tenha digitado o valor 10 para IDADE. Neste caso, o resultado da comparação (10>=18) será **FALSO**, pois 10 não é maior ou igual a 18. A estrutura de Seleção composta (SE ENTÃO SENÃO) pode executar um de seus BLOCOS CONDICIONAIS (conjunto de instruções interno a estrutura SE - comandos identados dentro da estrutura separados em dois blocos do ENTÃO e do SENÃO) de acordo com o resultado da expressão (condição) que ela avalia. Como o resultado foi **FALSO**, ela irá executar o bloco do **SENÃO**. Assim, a estrutura desvia a execução do algoritmo diretamente para o **bloco do SENÃO** e, dentro dele, executa o comando **escreva com a mensagem "Menor de idade"**. Depois disso chega ao **fimse** e termina o algoritmo no **fimalgoritmo**.

Agora, vamos imaginar que o usuário tenha digitado o valor 18 para a variável IDADE. Neste caso, o resultado da comparação **(18>=18)** será **VERDADEIRO**, pois 18 é igual a 18. A estrutura de Seleção composta (SE ENTÃO SENÃO) pode executar um de seus BLOCOS CONDICIONAIS (conjunto de instruções interno a estrutura SE - comandos identados dentro da estrutura, separados em dois blocos do ENTÃO e do SENÃO) de acordo com o resultado da expressão (condição) que ela avalia (observe o fluxograma da Figura 5).

Como o resultado foi **VERDADEIRO**, a estrutura desvia o fluxo de execução para o **bloco do ENTÃO**. Assim, o algoritmo executará o comando **escreva** e fará a impressão na tela da **mensagem "Maior de idade"**. Depois, o algoritmo chega ao **fimse** e ao **fimalgoritmo** terminando a sua execução.

Figura 5 - Exemplo de Seleção Composta (SE ENTÃO SENÃO)

```
algoritmo “Maior de idade 2”
início
  IDADE : numérico
  leia (IDADE)
  se (IDADE >= 18) entao
      escreva (“Maior de idade”)
    senão
      escreva (“Menor de idade”)
  fimse
fimalgoritmo
```



Fonte: Elaborada pelo autor.

2.1.5 Seleção encadeada

A instrução de seleção SE pode ser aninhada uma dentro da outra. É possível colocar outras instruções de seleção simples ou compostas dentro dos blocos verdadeiros (então) ou falsos (senão). Todas as instruções programáveis podem estar dentro de qualquer um dos blocos da instrução de seleção, inclusive a própria instrução de seleção. Não há limite na quantidade de testes condicionais que podem estar dentro de um outro. O aninhamento de instruções é ilimitado e vai depender da complexidade lógica que se deseja resolver. Observe o exemplo de sintaxe da estrutura de seleção encadeada conforme a Figura 6.

Figura 6 - Sintaxe da Seleção Encadeada

Sintaxe geral

```
se (<condição 1>) entao
  se (<condição 2>) entao
     <bloco de comandos 1>
  fimse
senao
  se (<condição 3>) entao
     <bloco de comandos 2>
  senao
     <bloco de comandos 3>
  fimse
  <comando n>
fimse
```

```
se (<condição 1>) entao

   se (<condição 2>) entao
      <bloco de comandos 1>
   fimse

senao

   se (<condição 3>) entao
      <bloco de comandos 2>
   senao
      <bloco de comandos 3>
   fimse

   ... <comando n>
fimse
```

Fonte: Elaborada pelo autor.

Como aprendizado de Seleção Encadeada, vamos codificar, executar e testar no VISUALg o seguinte exemplo:

Construa um algoritmo que exiba mensagens de acordo com um determinado valor de idade recebido, avaliando as seguintes condições:

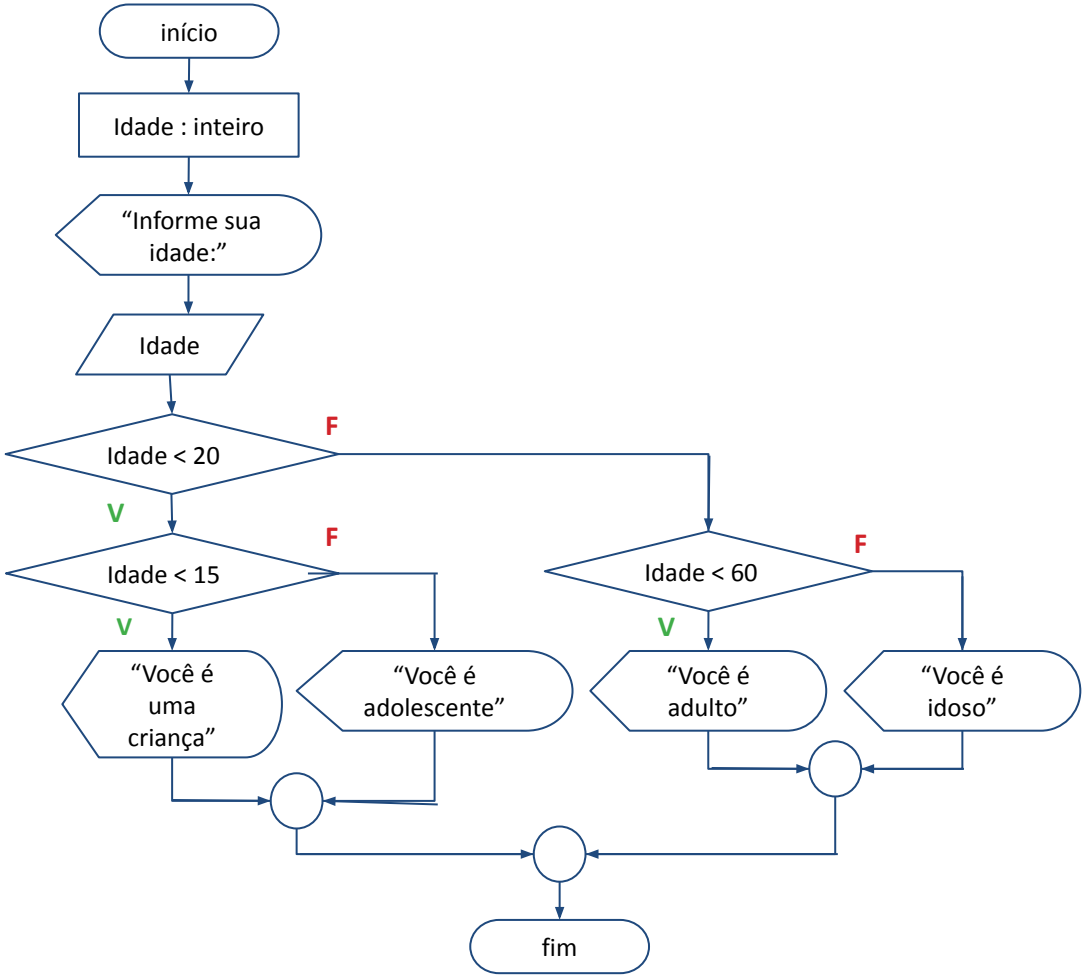
- Exibir “Você é uma criança...” para idade menor que 15 anos;
- Exibir “Você é adolescente...” para idade maior ou igual a 15 e menor que 20 anos;
- Exibir “Você é adulto...” para idade maior ou igual a 20 e menor que 60 anos;
- Exibir “Você é idoso...” para idade maior ou igual a 60 anos.

Resolução da atividade proposta - Seleção Encadeada - Algoritmo

```
algoritmo “avaliação da idade”
// Síntese
// Objetivo: avaliar a idade de uma pessoa
// Entrada: idade
// Saída: confirmação de sua classificação
// Declarações
var
idade: inteiro
início
escreva (“Informe sua idade: ”)
leia(idade)
se (idade < 20) entao
    se (idade < 15) entao
        escreva (“Você é uma criança”)
    senao
        escreva (“Você é adolescente”)
    fimse // encerra o condicional para os 15 anos
senao
    se (idade < 60) entao
        escreva (“Você é adulto”)
    senao
        escreva (“Você é idoso”)
    fimse // encerra o condicional para os 60 anos
fimse // encerra o condicional para os 20 anos
fimalgoritmo
```

Fonte: Algoritmo elaborado pelo autor.

Resolução da atividade proposta - Seleção Encadeada - Fluxograma



Fonte: Fluxograma elaborado pelo autor.

2.1.6 Seleção de Múltipla Escolha

Uma outra forma de seleção muito utilizada nas avaliações de igualdade, é a seleção de múltipla escolha. Por meio dela, é possível testar somente a igualdade de um valor que está na condição inicial, percorrendo as diversas escolhas disponíveis. Em caso de resultado verdadeiro desse teste, temos a execução de um bloco de instruções ou de uma única instrução específica separada pela nova palavra reservada **caso**.

Figura 7 - Sintaxe da Seleção de Múltipla Escolha

Essa instrução de seleção verifica somente a igualdade de valores entre **<identificador>** e um dos **<valor_n>** especificado após a palavra reservada caso.

O **<identificador>** só pode ser substituído por um elemento (variável ou constante) do tipo inteiro ou caracter com um único caracter.

A opção **outrocaso** é opcional e verifica se todos os outros casos resultaram em falso, se sim, então ela é que será executada (**outrocaso é verdadeiro**).

Porém é possível utilizar uma instrução escolha sem possuir **outrocaso** definido.

Fonte: Elaborada pelo autor.

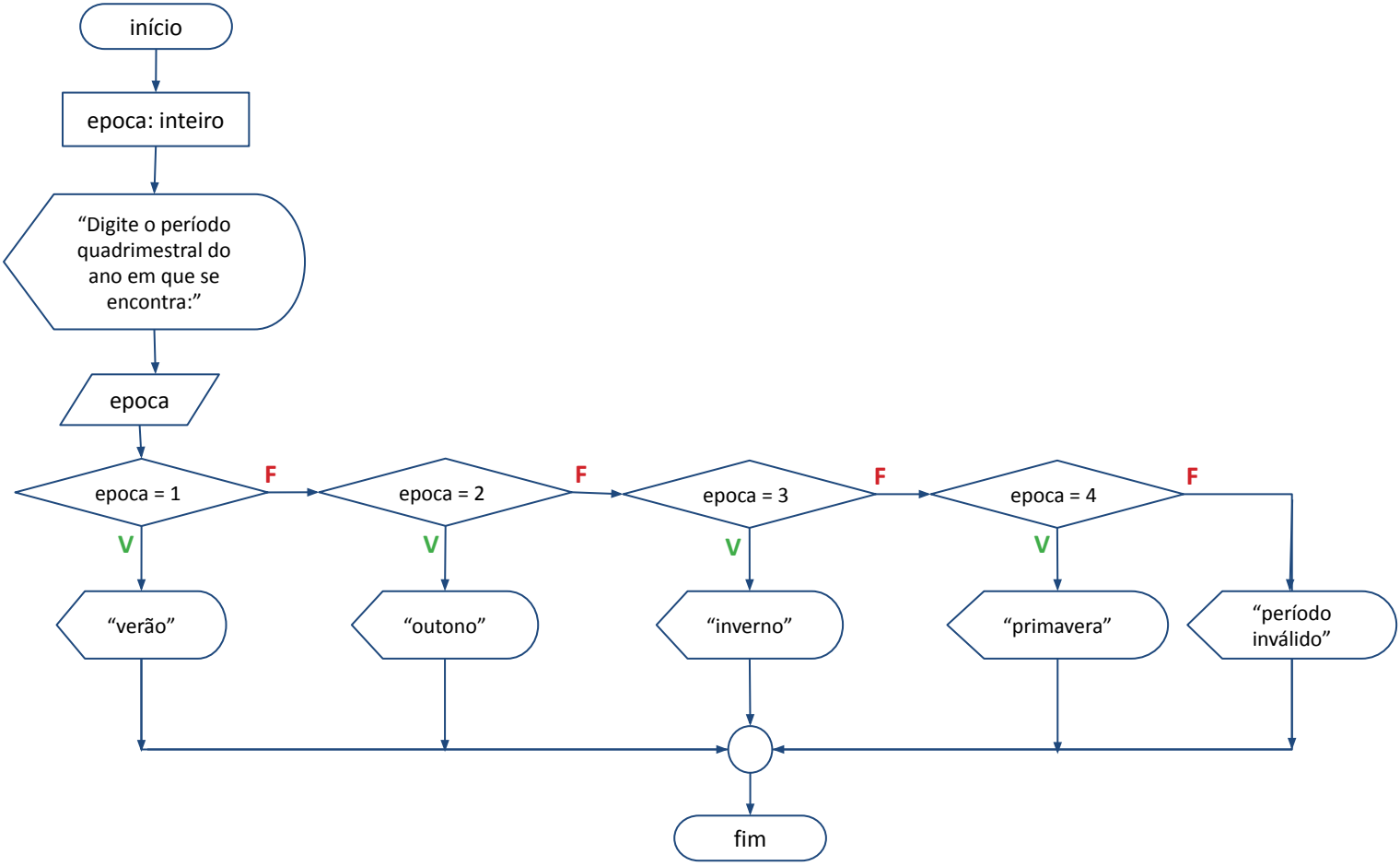
Como aprendizado de Seleção de Múltipla Escolha, vamos codificar, executar e testar no VISUALg o seguinte exemplo: Faça um algoritmo que monte um menu de opções com as estações do ano (1 - Verão, 2 - Outono, 3 - Inverno e 4 - Primavera) para que o usuário possa escolher. Se ele digitar algo errado, indique "Opção Inválida". No final da execução, indicar ao usuário a opção que foi escolhida. Elabore o fluxograma ao terminar o algoritmo.

Resolução da atividade proposta - Seleção de Múltipla Escolha - Algoritmo

```
algoritmo "estação climática do ano"
// Síntese
// Objetivo: identificar qual a estação climática do ano atual
// Entrada: código numérico que identifica a estação atual
// Saída: descreve a estação do ano atual
// Declarações
var
opcao : inteiro
início
escreva ("Escolha a opção (1-Verão, 2-Outono, 3-Inverno, 4-Primavera):")
leia(opcao)
// Instrução de múltipla escolha sobre época
escolha(opcao)
caso 1 // caso época esteja armazenando o valor inteiro 1
    escreva ("Escolheu -> Verão") // mensagem verão será apresentada
caso 2 // caso época esteja armazenando o valor inteiro 2
    escreva ("Escolheu -> Outono") // mensagem outono será apresentada
caso 3 // caso época esteja armazenando o valor inteiro 3
    escreva ("Escolheu -> Inverno") // mensagem inverno será apresentada
caso 4 // caso época esteja armazenando o valor inteiro 4
    escreva ("Escolheu -> Primavera") // mensagem primavera será apresentada
outrocaso // caso época não seja igual as opções anteriores
    escreva ("Opção inválida") // mensagem de erro
fimescolha
fimalgoritmo
```

Fonte: Algoritmo elaborado pelo autor.

Resolução da atividade proposta - Seleção de Múltipla Escolha - Fluxograma



Fonte: Fluxograma elaborado pelo autor.

Caso tenha alguma dificuldade de utilização do VISUALg retorne à unidade 1 e consulte o tópico 1.4.3. Aproveite para assistir ao vídeo a seguir e aprenda um pouco mais sobre as Estruturas de Controle.

Saiba mais!

Que tal dar uma espiada em um material de Lógica com VISUALg? Achou complicado? Vamos espiar um conjunto de vídeos para entender esses conceitos um pouco melhor?

Assista a alguns vídeos sobre Lógica de Programação disponíveis no YOUTUBE.

Não desista. Seja persistente!

Acesse:

https://www.youtube.com/watch?v=6-_leAMCi8M&list=PLlUjQffi3XKOc2OjC5aCekmxmhC5kSm70

Vamos rever?

As estruturas de controle de dados são responsáveis pela manipulação de dados. Elas possuem uma lógica de operação e estabelecem uma sequência de ações a serem efetuadas ou verificadas. A estrutura sequencial é uma estrutura básica de um algoritmo e executa os comandos como um processo de leitura, de forma natural, em uma ordem linear, de cima para baixo, da esquerda para a direita, respeitando a pontuação e o alinhamento (identação) especificado. A estrutura condicional ou de seleção realiza sua operação a partir de um teste condicional. Com base em lógica convencional e a partir do resultado dessa condição, uma instrução ou bloco de instruções será executado ou não. A execução ou não dependerá se o resultado lógico foi falso (F) ou verdadeiro (V). A estrutura condicional pode ser SIMPLES, COMPOSTA ou ENCADEADA. A estrutura simples é o SE.

A estrutura composta é o SE ENTÃO SENÃO. A estrutura encadeada é o aninhamento de estruturas simples e compostas. A instrução de seleção SE pode ser aninhada uma dentro da outra, seja no bloco verdadeiro (então) ou no falso (senão).

Para fechar, a estrutura de seleção MÚLTIPLA ESCOLHA é muito utilizada nas avaliações de igualdade. Por meio dela, é possível testar somente a igualdade de um valor que está na condição inicial, percorrendo as diversas escolhas disponíveis. Em caso de resultado verdadeiro desse teste, temos a execução de um bloco de instruções ou de uma única instrução específica separada pela nova palavra reservada caso. Todas as instruções programáveis podem estar dentro de qualquer um dos blocos da instrução de seleção, inclusive a própria instrução de seleção. Não há limite na quantidade de testes condicionais que podem estar dentro de um outro.

Sites indicados

1) Portal Posso ajudar? VISUALg: Exercícios Para Iniciantes I
http://programadorantigo.blogspot.com.br/2015/02/visualg-exercicios-para-iniciantes_8.html

2) Portal Posso ajudar? VISUALg: Exercícios Para Iniciantes II
http://programadorantigo.blogspot.com.br/2015/02/visualg-exercicios-para-iniciantes_9.html

3) Portal Posso ajudar? VISUALg: Exercícios Para Iniciantes III
http://programadorantigo.blogspot.com.br/2015/02/visualg-exercicios-para-iniciantes.html

4) Portal Posso ajudar? VISUALg: Exercícios SE, SENAO I
http://programadorantigo.blogspot.com.br/2015/02/visualg-exercicios-se-senao_8.html

5) Portal Posso ajudar? VISUALg: Exercícios SE, SENAO II
http://programadorantigo.blogspot.com.br/2015/02/visualg-exercicios-se-senao_9.html

6) Portal Posso ajudar? VISUALg: Exercícios SE, SENAO III
http://programadorantigo.blogspot.com.br/2015/02/visualg-exercicios-se-senao.html

Audiovisuais indicados

1) Estruturas Condicionais 1 - Curso de Algoritmos #07 - Gustavo Guanabara
https://www.youtube.com/watch?v=_g05aHdBAEY

2) Estruturas Condicionais 2 - Curso de Algoritmos #08 - Gustavo Guanabara
https://www.youtube.com/watch?v=7gGFHzqh4d8

3) VISUALg Aula 6 - Operadores Relacionais
https://www.youtube.com/watch?v=CmLej36gcOs&index=6

4) VISUALg Aula 12 - ESCOLHA CASO
https://www.youtube.com/watch?v=5FNebG7sBP4

Questões de autoaprendizagem

a) Faça um algoritmo que receba duas notas e calcule a média das notas. Caso a média seja maior de 8,0 o sistema deverá mostrar ao aluno um agradecimento (Excelente!). No final, o sistema deverá mostrar o valor da média do aluno.

b) Faça um algoritmo que avalie uma idade recebida e exiba uma mensagem “Você é uma criança” se a idade recebida for menor de 15 anos de idade.

c) A partir do algoritmo descrito na imagem abaixo, prepare um fluxograma que avalie a idade recebida e exiba uma mensagem “Você é uma criança” se a idade recebida for menor de 15 anos de idade.

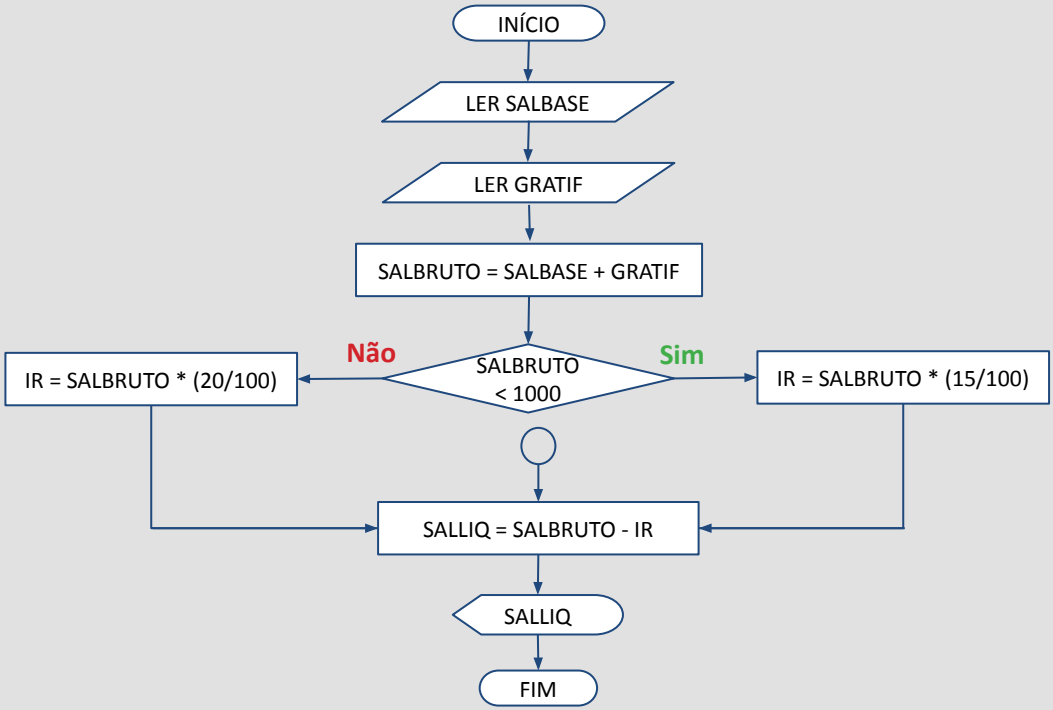
```
1  algoritmo "avaliação da idade"
2  // Síntese
3  // Objetivo: avaliar a idade de uma pessoa
4  // Entrada: idade
5  // Saída: confirmação desta pessoa ser criança
6  // Declarações
7  var
8  idade : inteiro // cria uma variável inteira denominada idade
9  inicio
10 escreva("Informe sua idade: ")
11 leia(idade)
12 se (idade < 15) entao // realiza um teste condicional sobre a idade
13    escreva ("Você é uma criança") // mostra mensagem se verdadeiro
14 fimse // encerra a instrução de seleção ou condicional
15 fimalgoritmo
```

Questões de autoaprendizagem

d) Crie um algoritmo que receba o ano de nascimento de uma pessoa e calcule a sua idade. No final da execução, o sistema deverá, para mostrar os resultados, limpar a tela e exibir os seguintes dados: a indicação se é criança caso a idade seja menor de 15 anos, o ano de nascimento e a idade da pessoa.

e) Faça um algoritmo que calcule a média do aluno a partir de 3 notas recebidas e escreva aprovado, caso a média seja maior ou igual que 5,0, e reprovado, caso a média seja menor que 5,0. O sistema deve informar a média ao final da execução.

f) Crie o algoritmo a partir do fluxograma abaixo.



Fonte: Fluxograma elaborado pelo autor.

Questões de autoaprendizagem

g) Faça um algoritmo que receba um número e responda se ele é par ou impar, utilizando o RESTO de uma divisão inteira. O algoritmo deve empregar apenas Seleção Simples (SE) para a sua resolução.

h) Mude o algoritmo anterior (letra g) para que ele execute utilizando apenas Seleção Composta (Se... entao... senao...).

i) Faça um algoritmo que avalie uma idade recebida e exiba uma mensagem "Você é uma criança" se a idade recebida for menor de 15 anos. Para outras idades, deverá indicar que a pessoa é adulta. Depois de implementar o algoritmo, construa o fluxograma adequado.

j) Construa um algoritmo que avalie as condições abaixo e apresente as mensagens de acordo com cada situação.

Idade	Sexo	Mensagem
< 18 anos	Masculino (M)	"Garoto"
>=18 anos	Masculino (M)	"Homem"
< 18 anos	Feminino (F)	"Garota"
>=18 anos	Feminino (F)	"Mulher"
Em qualquer outra situação exibir "Algo errado..."		

Questões de autoaprendizagem

k) Faça um algoritmo que monte uma calculadora com as seguintes funções: (1) Soma (2) Subtração (3) Divisão (4) Multiplicação. A calculadora recebe os dois números, efetua a opção escolhida e apresenta o resultado.

l) Construa um algoritmo que, tendo como dados de entrada o preço de um produto e seu código de origem, apresente o preço junto de sua procedência. Caso o código não seja nenhum dos especificados, o produto deve ser encarado como importado. Siga a tabela de código a seguir:

Código de origem	Procedência
1	Sul
2	Norte
3	Leste
4	Oeste
5 ou 6	Nordeste
7, 8 ou 9	Sudeste
10 até 20	Centro-Oeste
25 até 30	Amazonas

Gabarito das questões de autoaprendizagem

a)

```
algoritmo "MEDIA"
   // Função :
   // Autor :
   // Data : 12/3/2011
   // Seção de Declarações
var
    MEDIA, NOTA1, NOTA2:real
inicio
// Seção de Comandos
    escreval("Informe a nota 1:")
    leia(NOTA1)
    escreval("Informe a nota 2:")
    leia(NOTA2)
    MEDIA <- (NOTA1 + NOTA2)/2
    se (MEDIA > 8.5) entao
       escreva("Excelente ")
    fimse
    escreva("MEDIA = ", MEDIA)
fimalgoritmo
```

b)

```
algoritmo "avaliação da idade"
// Síntese
// Objetivo: avaliar a idade de uma pessoa
// Entrada: idade
// Saída: confirmação desta pessoa ser criança
// Declarações
var
idade : inteiro // cria uma variável inteira denominada idade
inicio
escreva("Informe sua idade: ")
leia(idade)
se (idade < 15) entao // realiza um teste condicional sobre a idade
   escreva ("Você é uma criança") // mostra mensagem se verdadeiro
fimse // encerra a instrução de seleção ou condicional
fimalgoritmo
```

Gabarito das questões de autoaprendizagem

c)

d)

```
algoritmo "avaliação da idade"
// Síntese
// Objetivo: avaliar a idade de uma pessoa
// Entrada: idade
// Saída: confirmação desta pessoa ser criança
// Declarações
var
Idade, nascimento,ano_atual : inteiro
Inicio
//constante
ano_atual<-2011
escreva("Informe seu ano de nascimento: ")
leia(nascimento)
idade<-ano_atual-nascimento
limpatela
se (idade < 15) entao // realiza um teste condicional sobre a idade
   escreval ("Você é uma criança") // mostra mensagem se verdadeiro
fimse // encerra a instrução de seleção ou condicional
escreval ("Ano nascimento:",nascimento)
escreva ("Idade:",idade)
fimalgoritmo
```

Gabarito das questões de autoaprendizagem

e)

```
algoritmo "Média - Aprovado / Reprovado"
// Seção de Declarações
var
n1, n2, n3, media : real
inicio
// Seção de Comandos
escreva ("Informe a nota 1: ")
leia (n1)
escreva ("Informe a nota 2: ")
leia (n2)
escreva ("Informe a nota 3: ")
leia (n3)
media <- (n1+n2+n3)/3
se (media >= 5) entao
   escreval ("Aprovado!")
 senao
   escreval ("Reprovado!")
fimse
escreva ("A media é: ",media)
fimalgoritmo
```

f)

```
algoritmo "FOLHA DE PAGAMENTO"
var
   SALBASE,SALLIQ,SALBRUTO,GRATIF,IR:real
inicio
      escreval("Informe o salário base")
      leia(SALBASE)
      escreval("Informe a gratificação")
      leia(GRATIF)
      SALBRUTO <- SALBASE + GRATIF
      se (SALBRUTO < 1000) entao
         IR <- SALBRUTO * 15/100
      senao
           IR <- SALBRUTO * 20/100
      fimse
      SALLIQ <- SALBRUTO - IR
      escreval("O salário líquido:",SALLIQ)
fimalgoritmo
```

g)

```
algoritmo "PAR_IMPAR"
// Função :
// Autor :
// Data : 12/3/2011
// Seção de Declarações
var
   NUMERO : inteiro
inicio
// Seção de Comandos
   escreval("Informe um numero:")
   leia(NUMERO)
   se (NUMERO % 2) = 0 entao
      escreval("PAR")
   fimse
   se (NUMERO % 2) = 1 entao
         escreval("IMPAR")
   fimse
fimalgoritmo
```

h)

```
algoritmo "PAR_IMPAR"
// Função :
// Autor :
// Data : 12/3/2011
// Seção de Declarações
var
   NUMERO : inteiro
inicio
// Seção de Comandos
   escreval("Informe um numero:")
   leia(NUMERO)
   se (NUMERO % 2) = 0 entao
       escreval("PAR")
     senao
        escreval("IMPAR")
   fimse
fimalgoritmo
```

Gabarito das questões de autoaprendizagem

i)

```
algoritmo "avaliação da idade"
// Síntese
// Objetivo: avaliar a idade de uma pessoa
// Entrada: idade
// Saída: confirmação desta pessoa ser criança ou adulto
// Declarações
var
// cria uma variável inteira denominada idade
idade : inteiro
inicio
escreva("Informe sua idade: ")
leia(idade)
// realiza um teste condicional sobre a idade
se (idade < 15) entao
   // mostra mensagem se verdadeiro
   escreva ("Você é uma criança")
senao
   // mostra mensagem se falso
   escreva ("Você já é um adulto")
// encerra a instrução de seleção ou condicional
fimse
fimalgoritmo
```

Gabarito das questões de autoaprendizagem

j)

```
algoritmo "semnome"
var
sexo:caracter
idade:inteiro
inicio
Escreva("Entre com o sexo da pessoa (M/F):")
Leia (sexo)
Escreva ("Entre com a idade:")
Leia (idade)
se (sexo="M") E (idade>=18) entao
   Escreva ("Você é um homem")
senao
   se (sexo="M") E (idade<18) entao
      Escreva ("Você é um garoto")
   senao
      se (sexo="F") E (idade>=18) entao
         Escreva ("Você é uma mulher")
      senao
         se (sexo="F") E (idade<18) entao
            Escreva ("Você é uma garota")
         senao
            Escreva ("ALGO ERRADO!!!")
         fimse
      fimse
   fimse
fimse
fimalgoritmo
```

Gabarito das questões de autoaprendizagem

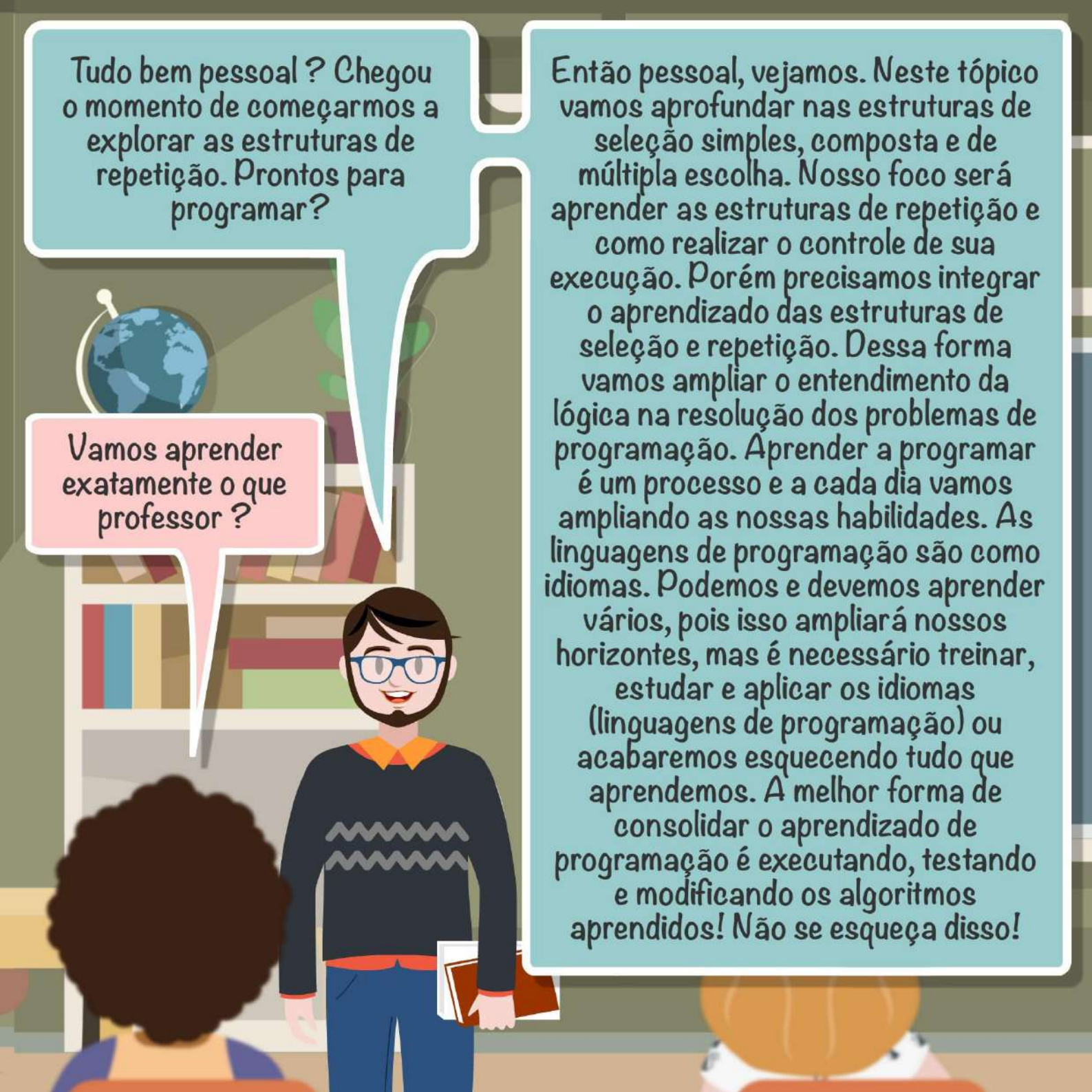
k)

```
algoritmo "estação climática do ano"
var
opcao: inteiro
a, b, resultado: real
inicio
escreval("Escolha a opção:")
escreval(" (1)-Soma, (2)-Subtração, (3)-Divisão, (4)-Multiplicação)")
escreva("  => ")
leia(opcao)
escreval
escreval ("Digite o valor do primeiro número (A)")
leia (a)
escreval ("Digite o valor do primeiro número (B)")
leia (b)
limpatela
escolha (opcao)
caso 1
   escreval ("Escolheu -> Soma  (Resultado=A+B)")
   escreval("----------------------------")
   resultado<-a+b
   escreval("Resultado:",resultado)
caso 2
   escreval ("Escolheu -> Subtração (Resultado=A-B)")
   escreval("----------------------------")
   resultado<-a-b
   escreval("Resultado:",resultado)
caso 3
   escreval ("Escolheu -> Divisão (Resultado=A/B)")
   escreval("----------------------------")
   resultado<-a/b
   escreval("Resultado:",resultado)
caso 4
   escreval ("Escolheu -> Multiplicação (Resultado=A*B)")
   escreval("----------------------------")
   resultado<-a*b
   escreval("Resultado:",resultado)
outrocaso
   escreval ("Opção Inválida")
   escreval("----------------------------")
fimescolha
fimalgoritmo
```

Gabarito das questões de autoaprendizagem

l)

```
algoritmo "Produtos.alg"
var
preco:real
codigo:inteiro
inicio
// Seção de Comandos
escreva("Informe o preço do produto:")
leia(preco)
escreva("Informe o codigo do produto:")
leia(codigo)
escolha codigo
   caso 1
        Escreva ("Veio do Sul com o preço:",preco)
   caso 2
        Escreva ("Veio do Norte com o preço:",preco)
   caso 3
        Escreva ("Veio do Leste com o preço:",preco)
   caso 4
        Escreva ("Veio do Oeste com o preço:",preco)
   caso 5,6
        Escreva ("Veio do Nordeste com o preço:",preco)
   caso 7,8,9
        Escreva ("Veio do Sudeste com o preço:",preco)
   caso 10,11,12,13,14,15,16,17,18,19,20
        Escreva ("Veio do Centro-Oeste com o preço:",preco)
   caso 25,26,27,28,29,30
        Escreva ("Veio do Amazonas com o preço:",preco)
   outrocaso
        Escreva ("Produto Importado com o preço:",preco)
fimescolha
fimalgoritmo
```


Tudo bem pessoal ? Chegou o momento de começarmos a explorar as estruturas de repetição. Prontos para programar?
Vamos aprender exatamente o que professor ?
Então pessoal, vejamos. Neste tópico vamos aprofundar nas estruturas de seleção simples, composta e de múltipla escolha. Nosso foco será aprender as estruturas de repetição e como realizar o controle de sua execução. Porém precisamos integrar o aprendizado das estruturas de seleção e repetição. Dessa forma vamos ampliar o entendimento da lógica na resolução dos problemas de programação. Aprender a programar é um processo e a cada dia vamos ampliando as nossas habilidades. As linguagens de programação são como idiomas. Podemos e devemos aprender vários, pois isso ampliará nossos horizontes, mas é necessário treinar, estudar e aplicar os idiomas (linguagens de programação) ou acabaremos esquecendo tudo que aprendemos. A melhor forma de consolidar o aprendizado de programação é executando, testando e modificando os algoritmos aprendidos! Não se esqueça disso!

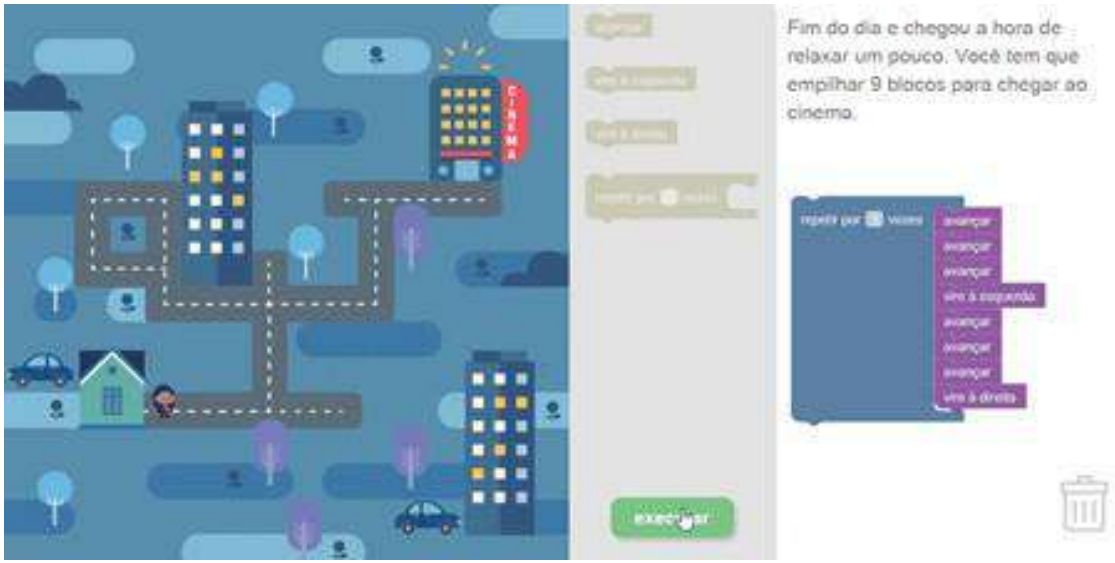
2.2 Estruturas de Repetição e Seleção

2.2.1 Estrutura de Controle de Dados para Repetição

A partir do conhecimento sobre algumas estruturas de controle de dados, torna-se possível a aprendizagem de uma das principais instruções utilizadas na programação de computadores: a estrutura de repetição. Ela é o mecanismo que facilita a resolução de problemas repetitivos. Normalmente, é empregada para a realização de tarefas que envolvem o processamento de muitas informações.

As estruturas de repetição são essenciais para execução dos mais diversos tipos de algoritmos.

Figura 8 - Estruturas de repetição



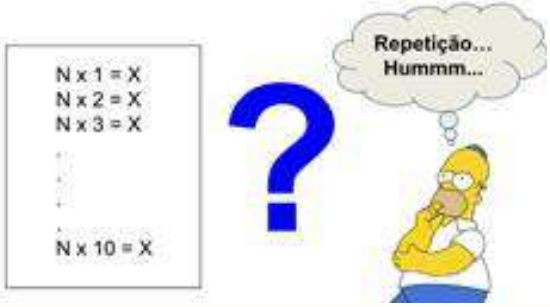
Fonte: Retirada da internet – Disponível em: <https://tab.uol.com.br/programacao/>. Acesso em: 01 de fev.2018.

Um processamento de valores pode acontecer sobre uma mesma lógica operacional, porém os dados envolvidos são diferentes e resultam em valores finais também diferentes, utilizado em algoritmos nos quais um determinado cálculo (processamento) deverá ser feito várias vezes, apenas com mudança de valores. Trata-se de um processo repetitivo.

Vamos observar o seguinte exemplo:

"Vamos criar um algoritmo que receba um número positivo e calcule a tabuada de 1 a 10 com o número informado".

Figura 9 - Exemplo - Construindo a tabuada




```
1  algoritmo "tabuada"
2  // Sintese
3  // Objetivo: calcular a tabuada de um número informado pelo usuário
4  // Entrada: um número
5  // Saida: tabuada de 1 até 10 do número informado
6  // Declarações
7  var
8  numero : inteiro
9  inicio
10 escreval("Digite o número inteiro desejado para tabuada: ")
11 leia(numero)
12 se(numero < 0) entao
13 escreva("Valor informado deve ser positivo.")
14 senao
15 escreval(numero:2, " x 1 = ", (numero * 1):2 )
16 escreval(numero:2, " x 2 = ", (numero * 2):2 )
17 escreval(numero:2, " x 3 = ", (numero * 3):2 )
18 escreval(numero:2, " x 4 = ", (numero * 4):2 )
19 escreval(numero:2, " x 5 = ", (numero * 5):2 )
20 escreval(numero:2, " x 6 = ", (numero * 6):2 )
21 escreval(numero:2, " x 7 = ", (numero * 7):2 )
22 escreval(numero:2, " x 8 = ", (numero * 8):2 )
23 escreval(numero:2, " x 9 = ", (numero * 9):2 )
24 escreval(numero:2, " x 10 = ", (numero * 10):2 )
25 fimse
26 fimalgoritmo
27
```

Fonte: Elaborada pelo autor.

Continua...

Figura 9 - Exemplo - Construindo a tabuada (continuação)

O exemplo desenvolvido tem um algoritmo que atende ao problema apresentado. Mas imagine que a nova necessidade de cálculo seja para geração de até 1.000 ou 1.000.000 de valores para o número escolhido da tabuada.
Não ficará apenas mais complexo. Ficará enorme ou muito extenso!

Para evitar esse tipo de extensão no corpo do algoritmo
(evitar a repetição muito grande de linhas), é possível a construção de algoritmos com instruções de repetição que permitem a repetição contínua de uma operação previamente identificada por quantas vezes forem necessárias. Tal repetição tem como base o resultado de um teste condicional.

Fonte: Elaborada pelo autor.

Repetição para... faça

A estrutura de repetição PARA executa um conjunto de instruções por uma quantidade de vezes conhecida (bem definida antes de seu início). Ela emprega uma variável de controle que recebe a indicação de um valor inicial e um valor final. Seu valor será incrementado pelo valor do passo a cada execução do bloco de comandos da estrutura.

Sintaxe Geral:

```
para <variável_controle> de <v_inicial> ate <v_final> passo <valor> faca
        <bloco de comandos>
fimpara
```

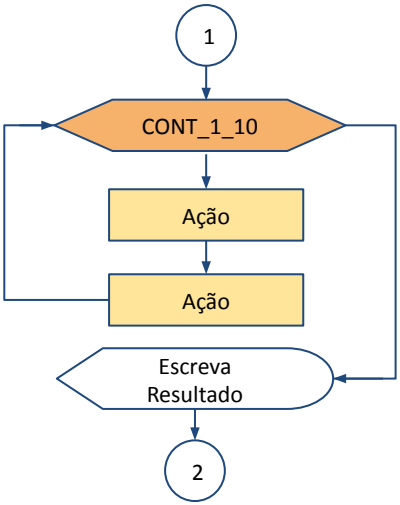
- <variável_controle> é uma variável do tipo inteiro, ou de um único caracter, que auxiliará na contagem da repetição iniciada com o valor inicial <v_inicial> até o valor final especificado em <v_final>.
- <valor> corresponde a um número inteiro de incremento ou decremento dado a variável definida em <variável_controle>, também conhecida como variável de controle da repetição.

Figura 10 - Exemplo de estrutura de repetição PARA

Escreva um algoritmo que escreva na tela os números de 1 até 10, utilizando a repetição para… faça…:

1 2 3 4 5 6 7 8 9 10

Fluxograma



O problema da tabuada, apresentado anteriormente, pode ser solucionado com o algoritmo que empregue a instrução de repetição **para… faça**, que o tornará mais eficiente e menor em quantidade de linhas (menor extensão do código).

Vamos refazer agora com a estrutura de repetição… para… faça


```
1  algoritmo "tabuada"
2  // Sintese
3  // Objetivo: calcular a tabuada de um número informado pelo usuário
4  // Entrada: um número
5  // Saida: tabuada de 1 até 10 do número informado
6  // Declarações
7  var
8  numero, contador : inteiro
9  inicio
10 escreval("Digite o número inteiro desejado para tabuada: ")
11 leia(numero)
12 se (numero < 0) entao
13    escreva("Valor informado deve ser positivo.")
14 senao
15    para contador de 1 ate 10 passo 1 faca
16       escreval(numero:2, " x ", contador:2, " = ", (numero * contador):2
17       )
18    fimpara
19 fimse
20 fimalgoritmo
```

Fonte: Elaborada pelo autor.

Repetição enquanto... faça

A estrutura de repetição ENQUANTO permite a execução de um conjunto de comandos pertencentes ao seu bloco de repetição (comandos) a partir de um teste condicional que resulta em verdadeiro. Enquanto este teste permanecer verdadeiro, a repetição é executada continuamente, sendo encerrada somente quando a condição for falsa (terminando o bloco de repetição).

A estrutura de repetição ENQUANTO obedece às seguintes operações:

- Primeiramente, efetua-se um teste lógico antes de iniciar o bloco de repetição.
- Caso o resultado desse teste lógico seja verdadeiro, todo bloco de repetição é executado até a instrução fimenquanto.
- Chegando a instrução fimenquanto, o foco da execução retorna para a instrução enquanto e o teste condicional é refeito (um teste a cada volta).
- Enquanto o teste resultar em verdadeiro, esta sequência lógica prossegue em execução, repetindo o processamento previsto em seu bloco de repetição.
- Quando seu teste resultar em falso, o bloco de repetição é saltado e a execução prossegue a partir da primeira instrução após o fimenquanto.

O controle desta repetição pode acontecer de duas formas:

1) controlada pelo próprio usuário do algoritmo; e
2) controlada de maneira automática, de acordo com a lógica ou o raciocínio implementado pelo programador.

Veja a seguir a sintaxe geral dessa estrutura.

Sintaxe Geral:

```
enquanto (<condição>) faca
    // também chamado de bloco de repetição
   <bloco de comandos>
fimenquanto
```

- <condição> é um teste condicional que resulta em um dado do tipo lógico, ou seja, tem resultado verdadeiro ou falso somente, sendo estes valores excludentes (ou é falso ou é verdadeiro, nunca podendo ser os dois ao mesmo tempo);

- <bloco de comandos> corresponde ao bloco de instruções que deve possuir todos os comandos a serem repetidas vezes executado pelo algoritmo, além das instruções que permitirão o controle desta repetição. Este bloco inicia com a palavra reservada enquanto e termina em fimenquanto.

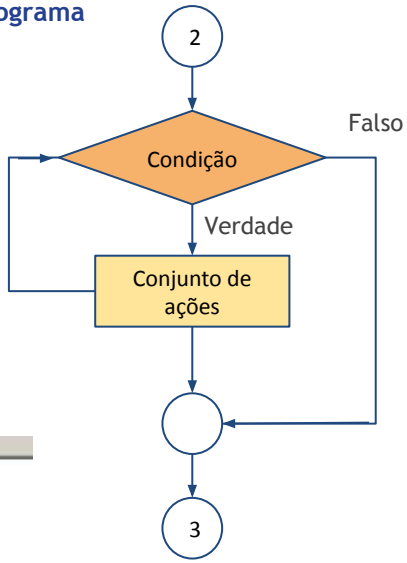
Figura 11 - Exemplo de estrutura de repetição ENQUANTO

Escreva um algoritmo que escreva na tela os números de 1 até 10, utilizando a repetição enquanto... faça...:

1 2 3 4 5 6 7 8 9 10

Repetição enquanto... faça...

Fluxograma




```
1  algoritmo "média da turma"
2  // Sintese
3  // Objetivo: calcular a média aritmética de uma turma de alunos
4  // Entrada: 50 notas individuais de cada aluno
5  // Saida: média aritmética de toda a turma
6  // Declarações
7  var
8  auxiliar : inteiro
9  nota, media : real
10 inicio
11 media<-0 // inicialização obrigatória da variável media
12 auxiliar<-1 // inicialização obrigatória da variável de controle auxiliar
13 enquanto (auxiliar <= 50) faca
14    escreval("Informe a nota do aluno: ")
15    leia(nota)
16    media<-media + nota // realiza a soma de cada nota informada
17    auxiliar<-auxiliar + 1 // conta a quantidade de notas informadas
18 fimenquanto
19 media<-media / 50 // calcula a média da turma com todas as notas
20 escreval("Média Aritmética da turma = ", media:3:1)
21 fimalgoritmo
```

Fonte: Elaborada pelo autor.

Ao se trabalhar com estruturas de repetição, devemos tomar muito cuidado com as variáveis envolvidas, especialmente com as variáveis de controle. Todo laço com variável de controle deve conter:

1) inicialização da variável de controle;
2) incremento (aumento do valor da variável de controle) ou decremento (diminuição do valor da variável de controle) da variável de controle; e
3) teste de valor da variável de controle (condição para finalizar o loop).

> O bloco de comandos da estrutura de repetição ENQUANTO só será executado se o teste da condição inicial for verdadeiro.
>
> "Enquanto for verdadeiro faça!!!"

Repetição repita... ate

A estrutura de repetição REPITA ATE possui características relevantes e diferenciadas na sua lógica de execução. Vejamos as principais características na relação abaixo:

- O bloco de comandos de repetição (instruções) está localizado entre as palavras reservadas repita e ate;
- O teste condicional para realizar a repetição sobre o bloco de instruções existente é executado somente no final do bloco;
- O bloco entre as duas palavras reservadas desta instrução (bloco de comandos) será executado, no mínimo uma vez, independente do resultado do seu teste condicional que só será realizado ao final de cada execução do próprio bloco.
- A repetição é executada novamente (pela segunda vez) somente se o teste condicional for falso, pois sendo ele verdadeiro, a repetição é encerrada.

Observe que a lógica de repetição desta instrução é diferente das outras duas (para... faca e enquanto... faca) estudadas anteriormente.

Sintaxe Geral:

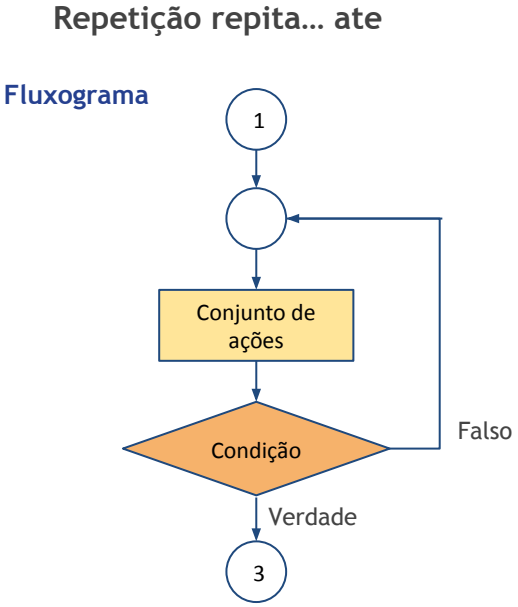
```
repita
   // também chamado de bloco de repetição
  <bloco de comandos>
 ate (<condição>)
```

- <condição> é um teste condicional que resulta em um dado do tipo lógico, ou seja, tem resultado somente de verdadeiro ou falso que só é realizado ao final da execução de todas as instruções existentes em seu bloco de repetição;

- <bloco de comandos> corresponde ao bloco de instruções que deve possuir todos os comandos a serem executados repetidamente no algoritmo, além das instruções que permitirão o controle desta repetição.

Figura 12 - Exemplo de estrutura de repetição REPITA ATE

Escreva um algoritmo para calcular a média aritmética de uma turma de 50 alunos utilizando a estrutura REPITA ATE. Ao final, informa o valor da média da turma.




```
1  algoritmo "média da turma"
2  // Sintese
3  // Objetivo: calcular a média aritmética de uma turma de alunos
4  // Entrada: 50 notas individuais de cada aluno
5  // Saida: média aritmética de toda a turma
6  // Declarações
7  var
8  auxiliar : inteiro
9  nota, soma : real
10 inicio
11 soma<-0 // inicialização obrigatória da variável soma
12 auxiliar<-1 // inicialização obrigatória da variável de controle auxiliar
13 repita
14    escreval("Informe a nota do aluno: ")
15    leia(nota)
16    soma<-soma + nota // realiza a soma de cada nota informada
17    auxiliar<-auxiliar + 1 // conta a quantidade de notas informadas
18 ate (auxiliar > 50)
19 escreval("Média Aritmética da turma = ", (soma / 50):3:1 )
20 fimalgoritmo
```

Fonte: Elaborada pelo autor.

O bloco de comandos da estrutura de repetição REPITA ATE sempre é executado um vez e somente depois disso é que o teste para a repetição é feito. Se for falso, repete novamente. Se for verdadeiro, termina a estrutura.

"Repita até ser verdadeiro!!!"

2.2.2 Formas de Controle das Estruturas de Repetição

Podemos controlar a forma de execução de uma estrutura de repetição. Esse controle pode ser automático ou por meio de interação com o usuário. Podemos ainda utilizar estruturas de controle dentro das estruturas de repetição, e por meio de comandos especiais interromper ou modificar a execução das estruturas de repetição.

Controlador Automático

Uma variável auxiliar conta quantas vezes será executado o conjunto de comandos (bloco de repetição), sem interferência direta do usuário sobre a quantidade de vezes. Este bloco de instruções sempre será repetido a quantidade de vezes prevista pelo desenvolvedor do algoritmo (programador).

A seguir, veja alguns exemplos de estrutura de controle automático.

Figura 13 - Exemplos de controle automático

enquanto... faca...

```
1  algoritmo "enquanto automático"
2  // Síntese
3  // Objetivo: confirmar a leitura de 5 pesos
4  // Entrada: 5 pesos
5  // Saída: mensagem confirmando a leitura de 5 pesos
6  // Declarações
7  var
8  contador : inteiro
9  peso : real
10 inicio
11 contador<-0
12 enquanto (contador < 5) faca
13    escreval("Digite o peso: ")
14    leia(peso)
15    contador<-contador + 1 // contabiliza quantidade de pesos lidos
16 fimenquanto
17 escreval("5 pesos foram informados.")
18 fimalgoritmo
```

repita... ate...

```
1  algoritmo "repita automático"
2  // Síntese
3  // Objetivo: confirmar a leitura de 5 pesos
4  // Entrada: 5 pesos
5  // Saída: mensagem confirmando a leitura de 5 pesos
6  // Declarações
7  var
8  contador : inteiro
9  peso : real
10 inicio
11 contador<-0
12 repita
13    escreval("Digite o peso: ")
14    leia(peso)
15    contador<-contador + 1 // contabiliza quantidade de pesos lidos
16 ate (contador = 5)
17 escreval("5 pesos foram informados.")
18 fimalgoritmo
```


Fonte: Elaborada pelo autor

Controlado pelo Usuário

O algoritmo recebe do usuário uma interação que é armazenada em uma variável normalmente como resposta a uma pergunta realizada ao usuário. Essa variável será usada para a verificação de controle (condição). O algoritmo sempre respeitará a solicitação do usuário, executando o bloco de repetição de acordo com a solicitação dele e respeitando sempre a lógica existente no algoritmo elaborado para solução de um problema.

Figura 14 - Exemplos de controle pelo usuário

enquanto...

faca...

```
algoritmo "enquanto controlado pelo usuário"
// Síntese
// Objetivo: confirma a quantidade de pesos lidos
// Entrada: pesos
// Saida: quantidade de pesos informados
// Declarações
var
contador : inteiro
peso : real
inicio
contador<-0
peso<-1
enquanto (peso > 0) faca
   escreval("Digite um peso válido ou negativo para encerrar: ")
   leia(peso)
   contador<-contador + 1 // contabiliza quantidade de pesos lidos
fimenquanto
escreval("Foram informados ", (contador - 1):2, " pesos. ")
fimalgoritmo
```

repita...

ate...

```
algoritmo "repita controlado pelo usuário"
// Síntese
// Objetivo: confirma a quantidade de pesos lidos
// Entrada: pesos
// Saida: quantidade de pesos informados
// Declarações
var
contador : inteiro
peso : real
inicio
contador<-0
repita
   escreval("Digite um peso válido ou negativo para encerrar: ")
   leia(peso)
   contador<-contador + 1 // contabiliza quantidade de pesos informados
ate (peso < 0)
escreval("Foram informados ", (contador - 1):2, " pesos. ")
fimalgoritmo
```

Vamos refletir?

CONTADOR

Um contador é uma variável de controle normalmente empregada para contar o número de repetições dentro de uma estrutura de repetição. Geralmente é empregada como variável de controle para indicar o momento de encerrar a execução de uma estrutura de repetição.

Exemplo:

```
var
 contador: inteiro       // declaração do contador
 contador <- 0   // inicializa o contador
 contador <- contador + 1      // incrementa o contador
```

Vamos refletir?

ACUMULADOR

O acumulador também é uma variável importante, mas a sua função é diferente do contador. O acumulador busca acumular o valor anterior ao valor "x" que está sendo adicionado. Normalmente, é usando dentro de estruturas de repetição, mas com a finalidade de "acumular" e não de contar. É importante lembrar que ele pode ser utilizado em conjunto com o contador dentro de uma estrutura de repetição. Em alguns casos pode ser a variável de controle para a saída (finalização) de uma estrutura de repetição.

Exemplo:

```
var
acumulador: inteiro // declaração do acumulador
x: inteiro // declaração de outra variável qualquer
acumulador <- 0 // inicializa o acumulador
// acumula na variável ACUMULADOR o valor anterior mais o valor de X
acumulador <- acumulador + x
```

Saiba mais!

Que tal dar uma espiada em um material de Lógica com VISUALg ?

Está ficando cada vez mais fácil. Para ajudar um pouco mais, vamos espiar um conjunto de vídeos para entender esses conceitos um pouco melhor?

Vamos dar uma espiada em alguns vídeos sobre Lógica de Programação disponíveis no YOUTUBE.

Não desista. Seja persistente! Teste e modifique os códigos!

Acesse:

https://www.youtube.com/watch?v=6-_leAMCi8M&list=PLlUjQffi3XKOc2OjC5aCekmxmhC5kSm70

Que maravilha! Cada vez mais conhecimentos e possibilidades! Não esqueça de voltar aos tópicos anteriores para revisar o que aprendeu e para buscar e integrar os códigos! Isso mesmo! Experimente, teste, modifique e aprenda! Essa é a fórmula mágica! A curiosidade e a prática! Neste tópico aprendemos a trabalhar com estruturas de controle de dados, particularmente para a repetição. Lembre-se da importância das variáveis de controle e que cada estrutura de repetição tem suas funcionalidades. É importante ter em mente que existe mais de uma solução para o mesmo problema. Podemos empregar diferentes estruturas para solucionar um mesmo problema. Neste tópico aprendemos as estruturas de repetição PARA, ENQUANTO e REPITA ATE. Aprendemos que uma estrutura pode ter controle automatizado ou depender do usuário. Foi possível também compreender os conceitos de variáveis que atuam como contador e acumulador. Aproveite para consultar o glossário da disciplina se tiver alguma dúvida de algum termo. Vamos partir para algumas questões de aprendizagem e para a revisão final do tópico. Não esqueça de olhar os sites e vídeos indicados que irão ampliar os seus horizontes de conhecimentos sobre o tema da unidade.

Vamos rever?

A partir do conhecimento sobre algumas estruturas de controle de dados, torna-se possível a aprendizagem de uma das principais instruções utilizadas na programação de computadores: a estrutura de repetição.

Ela é o mecanismo que facilita a resolução de problemas repetitivos. Normalmente, é empregada para a realização de tarefas que envolvem o processamento de muitas informações.

A estrutura de repetição PARA executa um conjunto de instruções por uma quantidade de vezes conhecida (bem definida antes de seu início). Ela emprega uma variável de controle que recebe a indicação de um valor inicial e um valor final. Seu valor será incrementado pelo valor do passo a cada execução do bloco de comandos da estrutura.

A estrutura de repetição ENQUANTO permite a execução de um conjunto de comandos pertencentes ao seu bloco de repetição (comandos) a partir de um teste condicional que resulta em verdadeiro. Enquanto este teste permanecer verdadeiro, a repetição é executada continuamente, sendo encerrada somente quando a condição for falsa (terminando o bloco de repetição).

A estrutura de repetição REPITA ATE possui características relevantes e diferenciadas na sua lógica de execução. Dentro do “repita ate”, o bloco entre as duas palavras reservadas desta instrução (bloco de comandos) será executado no mínimo uma vez, independente do resultado do seu teste condicional que só será realizado ao final de cada execução do próprio bloco. A repetição é executada novamente (pela segunda vez) somente se o teste condicional for falso, pois sendo ele verdadeiro a repetição é encerrada.

Vamos rever?

Podemos controlar a forma de execução de uma estrutura de repetição. Esse controle pode ser automático ou por meio de interação com o usuário. Podemos ainda utilizar estruturas de controle dentro das estruturas de repetição e, por meio de comandos especiais, interromper ou modificar a execução das estruturas de repetição. No CONTROLADOR AUTOMÁTICO, uma variável auxiliar conta quantas vezes será executado o conjunto de comandos (bloco de repetição), sem interferência direta do usuário sobre a quantidade de vezes. Já o CONTROLADO PELO USUÁRIO é uma variável auxiliar que conta quantas vezes será executado o conjunto de comandos (bloco de repetição), sem interferência direta do usuário sobre a quantidade de vezes. Este bloco de instruções sempre será repetido a quantidade de vezes prevista pelo desenvolvedor do algoritmo (programador).

Sites indicados

1)Dicas de Programação - Estrutura de repetição ENQUANTO
https://dicasdeprogramacao.com.br/estrutura-de-repeticao-enquanto/

2)Dicas de Programação - Estrutura de repetição PARA
https://dicasdeprogramacao.com.br/estrutura-de-repeticao-para/

3)Dicas de Programação - Estrutura de repetição REPITA-ATÉ
https://dicasdeprogramacao.com.br/estrutura-de-repeticao-repita-ate/

4)Dicas de Programação - Estrutura de seleção múltipla ESCOLHA-CASO
https://dicasdeprogramacao.com.br/estrutura-de-selecao-multipla-escolha-caso/

5)Dicas de Programação -Estrutura de decisão SE-ENTÃO-SENÃO
https://dicasdeprogramacao.com.br/estrutura-de-decisao-se-entao-senao/

Audiovisuais indicados

1)Lógica de Programação com VISUALg - Estrutura de Repetição - Para - 04
https://www.youtube.com/watch?v=lQjGDLSRUDo

2)Lógica de Programação com VISUALg Estrutura de Repetição - Enquanto - 05
https://www.youtube.com/watch?v=8-JWuzb-gIE

3)Lógica de Programação com VISUALg Estrutura de Repetição - Repita - 06
https://www.youtube.com/watch?v=cgfe08eg85o

4)Lógica de Programação com VISUALg - Estrutura de Seleção ou Decisão - 02
https://www.youtube.com/watch?v=mmHfui8nenw&index=2

Questões de autoaprendizagem

a) Vamos conferir se você entendeu este tópico. Faça um algoritmo que receba valores de peso válidos (positivos) e conte a quantidade de valores recebidos. Ao receber um valor de peso igual a zero, o sistema termina informando quantos pesos válidos foram recebidos. Use REPITA ATE.

b) Crie um algoritmo para exibir a tabuada de um número. O usuário informa o número e o sistema exibe a tabuada. Utilize REPITA ATE.

c) Crie um algoritmo para receber as notas de uma turma (sem saber a quantidade de alunos) até que uma nota negativa seja digitada (nota<0). Ao final, imprimir o valor da média aritmética da turma (média aritmética = soma das notas / quantidade de notas), sem considerar o último valor negativo digitado. Use a estrutura ENQUANTO.

d) Crie um algoritmo que identifique os números digitados 1, 5 ou 10 apenas. Caso o usuário digite outro valor qualquer, o sistema deve indicar “Número incorreto”. Use a estrutura CASO.

Gabarito das questões de autoaprendizagem

a)

```
algoritmo "analisa pesos"
// Síntese
// Objetivo: confirma a quantidade de pesos lidos
// Entrada: pesos
// Saida: quantidade de pesos informados
// Declarações
var
contador : inteiro
peso : real
inicio
contador<-0
peso<-1
repita
   escreval("Digite um peso de pessoa válido ou zero para encerrar: ")
   leia(peso)
   se (peso < 0) entao
      escreval("Peso inválido! Informe somente pesos positivos.")
   senao
      contador<-contador + 1 // só contabiliza pesos válidos
   fimse
ate (peso = 0)
escreval("Foram informados ", (contador - 1):2, " pesos válidos. ")
fimalgoritmo
```

Gabarito das questões de autoaprendizagem

b)

```
algoritmo "semnome"
// Função :
// Autor :
// Data : 19/1/2018
// Seção de Declarações
var
x,y: inteiro
inicio
// Seção de Comandos
escreva ("Informe o numero da tabuada:")
leia(x)
y<-0
repita
  escreval(x," x ",y," = ",x*y)
  y<-y+1
ate(y>10)
fimalgoritmo
```

```
Informe o numero da tabuada:5
 5 x  0 =  0
 5 x  1 =  5
 5 x  2 =  10
 5 x  3 =  15
 5 x  4 =  20
 5 x  5 =  25
 5 x  6 =  30
 5 x  7 =  35
 5 x  8 =  40
 5 x  9 =  45
 5 x  10 =  50

*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.
```

c)

```
1  algoritmo "notas da turma"
2  // Seção de Declarações
3  var
4  x:inteiro
5  nota,soma,media:real
6  inicio
7  // Seção de Comandos
8
9  x<-0
10 Escreva ("Digite a (",x+1,"ª) nota")
11 Leia(nota)
12 se (nota >=0) entao
13    soma<-soma+nota
14    x<-x+1
15    enquanto (nota >= 0)  faca
16       Escreva ("Digite a (",x+1,"ª) nota")
17       Leia(nota)
18       se (nota>=0) entao
19          x<-x+1
20          soma<-soma+nota
21       fimse
22    fimenquanto
23    media<-soma/x
24    Escreva ("Média:",media)
25 fimse
26 fimalgoritmo
27
```

Gabarito das questões de autoaprendizagem

d)

```
// Autor :
// Data : 10/9/2011
// Seção de Declarações
var
numero:inteiro

inicio
// Seção de Comandos
Escreval ("Opções disponiveis: (1, 5 ou 10)")
Escreva ("Entre com o valor escohido=>")
Leia (numero)

escolha (numero)
   caso (1)
     Escreva ("O valor digitado foi -> 1")
   caso (5)
     Escreva ("O valor digitado foi -> 5")
   caso (10)
     Escreva ("O valor digitado foi -> 10")
   outrocaso
     Escreval ("Número incorreto...")
     Escreval
     Escreval
fimescolha

fimalgoritmo
```


Acorda gente! Chegou o momento de mais programação! Agora é hora de começar a falar sobre a importância das estruturas de dados e como elas se relacionam com nossos programas. O que acham?
Estruturas de dados?
Sim pessoal. Neste tópico vamos conhecer as estruturas de dados homogêneas e heterogêneas e como elas se organizam em Lógica de Programação. Vamos compreender a relação dessas estruturas com nossas variáveis, além de entender e aplicar os conceitos de vetor, matriz e registro (ou estrutura). Veremos como acessar e armazenar dados nessas estruturas diferenciadas por meio de seus índices. Será um salto que nos levará a outro patamar de programação. Estudaremos neste tópico, nessas estruturas rudimentares a base da construção de bancos de dados. Será uma viagem e tanto. Vamos começar! Preparem os índices e vamos acessar os dados !

Unidade 3 Estruturas de Dados e Modularização

Ao término desta unidade esperamos atingir os seguintes objetivos:

- Compreender e manipular estruturas de dados básicas: vetores e matrizes;
- Codificar algoritmos na linguagem de programação;
- Empregar funções e procedimentos para o reuso de código fonte;
- Produzir programas legíveis, eficientes e corretos.

Vamos começar!

Preparem os índices e vamos acessar os dados!

"Não é a linguagem de programação que define o programador, mas sim sua lógica."

David Ribeiro Guilherme

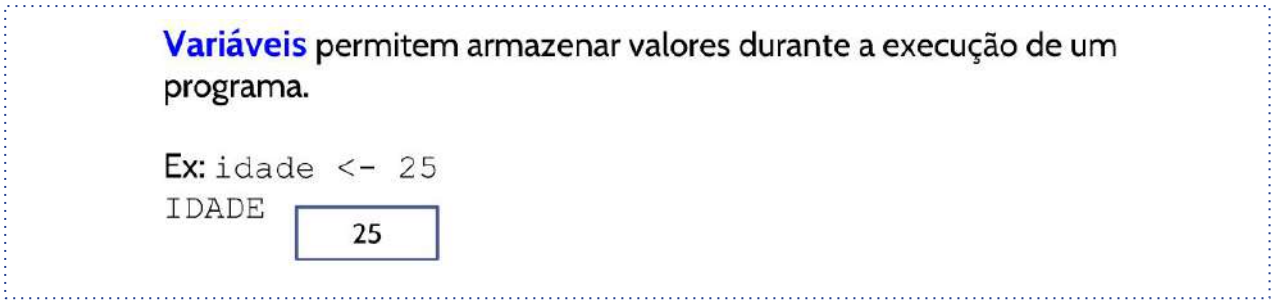
Mãos ao teclado!

3.1 Estruturas de Dados: Vetor e Matriz

3.1.1 Estrutura de Dados Composta Homogênea

Como já aprendemos, as variáveis são utilizadas para armazenar dados na memória. A partir de seus valores, o nosso código (programa) realiza ações, toma decisões e emite resultados. Mas, uma variável pode armazenar apenas um único valor de cada vez. Bem, pelo menos foi o que aprendemos até agora. Veja:

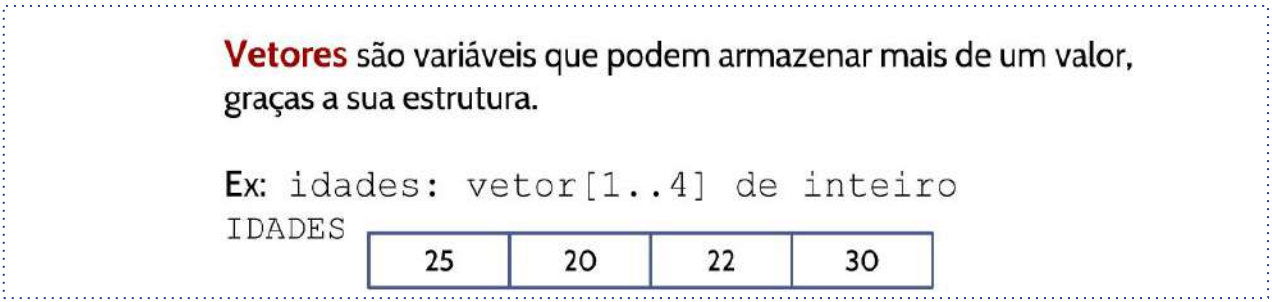
Figura 1 - O conceito de variável



Fonte: Elaborada pelo autor.

É preciso relembrar que toda variável é um apelido (ou simples nome) que aponta para um endereço de memória em que o dado está armazenado. Ocorre que as linguagens de programação implementam outras formas de armazenamento como os VETORES e as MATRIZES que veremos a seguir. Para começar, vamos observar o conceito de um vetor:

Figura 2 - O conceito de vetor



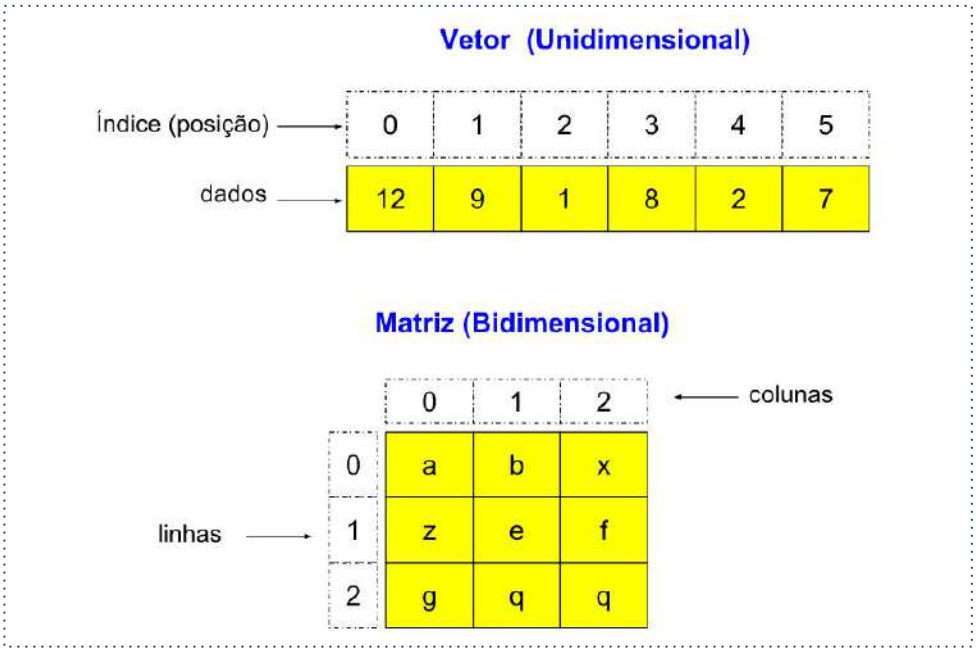
Fonte: Elaborada pelo autor

Observando a imagem acima, podemos perceber que o VETOR é um conjunto de variáveis que armazenam diversos valores. Cada valor armazenado possui uma localização que será acessada por um índice. O vetor, assim como a variável, também possui um apelido (ou simples nome) que aponta para um endereço de memória onde os dados estão armazenados.

As estruturas de dados consistem em organizações lógicas sobre o armazenamento e manipulação dos dados que serão necessários ao algoritmo e ao programa resultante de tal representação (português estruturado ou fluxograma).

As principais estruturas a serem manipuladas nos algoritmos são classificadas em homogêneas (de um mesmo tipo de dado) e heterogêneas (tipos de dados diferentes). Por meio do uso mais correto dessas estruturas de dados, os algoritmos e os programas se tornam altamente eficientes sobre os recursos computacionais disponíveis.

Figura 3 - Representações de Vetor e Matriz

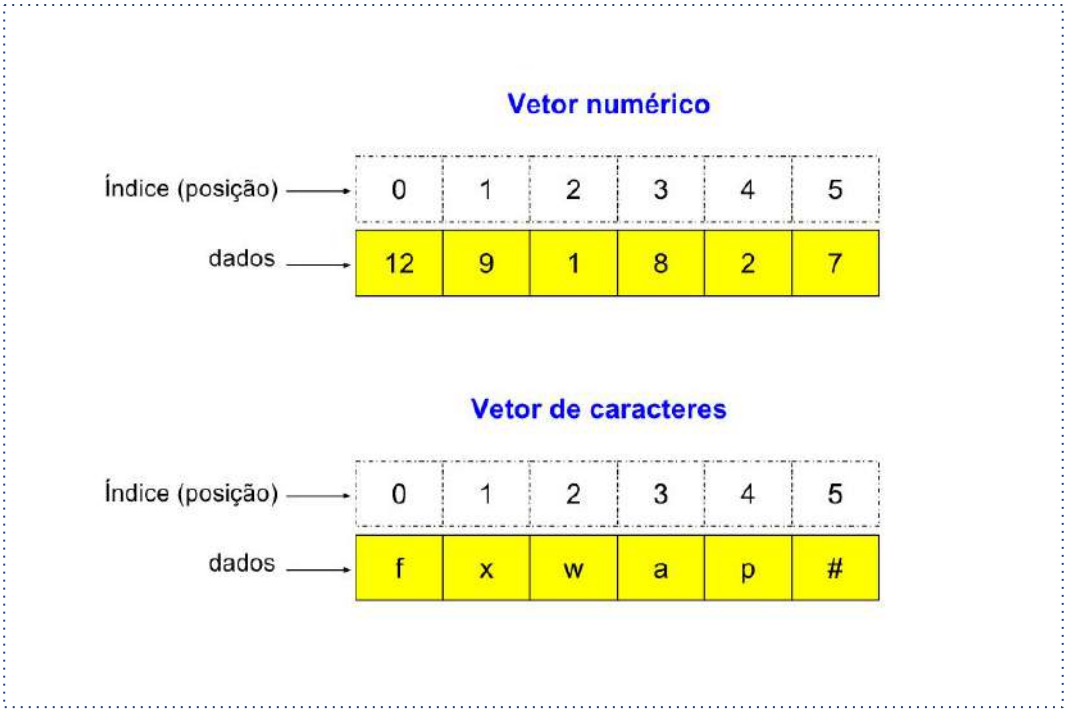


Fonte: Elaborada pelo autor.

A MATRIZ, assim como o VETOR e a VARIÁVEL, também possui um apelido (ou simples nome) que aponta para um endereço de memória onde os dados estão armazenados. Seus dados estão acessíveis por meio de dois índices (bidimensional - linhas e colunas), conforme demonstra a Figura 3.

Tanto o VETOR quanto a MATRIZ, particularmente em Português Estruturado (utilizando VISUALg), possuem a característica de serem estruturas homogêneas, pois irão armazenar dados de um mesmo tipo. Veja os exemplos desse aspecto nas Figuras 3 e 4.

Figura 4 - Vetores de números e caracteres



Fonte: Elaborada pelo autor.

Vamos refletir?

Vamos pensar em como ler e armazenar 5 notas de um aluno. Como ficaria esse algoritmo?

Bem, a reflexão mais rápida é criar 5 variáveis e armazenar os valores das 5 notas.

Veja o código abaixo. Observe como os valores estão sendo lidos e armazenados.

```
1  algoritmo "soma notas"
2  // Sintese
3  // Objetivo: armazenar 5 notas de 5 alunos diferentes e apresentá-las
4  // Entrada: 5 notas
5  // Saida: soma das notas
6  // Declarações
7  var
8  nota1, nota2, nota3, nota4, nota5, total : real
9  inicio
10 escreva("Informe a primeira nota: ")
11 leia(nota1)
12 escreva("Informe a segunda nota: ")
13 leia(nota2)
14 escreva("Informe a terceira nota: ")
15 leia(nota3)
16 escreva("Informe a quarta nota: ")
17 leia(nota4)
18 escreva("Informe a quinta nota: ")
19 leia(nota5)
20 total<-nota1 + nota2 + nota3 + nota4 + nota5
21 escreval("Total das notas do grupo = ", total:3:2)
22 fimalgoritmo
23
```

Fonte: Código elaborado pelo autor.

Vamos refletir?

Do exemplo apresentado, podemos tirar algumas reflexões:

Uma instrução de repetição não melhoraria muito esta lógica, pois os cinco valores precisarão estar disponíveis independentes dos outros valores informados em variáveis distintas. Imagine então que as leituras de todas as notas dos alunos do seu curso tivessem de ser lidas por seu algoritmo. Como você faria? Como saber a quantidade de alunos do próximo semestre para armazenar suas respectivas notas?

Em uma situação como essa, as estruturas de dados são fundamentais para o eficiente processamento dos dados. Essas estruturas fundamentais podem ser de dois tipos básicos:

- Homogênea (vetor ou matriz);
- Heterogênea (registro ou estrutura).

3.1.2 Estrutura de Dados Composta Homogênea Unidimensional (vetor)

Estrutura de Dados Composta Homogênea Unidimensional (vetor) consiste em uma única estrutura (variável ou constante) com capacidade de armazenamento de mais de um valor (por isso composta) com um mesmo tipo de dado (por isso homogênea).

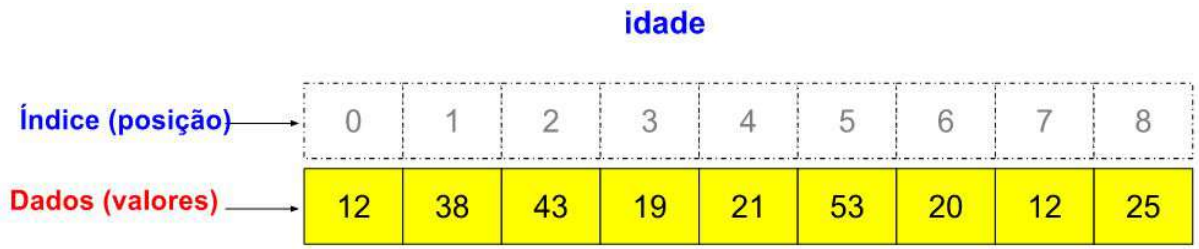
Suas principais características são:

- contém vários valores, porém com quantidade definida;
- todos os valores são do mesmo tipo de dado (homogêneo);
- possui um único nome, ou seja, um único identificador para vários valores a serem armazenados na memória; e
- cada um de seus valores é acessível independentemente, de acordo com o seu índice ou sua posição da estrutura de dados.

Vejamos um exemplo:

Suponha a existência de uma estrutura que armazene a idade de 12 pessoas, sendo o identificador desta estrutura a expressão idades. Observe a imagem do vetor "idade" com os dados na Figura 5.

Figura 5 - Vetor idade



Fonte: Elaborada pelo autor.

Os índices correspondem às posições de identificação dos valores armazenados de forma independente, apesar de possuírem acesso por meio do mesmo identificador. Por meio de seus valores inteiros é que se torna possível manipular, especificamente, um ou mais valores armazenados nesta estrutura de dados, conforme seja necessária a lógica operacional deste algoritmo.

Os valores correspondem aos conteúdos informados pelo usuário, ou seja, são as idades solicitadas ao usuário que às informou por meio da instrução leia. Esses valores são armazenados, independentemente, na estrutura de dados indicada nesta instrução de entrada de dados.

Atenção!!!! A primeira posição de armazenamento tem seu índice igual a zero, sendo seu sexto elemento de idade armazenado no índice ou posição 5 da estrutura.

Logo, o VETOR é uma estrutura unidimensional e possibilita a criação de variáveis compostas unidimensionais com tamanho definido na declaração do vetor, possuindo as seguintes características:

- Composta: porque podem armazenar vários valores.
- Unidimensional: porque só possui variação em uma dimensão.
- Homogênea: porque armazena um único tipo de dado.

A sintaxe geral para criação de uma estrutura de dados composta unidimensional, também chamada de vetor, em um algoritmo em português estruturado, é apresentada na forma geral a seguir:

Declaração da estrutura de dados composta homogênea unidimensional (VETOR):

<identificador> : vetor [**<inicial>..<final>**] de <tipo de dado>

Exemplo: nota : vetor [**0..9**] de numerico

Onde:

- <tipo de dado> é o tipo de dado que será armazenado na estrutura;
- <identificador> é o nome atribuído à estrutura de dados (vetor); e
- **<inicial>** e **<final>** correspondem respectivamente aos valores numéricos inteiros de início e fim dos índices do vetor, sendo seu intervalo a quantidade exata de elementos do vetor ou seu tamanho.

É possível perceber certa semelhança com a declaração de uma variável convencional. Houve apenas uma mudança que é o acréscimo da indicação de "vetor" (palavra reservada) com a definição do tamanho do vetor.

Vejamos, agora, como seria a declaração de um vetor chamado **notas** que pode armazenar até 20 números reais independentes.

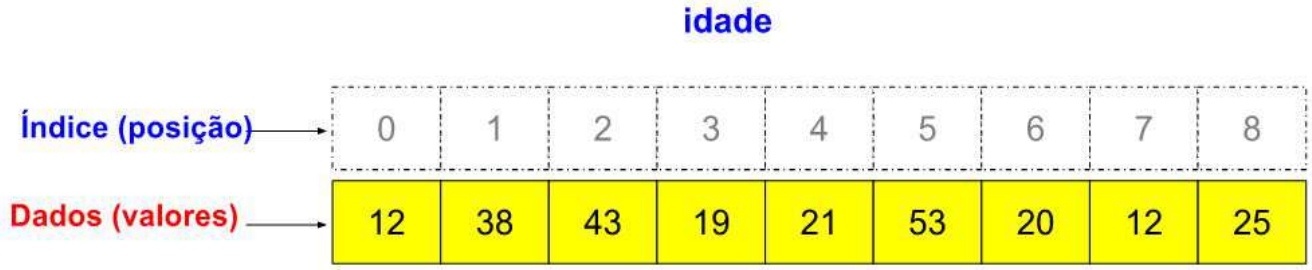
Exemplo:

notas : **vetor** [0 ..19] de **real**

Essa quantidade de valores (20) é a capacidade de armazenamento declarada no bloco de declarações do algoritmo em português estruturado, pois de zero a dezenove existem vinte posições, ou seja, a capacidade do vetor **notas** é de 20 valores **reais**. O primeiro guardado na posição zero, enquanto o último, na posição dezenove.

O acesso independente a um elemento do vetor acontece por meio da especificação de seu nome e de seu índice ou posição entre colchetes. No vetor “idade”, da Figura 6, temos que o valor armazenado em idades[2] é 43. Já para idades[8] o valor armazenado é 25 e assim por diante. Podemos acessar diretamente cada valor da variável idade, do tipo vetor, apenas indicando o índice da posição que armazena o valor que queremos acessar. Observe a Figura 6 que demonstra o acesso a outros valores do vetor “idade”.

Figura 6 - Acesso aos dados do vetor “idade”



idade[3] = 19
idade[0] = 12 (primeira posição do vetor tem índice zero)
O vetor “idade” tem apenas uma dimensão, pois tem apenas um índice.

Fonte: Elaborada pelo autor.

Propriedades dos vetores:

- O tamanho deve ser ajustado na definição do vetor.
- Todos os valores de um vetor pertencem ao mesmo tipo.

```
notas: vetor[0..3] de inteiro
nome, telefone: vetor[0..3] de literal
```

- Toda atribuição ou acesso a dados do vetor deve utilizar um índice inteiro ou valor (veja a Figura 7):

```
Valor: vetor[0..2] de real
idx <- 0
 Valor[1] <- 2.0      // (atribuição de valor para o vetor Valor na posição [1])
 Valor[idx] <- 1.0 // (atribuição de valor para o vetor Valor na posição [idx])
```

Figura 7 - Acesso aos dados do vetor "valor"

Vetor " Valor " **(Tipo de dados:** real)

Índice	0	1	2
Valor	1.0	2.0	0

Fonte: Elaborada pelo autor.

A utilização de vetores permite criar uma área de memória indexada. Podemos utilizar diversas variáveis para armazenar as notas de um aluno. Veja o exemplo abaixo:

```
not1,not2,not3,not4 : inteiro
```

Com vetores, podemos utilizar apenas uma variável e acessar os diversos valores por meio de um identificador (índice). Veja o exemplo abaixo:

```
Notas: vetor[1..4] de inteiro
```

Vamos observar os dois exemplos seguintes, nas Figuras 8 e 9, sobre a aplicação de vetores. Implemente e teste os dois exemplos no VISUALg. Execute e entre com valores. Observe os resultados.

Figura 8 - Algoritmo do vetor de idades para implementação no VISUALg

```
algoritmo "manipulação de várias idades no vetor"
// Sintese
// Objetivo: armazenar 12 idades e manipula-las convenientemente
// Entrada: 12 idades
// Saida: todos os 12 valores finais do vetor
// Declarações
var
auxiliar : inteiro
idades : vetor [0..11] de inteiro
inicio
para auxiliar de 0 ate 11 passo 1 faca
   escreva("Informe a ", (auxiliar+1):2, "ª idade: ") // mensagem orientadora
   leia(idades[auxiliar]) // suponha a leitura dos valores mostrados acima
fimpara
auxiliar<-4
idades[1]<-idades[auxiliar]
idades[auxiliar]<-idades[2 * auxiliar] + 5
limpatela // Instrução que limpa toda tela de execução do algoritmo
escreval("Elementos do Vetor Final") // mostra todos elementos do vetor
para auxiliar de 0 ate 11 passo 1 faca
   escreval("idades[", auxiliar:2, "] = ", idades[auxiliar]:4)
fimpara
fimalgoritmo
```

Fonte: Elaborada pelo autor.

Figura 9 - Algoritmo de média aritmética com vetor para implementação no VISUALg

```
algoritmo "média aritmética com vetor"
// Objetivo: armazenar as notas individuais de cada aluno em uma turma e
// apresentar a média aritmética desta mesma turma
var
auxiliar : inteiro
soma : real
notas : vetor [0..29] de real // vetor notas com capacidade de 30 reais
inicio
soma<-0 // atribuição inicial obrigatória para soma
para auxiliar de 0 ate 29 passo 1 faca
   escreva("Digite a nota (0 => 10) do ", (auxiliar+1):2, "º aluno: ")
   // Avaliação de nota válida (maior ou igual a 0 e menor ou igual a 10)
   repita
      leia(notas[auxiliar])
      se ((notas[auxiliar] < 0) ou (notas[auxiliar] > 10)) entao
         escreval("Nota inválida! Digite novamente (0 => 10): ")
      fimse
   ate((notas[auxiliar] >= 0) e (notas[auxiliar] <= 10))
fimpara
limpatela
// Apresentação dos resultados finais solicitados
escreval("Notas por aluno")
escreval
para auxiliar de 0 ate 29 passo 1 faca
   escreval((auxiliar + 1):2, "º Aluno =", notas[auxiliar]:3:1)
   soma<-soma + notas[auxiliar]
fimpara
escreval
escreva("Média Aritmética da turma =", (soma / 30):4:2)
fimalgoritmo
```

Fonte: Elaborada pelo autor.

Perceba que nos dois exemplos, empregamos estruturas de repetição. No exemplo 2 (Figura 9), temos a integração de estruturas de repetição e seleção. Observe que o tratamento do vetor é feito como uma variável normal, sempre indicando o índice do valor (posição) que queremos manipular.

3.1.3 Estrutura de Dados Composta Homogênea do Tipo Caracter (string)

Quando se declara uma variável do tipo caracter, para armazenar um nome, por exemplo, está se criando na verdade um vetor do tipo de dado caracter, afinal um nome consiste de um conjunto de letras (ou símbolos alfabéticos) e não simplesmente de um único caracter. Vejamos um exemplo da declaração de um vetor de caracteres no VISUALg:

```
var
 // vetor nome com capacidade de 10 caracteres
 nome : vetor [0 ..9 ] de caracter
```

Vamos observar os dois exemplos seguintes, nas Figuras 10 e 11, sobre a aplicação de vetores com caracteres. Implemente e teste os dois exemplos no VISUALg. Execute e entre com valores. Observe os resultados.

Figura 10 - Algoritmo de manipulação de um nome com vetor de caracteres

```
1  algoritmo "manipulação de um nome"
2  // Síntese
3  // Objetivo: armazenar o nome de uma pessoa e apresentar somente a
4  //primeira letra deste nome
5  var
6  auxiliar : inteiro
7  nome : vetor [0..9] de caracter
8  inicio
9  auxiliar<-0
10 escreval("Informe um nome com até 9 caracteres e termine com ponto final: ")
11 repita
12    leia(nome[auxiliar])
13    auxiliar<-auxiliar + 1
14 ate ((nome[auxiliar - 1] = ".") ou (auxiliar = 10))
15 escreval ("-----------------------------------------------------")
16 escreva("A primeira letra do nome informado = ", nome[0])
17 fimalgoritmo
```

```
Informe um nome com até 9 caracteres e termine com ponto final:
R
o
n
a
l
d
.
-----------------------------------------------------
A primeira letra do nome informado = R
*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.
```

Fonte: Elaborada pelo autor.

Figura 11 - Algoritmo de manipulação senha com vetor de caracteres

```
algoritmo "validar senha do usuário"
var
auxiliar : inteiro
igual : logico
senha : vetor [0..3] de caracter // vetor senha cadastrada
senhaUsuario : vetor [0..3] de caracter // senha informada pelo usuário
inicio
igual<-verdadeiro // tipo de dado lógico possui verdadeiro ou falso
auxiliar<-0
// registro da senha CUBO pelo programador deste algoritmo
senha[0]<-"C"
senha[1]<-"U"
senha[2]<-"B"
senha[3]<-"O"
escreva("Digite um caracter da senha por vez, pressionando <ENTER>:")
repita
   leia(senhaUsuario[auxiliar]) // digite <ENTER> entre cada caracter informado
   auxiliar<-auxiliar + 1
ate ((senhaUsuario[auxiliar - 1] = ".") ou (auxiliar = 4))
para auxiliar de 0 ate 3 passo 1 faca
   se (senha[auxiliar] <> senhaUsuario[auxiliar]) entao
      igual<-falso // tipo de dado lógico
   fimse
fimpara
se (igual = verdadeiro) entao
   escreva("Senha correta!")
senao
   escreva("Senha incorreta!")
fimse
fimalgoritmo
```

Fonte: Elaborada pelo autor.

3.1.4 Matriz

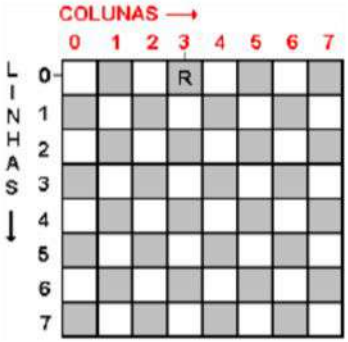
Uma matriz de dados consiste em uma estrutura de dados composta homogênea com variação em mais de uma dimensão, ou seja, sua organização lógica não está definida sobre uma única posição, mas sobre a quantidade de dimensões definidas em sua declaração.

Observe o tabuleiro de xadrez descrito na Figura 12 logo abaixo.

Figura 12 - Tabuleiro de xadrez - representação de uma matriz

Agora responda:

Qual a posição que o rei (R) de seu adversário estaria no início do jogo?

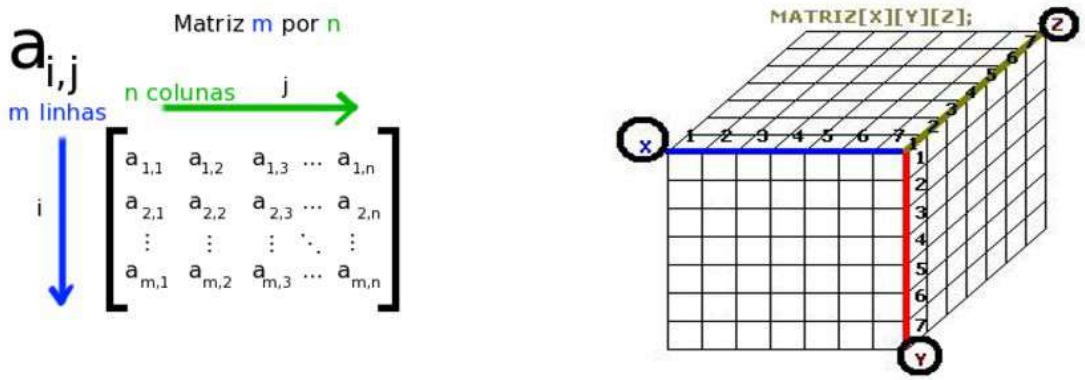


Usando uma notação de linha e coluna podemos indicar que o REI está na posição [0,3] ou seja linha 0 e coluna 3.

Fonte: Elaborada pelo autor.

É possível uma grande variação de dimensões na criação de um vetor. No entanto, essa criação deve ser coerente com o problema computacional que estará envolvendo este vetor. A limitação no número de dimensões de um vetor está apenas na quantidade de memória disponível no computador utilizado. Contudo, a capacidade de representação e entendimento de uma estrutura com muitas dimensões é limitada para o ser humano. O mais comum é utilizarmos o vetor com até 3 dimensões, ou seja, um cubo.

Figura 13 - Vetor de duas ou mais dimensões



Fonte: Elaborada pelo autor.

A forma de declaração de uma matriz bidimensional em português estruturado é definida abaixo:

Sintaxe Geral:

<identificador> : **vetor** [<inicial>..<final> , <inicial>..<final>] de <tipo de dado>

Onde:

- <tipo de dado> é o tipo de dado que será armazenado na estrutura;
- <identificador> é o nome atribuído à estrutura de dados (matriz);
- <inicial> e <final> correspondem, respectivamente, aos valores numéricos inteiros de início e fim dos índices da primeira dimensão da matriz, sendo seu intervalo a quantidade exata de elementos da primeira dimensão da matriz;
- <inicial> e <final> correspondem aos valores numéricos inteiros respectivos ao início e fim dos índices da segunda dimensão da matriz, sendo seu intervalo a quantidade exata de elementos da segunda dimensão da matriz.

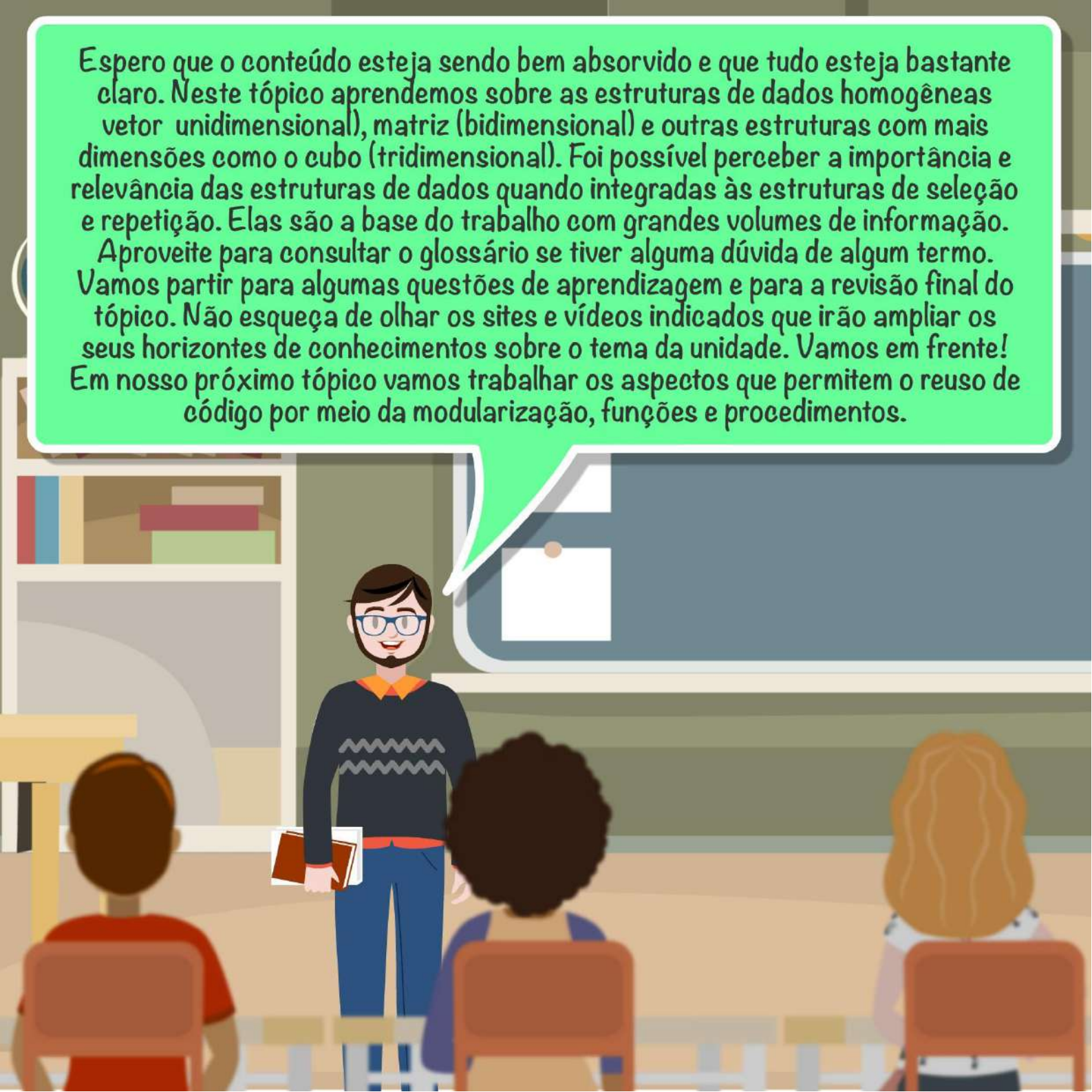
Exemplo: tabuleiro : vetor [0..7 , 0..7] de inteiro

Saiba mais!

Você sabe a diferença entre Variáveis, Vetores e Matrizes?

Que tal tirar as possíveis dúvidas sobre essas estruturas de dados que vimos neste tópico? Numa única videoaula de 5 minutos, você irá aprender os conceitos de variáveis, vetores e matrizes e entenderá a diferença entre cada um.

Aproveite a explicação disponível no YOUTUBE gravada pelo Professor Márcio em seu canal. Acesse: https://www.youtube.com/watch?v=dUTt4lHe5d4


Espero que o conteúdo esteja sendo bem absorvido e que tudo esteja bastante claro. Neste tópico aprendemos sobre as estruturas de dados homogêneas vetor unidimensional), matriz (bidimensional) e outras estruturas com mais dimensões como o cubo (tridimensional). Foi possível perceber a importância e relevância das estruturas de dados quando integradas às estruturas de seleção e repetição. Elas são a base do trabalho com grandes volumes de informação. Aproveite para consultar o glossário se tiver alguma dúvida de algum termo. Vamos partir para algumas questões de aprendizagem e para a revisão final do tópico. Não esqueça de olhar os sites e vídeos indicados que irão ampliar os seus horizontes de conhecimentos sobre o tema da unidade. Vamos em frente! Em nosso próximo tópico vamos trabalhar os aspectos que permitem o reuso de código por meio da modularização, funções e procedimentos.

Vamos rever?

As principais estruturas a serem manipuladas nos algoritmos são classificadas em homogêneas (de um mesmo tipo de dado) e heterogêneas (tipos de dados diferentes). Por meio do uso mais correto dessas estruturas de dados, os algoritmos e os programas se tornam altamente eficientes sobre os recursos computacionais disponíveis.

O VETOR é um conjunto de variáveis que armazenam diversos valores. Cada valor armazenado possui uma localização que será acessada por um índice. O vetor, assim como a variável, também possui um apelido (ou simples nome) que aponta para um endereço de memória onde os dados estão armazenados.

A MATRIZ, assim como o VETOR e a VARIÁVEL, também possui um apelido (ou simples nome) que aponta para um endereço de memória onde os dados estão armazenados. Seus dados estão acessíveis por meio de dois índices (bidimensional - linhas e colunas).

É possível uma grande variação de dimensões na criação de um vetor. No entanto, essa criação deve ser coerente com o problema computacional que estará envolvendo este vetor. A limitação no número de dimensões de um vetor está apenas na quantidade de memória disponível no computador utilizado. Contudo, a capacidade de representação e entendimento de uma estrutura com muitas dimensões é limitada para o ser humano. O mais comum é utilizarmos o vetor com até 3 dimensões, ou seja um cubo.

Sites indicados

1) Dicas de Programação - O que são Vetores e Matrizes (arrays)
https://dicasdeprogramacao.com.br/o-que-sao-vetores-e-matrizes-arrays/

2) Exercícios de matriz usando o VISUALg
https://fdocumentos.tips/document/adaobragacombr-exercicios-de-matriz-usando-o-visualg.html?page=1

3) Exercícios de vetores
https://ooffoo.wordpress.com/portugol-ide/exercicios-vetores/

Audiovisuais indicados

1) Vetores - Curso de Algoritmos #14 - Gustavo Guanabara
https://www.youtube.com/watch?v=j9473xQ39vY

2) Lógica de Programação - Ordenação de Vetores (Arrays)
https://www.youtube.com/watch?v=FQXDOVbdp0Y

3) Matrizes - Curso de Algoritmos #15 - Gustavo Guanabara
https://www.youtube.com/watch?v=hkE9WrjpAAk

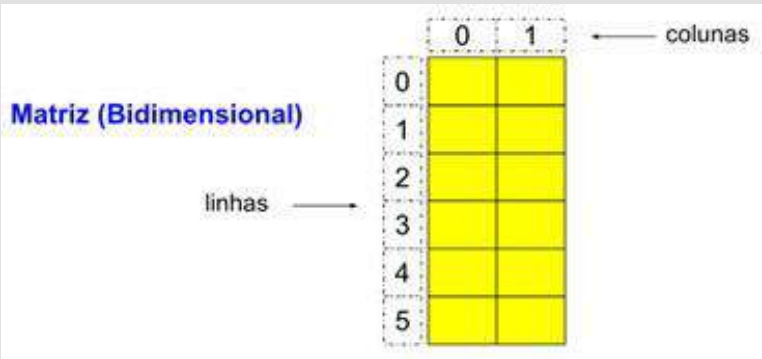
4) VISUALg - Vetores e Matrizes
https://www.youtube.com/watch?v=2TMf6MqSbOk

Questões de autoaprendizagem

a) Crie um algoritmo que efetue a leitura de 10 elementos inteiros e armazene-os em um vetor Teste1. A partir do vetor Teste1 construa um segundo vetor Teste2 do mesmo tipo, observando as seguintes regras de formatação: se o valor do índice for par, o valor do elemento deverá ser multiplicado por 5; se for ímpar, deverá ser somado com 5. Ao final, mostrar o conteúdo dos dois vetores.

b) O usuário irá digitar 10 elementos para um vetor inteiro. Faça um algoritmo que determine qual o maior e o menor elemento dentro desse vetor.

c) Dada uma matriz de 6 linhas e 2 colunas de inteiros, calcular e exibir a média geométrica dos valores de cada uma das linhas. A média geométrica é calculada pela seguinte expressão: SQRT (X1 * X2), que representa a raiz quadrada do resultado da multiplicação dos elementos da coluna 0 (X1) pelos elementos da coluna 1 (X2).



d) O usuário irá digitar os valores de uma matriz com 5 x 5 elementos do tipo inteiro. Faça um algoritmo que determine qual o maior e o menor elemento dentro dessa matriz.


Pronto para um novo desafio? É chegada a hora de aprender a economizar linhas de código com funções e procedimentos. Vamos em frente? As funções e os procedimentos facilitarão o entendimento do código fonte e irão permitir agrupar partes do algoritmo que tenha uma finalidade específica, ou seja uma tarefa peculiar que será realizada inúmeras vezes.
Certo, então com isso poderemos até trocar funcionalidades e partes de código entre os desenvolvedores (programadores) ?
Sim, isso mesmo. Perfeito! Essa proposta da reutilização, da divisão do código (em subprogramas) e da especialização de partes do programa que está sendo desenvolvido é a chave para o uso de funções. Tarefas que são executadas repetidamente durante a construção de um algoritmo podem indicar a necessidade ou a possibilidade de criação de uma função. O processo de manutenção do código também é facilitado com a utilização de funções e procedimentos. Mãos ao teclado!

3.2 Modularização

3.2.1 Introdução

A modularização é uma característica muito importante no desenvolvimento de programas. Ela é um método utilizado para facilitar a construção de grandes algoritmos, através de sua divisão em pequenas etapas, que são os subprogramas. É a aplicação direta do jargão "Dividir para conquistar".

Um "Subprograma" é um programa que auxilia o programa principal através da realização de uma determinada subtarefa. Os subprogramas são chamados dentro do corpo do programa principal como se fossem comandos. É conveniente utilizá-los quando uma determinada tarefa é efetuada em diversos lugares no mesmo algoritmo. Ao invés de escrever-se um trecho diversas vezes, escreve-se um subprograma e este pode ser invocado (executado) diversas vezes.

Existem dois tipos de subprogramas:

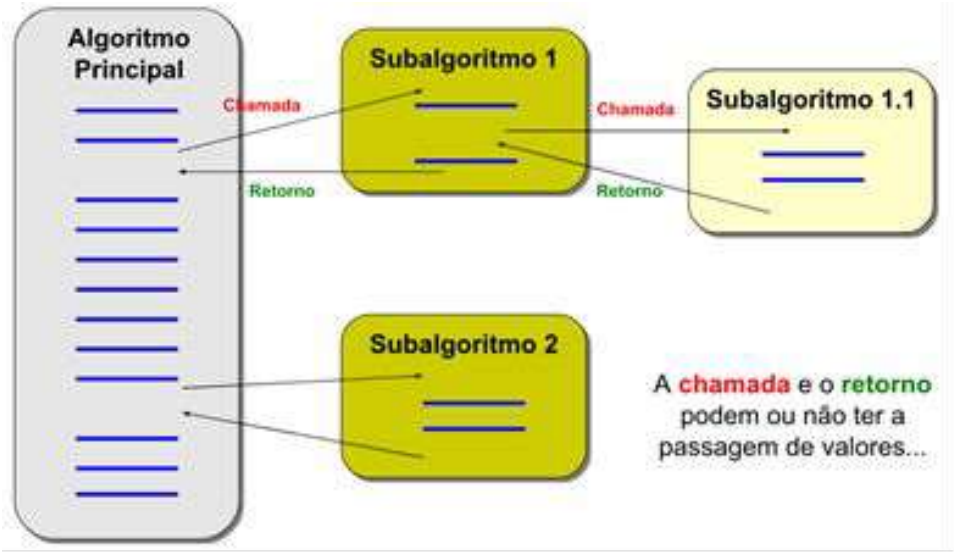
- procedimentos (sub-rotina): não retorna nenhum valor ao término de sua execução.
- funções: retorna um valor ou uma informação quando termina a execução da função.

É possível perceber que a diferença entre procedimentos e função está apenas em retornar ou não um valor ao término de sua execução.

A chamada de um subprograma ocorre quando invocamos o trecho de código pelo seu nome (identificador). Um algoritmo pode chamar um subprograma e, este por sua vez, poderá chamar outro e assim sucessivamente, transferindo o ponto de execução do algoritmo para o subprograma chamado. Ao término da execução do subprograma, o ponto de execução retorna, logo após o ponto inicial da chamada, conforme descrito na Figura 15.

Tanto a chamada quanto o término da execução de um subprograma podem ter ou não a passagem de valores por meio de uma variável, comumente denominada de parâmetro (na chamada) e retorno (no término da execução).

Figura 15 - Chamada de subalgoritmos



Fonte: Elaborada pelo autor.

O emprego de subalgoritmos apresenta algumas vantagens, tais como:

- Reduzem o tamanho do algoritmo como um todo, porque trechos repetidos só serão implementados uma vez em um subalgoritmo e, depois, serão apenas “chamados”;
- Facilitam a compreensão e visualização do que faz o algoritmo, porque o desenvolvimento modularizado faz com que se possa pensar no algoritmo por partes;
- Facilidade de depuração (correção/acompanhamento): é mais fácil corrigir/detectar um erro apenas uma vez do que em dez trechos diferentes;
- Facilidade de alteração do código: se é preciso alterar, altera-se apenas uma vez, no subalgoritmo;
- Generalidade de código com o uso de parâmetros: é possível escrever algoritmos para situações genéricas.

3.2.2 Funções

Uma função é um subalgoritmo que, além de executar uma determinada tarefa, retorna, explicitamente, um valor para quem a chamou (o algoritmo principal ou a outro subalgoritmo que a tenha chamado), que é o resultado de sua execução. Esse valor é retornado através da instrução Retorne. A chamada de uma função aparece como uma expressão e não apenas como um comando, como foi o caso do procedimento, isso porque tem de existir alguma variável para receber ou alguma expressão para usar o valor retornado pela função.

A declaração da função, de forma análoga ao procedimento, deve estar entre o final da declaração de variáveis do algoritmo principal e a linha início do mesmo e obedece à seguinte sintaxe:

```
funcao nomeFuncao (<sequência-de-declarações-de-parâmetros>): Tipo de
Retorno
var //declaração de variáveis locais
inicio
// Seção de comandos
retorne <variável ou expressão de retorno>
fimfuncao
```

A <sequência-de-declarações-de-parâmetros> é uma sequência de [var] <sequência-de-parâmetros>: <tipo-de-dado> separadas por ponto e vírgula.

3.2.3 Passagem de parâmetros por Valor

A ausência (opcional) da palavra-chave var indica passagem de parâmetros por referência!

```
funcao nomeFuncao (<sequência-de-declarações-de-parâmetros>): Tipo de
Retorno
var //declaração de variáveis locais
inicio
// Seção de comandos
retorne <variável ou expressão de retorno>
fimfuncao
```

Figura 16 - Passagem de parâmetros por valor

funcao soma (x,y : inteiro) : inteiro

Na <sequência-de-declarações-de-parâmetros> não aparece a palavra var, logo o recebimento de parâmetros é feito por VALOR !

```
1  Algoritmo "soma"
2  var a,b,resultado: inteiro
3
4  funcao soma(x,y: inteiro):inteiro
5     var res:inteiro
6     inicio
7       res <- x + y
8       retorne res
9     Fimfuncao
10
11 Inicio
12   a <- 10
13   b <- 20
14   resultado <-soma (a,b)
15   escreval("SOMA = ", resultado)
16 fimalgoritmo
```

Escopo	Nome	Tipo	Valor
GLOBAL	A	I	10
GLOBAL	B	I	20
GLOBAL	RESULTADO	I	30

```
Inicio da execução
SOMA =  30

Fim da execução.
```

Os valores das variáveis a e b são passados diretamente para as variáveis x e y (parâmetros da função soma).

São passados por VALOR (seu conteúdo).

Implemente e teste no VISUALg!

Fonte: Elaborada pelo autor.

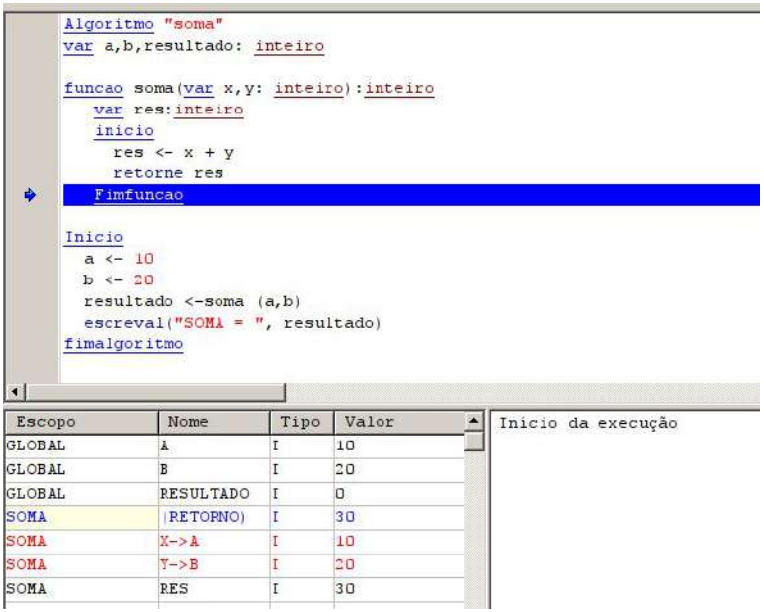
3.2.4 Passagem de parâmetros por Referência

A presença (opcional) da palavra-chave var indica passagem de parâmetros por referência!

```
funcao nomeFuncao (var<sequência-de-declarações-de-parâmetros>): Tipo de Retorno
var //declaração de variáveis locais
inicio
// Seção de comandos
retorne <variável ou expressão de retorno>
fimfuncao
```

Figura 17 - Passagem de parâmetros por referência

funcao soma (var x,y : inteiro) : inteiro
Na <sequência-de-declarações-de-parâmetros> aparece a palavra **var** logo o recebimento de parâmetros é feito por REFERÊNCIA!



As variáveis x e y (parâmetros da função soma) apontam para os endereços de memória das variáveis a e b (local onde estão armazenados os valores delas).

São passados por REFERÊNCIA (endereços em vermelho na imagem ao lado X->A e Y->B).

Implemente e teste no VISUALg!

Fonte: Elaborada pelo autor.

3.2.5 Procedimentos

Um procedimento é um subalgoritmo que não retorna valores ao algoritmo principal ou a outro subalgoritmo que o tenha chamado. Ele pode apenas receber valores por meio dos parâmetros e nunca retornar valores explicitamente como no caso das funções que usam a instrução "retorne".

Sua declaração, como descrito anteriormente, deve estar entre o final da declaração de variáveis do algoritmo principal e a linha início do mesmo e obedece à seguinte sintaxe:

```
procedimento nomeProcedimento (<sequência-de-declarações-de-parâmetros>)
  Var //declaração de variáveis locais
inicio
//Seção de Comandos
fimprocedimento
```

A <sequência-de-declarações-de-parâmetros> é uma sequência de [var] <sequência-de-parâmetros>: <tipo-de-dado> separadas por ponto e vírgula.

3.2.6 Uso de funções e procedimentos

Cada subalgoritmo pode ter suas próprias variáveis, chamadas de variáveis LOCAIS. Elas existem apenas durante a execução do subalgoritmo e só fazem sentido dentro dele.

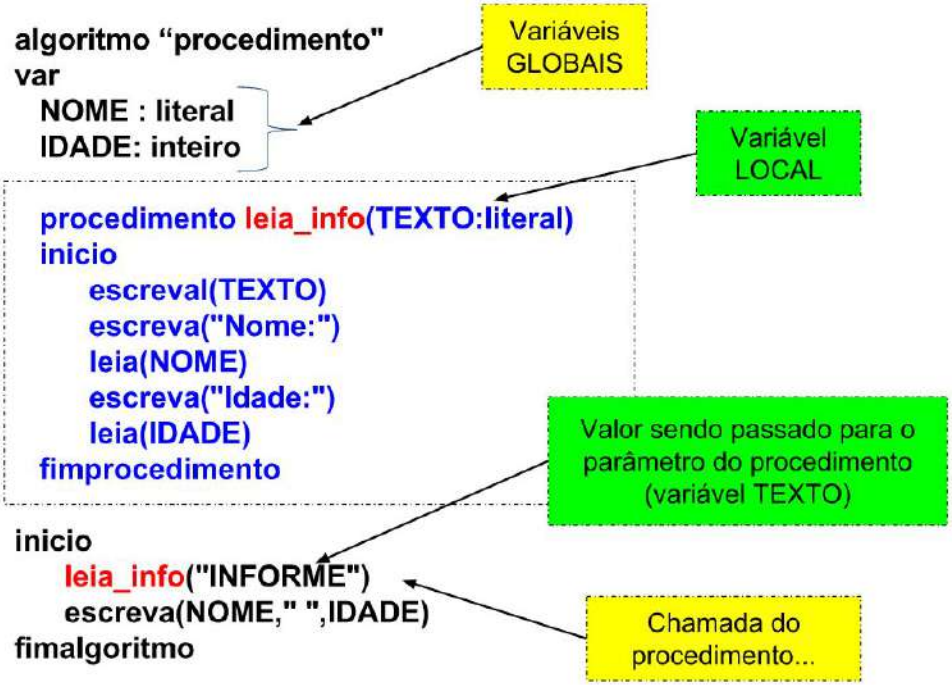
O subalgoritmo pode também ter acesso às variáveis definidas no algoritmo principal. Na verdade, as variáveis declaradas na parte inicial do algoritmo principal ficam disponíveis para uso em qualquer parte deste algoritmo ou por qualquer subalgoritmo. Por isso, elas são chamadas de variáveis GLOBAIS.

Porém, para manter a modularidade dos programas, NÃO É RECOMENDADO que se faça uso de variáveis GLOBAIS dentro dos subalgoritmos. E sim, ao invés disso, é importante que se use passagem de parâmetros.

As variáveis GLOBAIS existem na memória do computador durante toda a execução do algoritmo, ou seja, até que o algoritmo completo chegue ao seu final. Já as variáveis LOCAIS somente são criadas quando o subalgoritmo que as contém é ativado e, ao seu término, elas são liberadas/desalocadas da memória, tornando-se inacessíveis.

A Figura 18, a seguir, demonstra um algoritmo que contém um procedimento chamado “leia_info”. Esse procedimento é um subprograma que recebe um parâmetro do tipo “literal” que será armazenado na variável TEXTO, uma variável LOCAL ao procedimento. O algoritmo possui duas variáveis GLOBAIS chamadas NOME e IDADE, as quais estão disponíveis durante toda a execução do algoritmo. Perceba que a variável TEXTO, local ao procedimento, só existe dentro do subprograma.

Figura 18 - Procedimento e escopo de variáveis GLOBAIS e LOCAIS

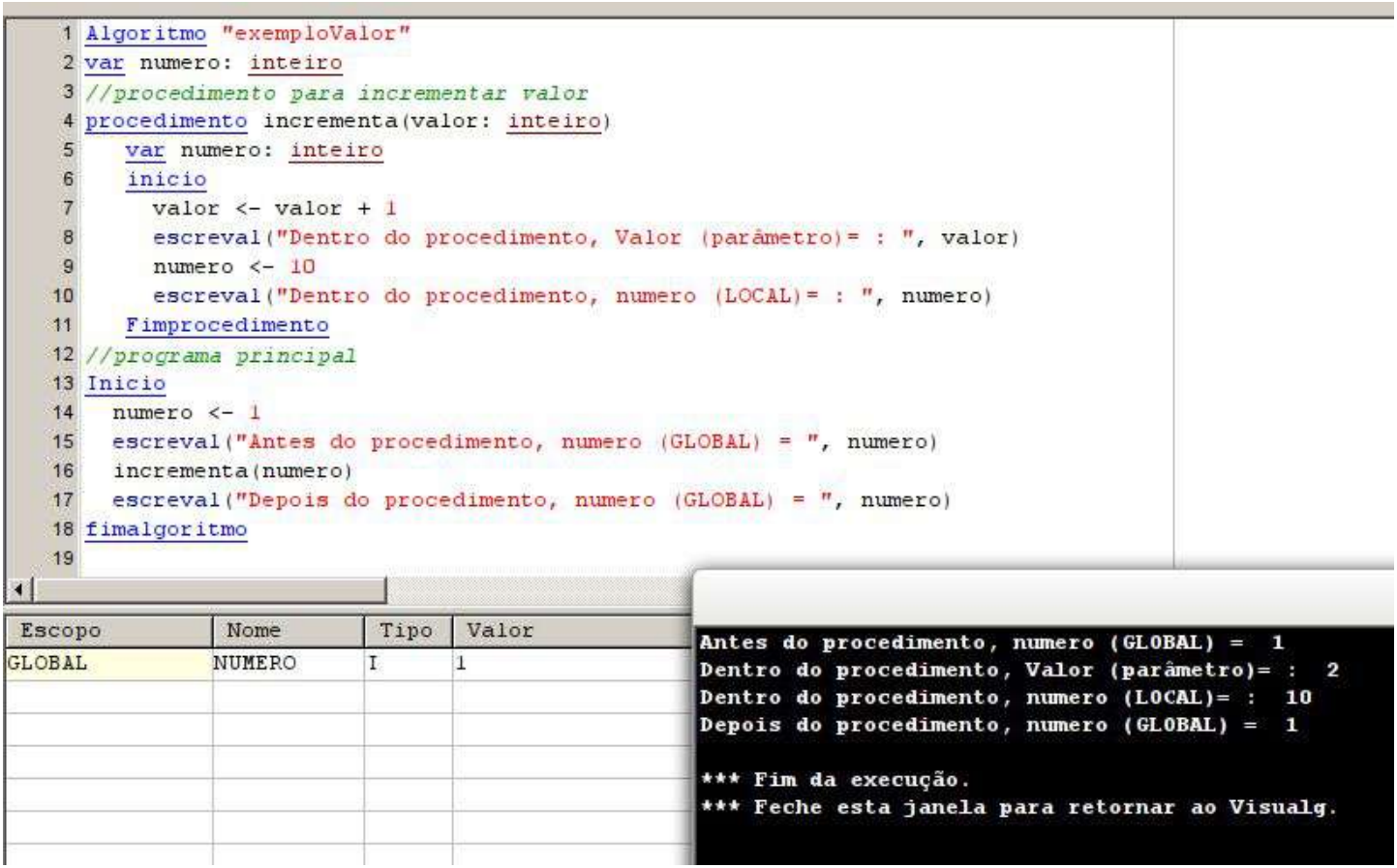


Fonte: Elaborada pelo autor.

Quando o procedimento leia_info é chamado, ele passa a string (conjunto de caracteres) "INFORME" para a variável TEXTO, variável local do subprograma. É uma forma de passar valor do programa principal para o subprograma (um parâmetro). Os comandos leia dentro do procedimento receberão valor para as variáveis GLOBAIS NOME e IDADE. Ao término do procedimento, elas não deixam de existir, pois são GLOBAIS.

Na Figura 19, temos um outro exemplo interessante. Há uma variável GLOBAL chamada número do tipo inteiro no algoritmo principal e também, de igual forma, dentro do procedimento "incrementa" há outra chamada número, só que esta última, apesar de idêntica, é uma variável LOCAL. Isso significa que estão separadas pelo ESCOPO e, portanto, uma não irá influenciar a outra. Note que o procedimento não tem retorno!

Figura 19 - Exemplo de procedimento com escopo de variável com mesmo nome

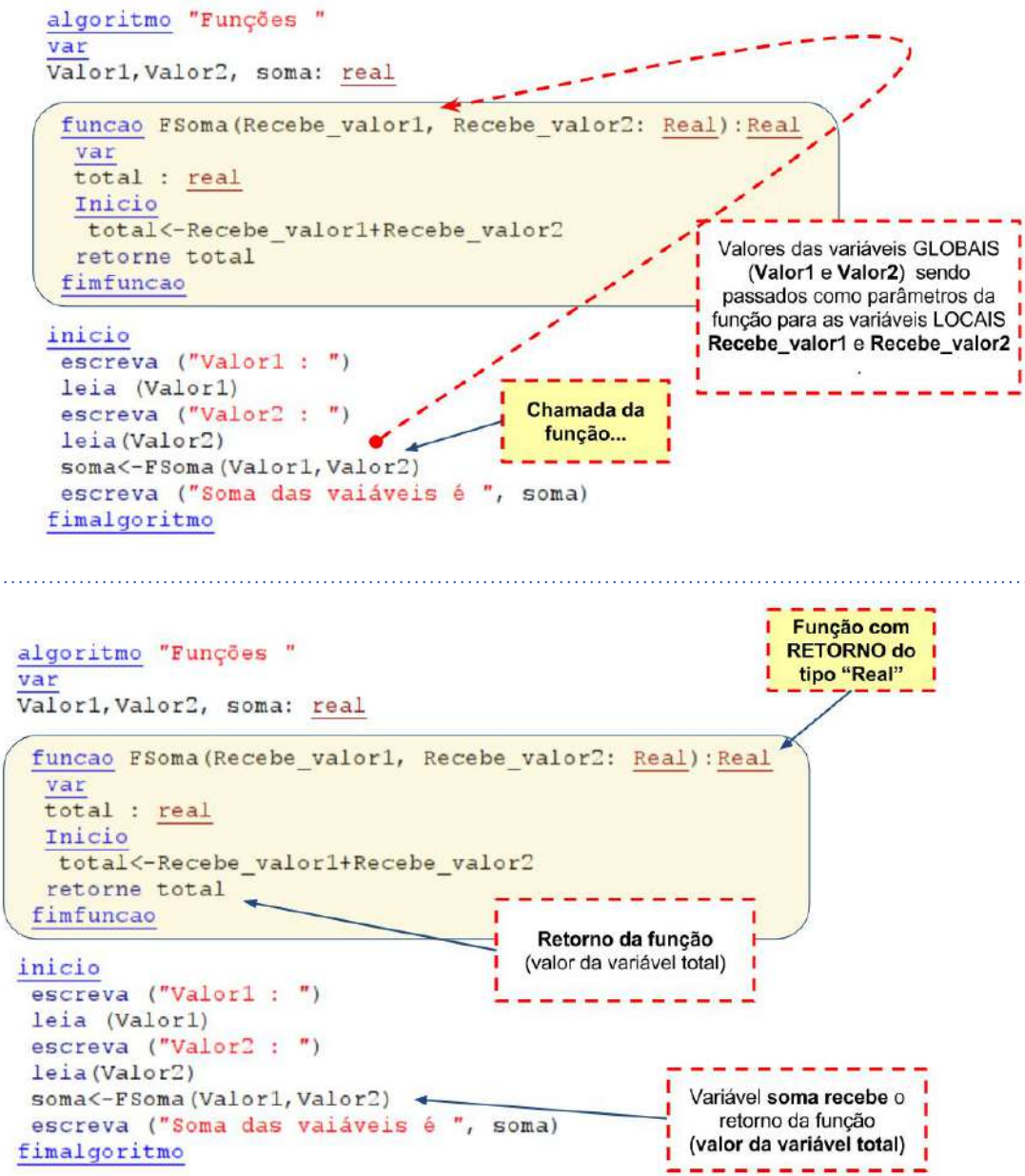


Fonte: Elaborada pelo autor.

No início da execução, a variável GLOBAL número é criada (linha2). No início do algoritmo, na linha 13 (próxima linha a ser executada depois da declaração de variáveis globais) a variável número GLOBAL recebe (armazena) o valor 1. A linha 14 imprime o seu valor (número vale 1) na tela. A linha 15 chama o procedimento e passa o valor de número para o parâmetro do procedimento (valor) na linha 4. Veja que a execução foi desviada para a linha 4. A variável valor é local e só existe dentro do procedimento "incrementa". A linha 5, a ser executada na sequência, CRIA a variável LOCAL número (ela ainda não tem valor armazenado dentro dela). As linhas continuam executando na sequência e na linha 9 a variável número (local) recebe (armazena) o valor 10. Note que o valor de número (variável global) não se altera. Na linha 10, a variável LOCAL número é impressa na tela. Na linha 11, o procedimento termina (Fimprocedimento) e a execução retorna para a linha 17, a linha posterior a chamada do procedimento. A linha 17 fará a impressão na tela do valor da variável GLOBAL número (pois estamos fora do procedimento - ele já terminou e a variável número local não existe mais). A variável GLOBAL número continua com o valor 1 armazenado. Ela não foi alterada devido ao ESCOPO DAS VARIÁVEIS.

O próximo exemplo apresenta a utilização de função com retorno. O algoritmo apresentado possui variáveis globais (Valor1, Valor2 e soma). Dentro da função também temos a variável total com escopo local, ou seja, só existe dentro da função. A chamada da função é feita por meio de uma variável com atribuição (para armazenar o retorno da função Fsoma. A chamada da função possui dois parâmetros que são recebidos por duas variáveis. É uma forma de passar valores para a função.

Figura 20 - Exemplo de função com escopo de variável local e global

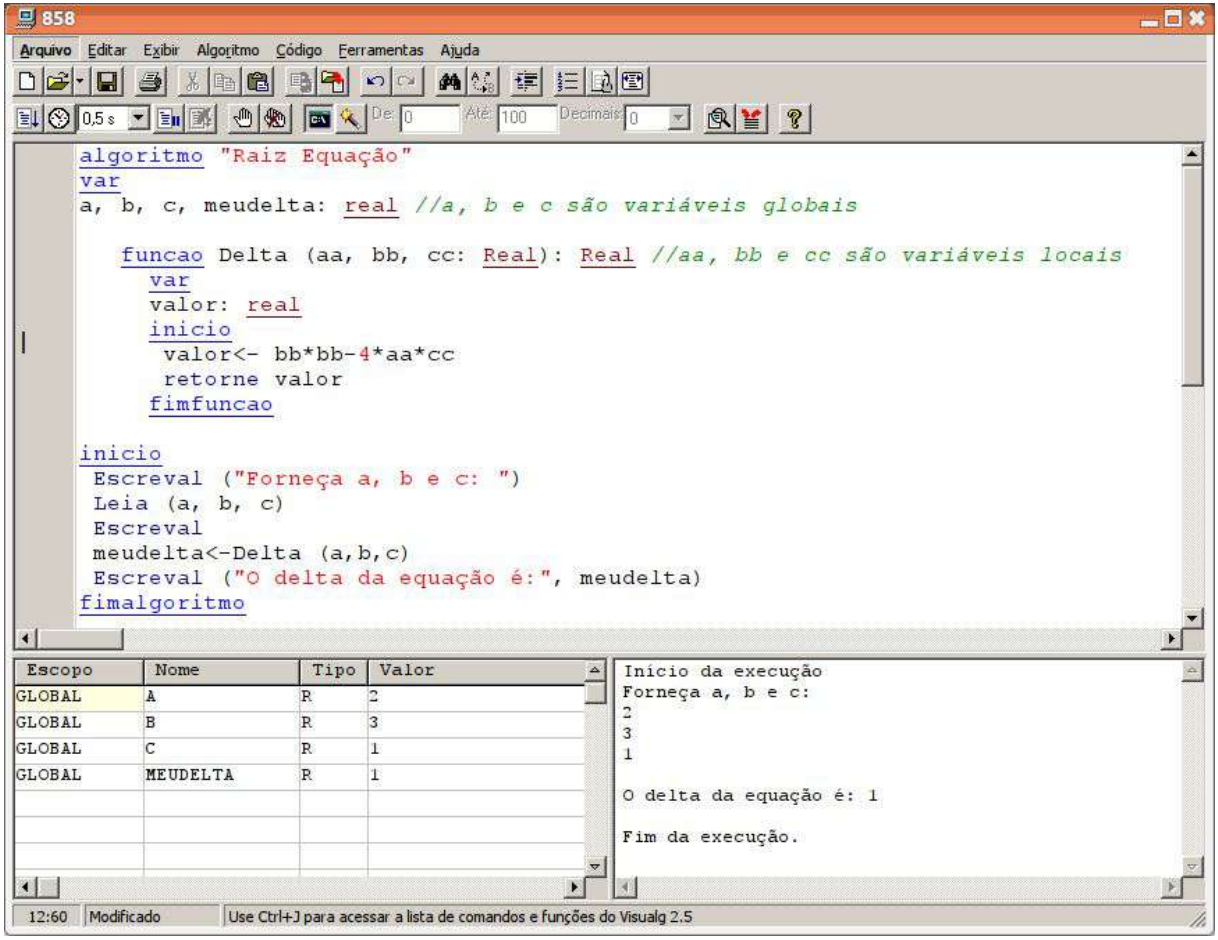


Fonte: Elaborada pelo autor.

Na Figura 20, é importante notar também que a função tem sempre um retorno e na sua declaração fazemos a definição do tipo de retorno que ela utilizará. A função, como tem retorno, terá sempre a sua chamada precedida pela atribuição a uma variável, pois, desta maneira, a variável que está em sua chamada receberá, por meio de atribuição, o valor do retorno da função.

Observe e implemente em VISUALg o exemplo de função da Figura 21.

Figura 21 - Função para calcular o delta



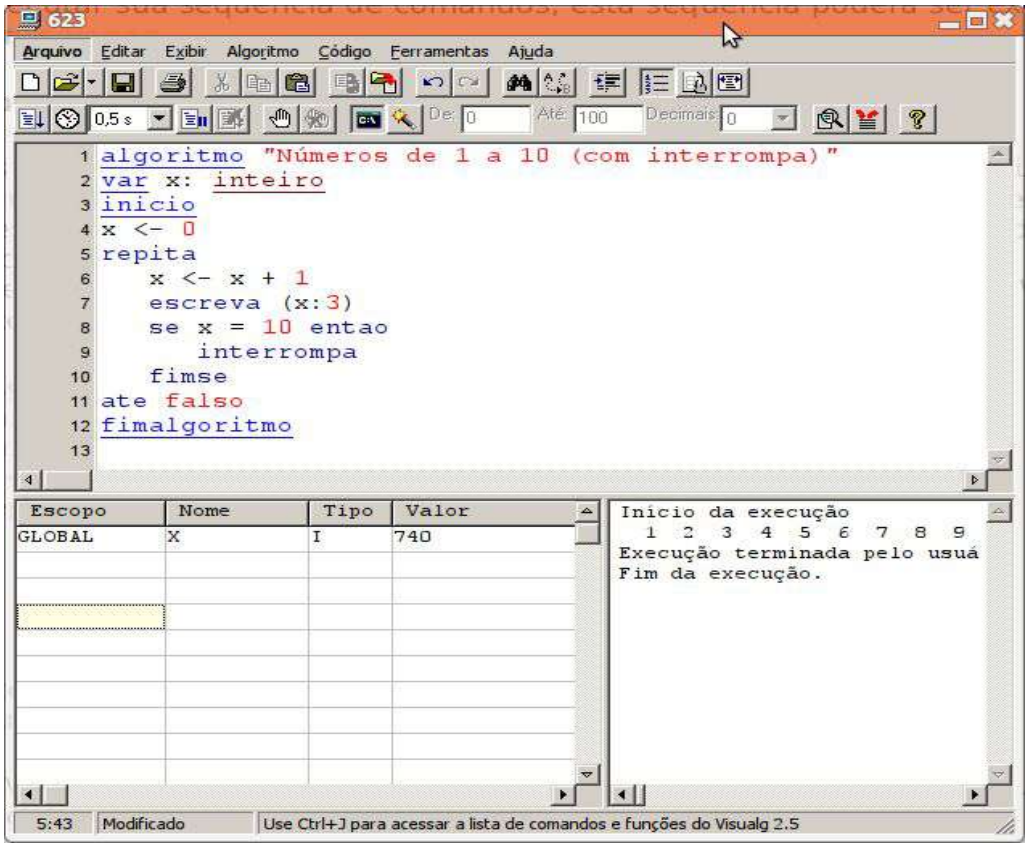
Fonte: Elaborada pelo autor.

Comando Interrompa

As estruturas de repetição aprendidas permitem o uso do comando interrompa, que causa uma saída imediata do laço. Embora esta técnica esteja de certa forma em desacordo com os princípios da programação estruturada, o comando interrompa foi incluído no VISUALg por ser encontrado na literatura de introdução à programação e mesmo em linguagens como o Object Pascal (Delphi/Kylix), Clipper, C, C++, PHP, JAVA, VB, etc.

Observe e implemente em VISUALg o exemplo de função com o uso do comando interrompa conforme a figura abaixo.

Figura 22 - Função com uso do comando interrompa



Fonte: Elaborada pelo autor.

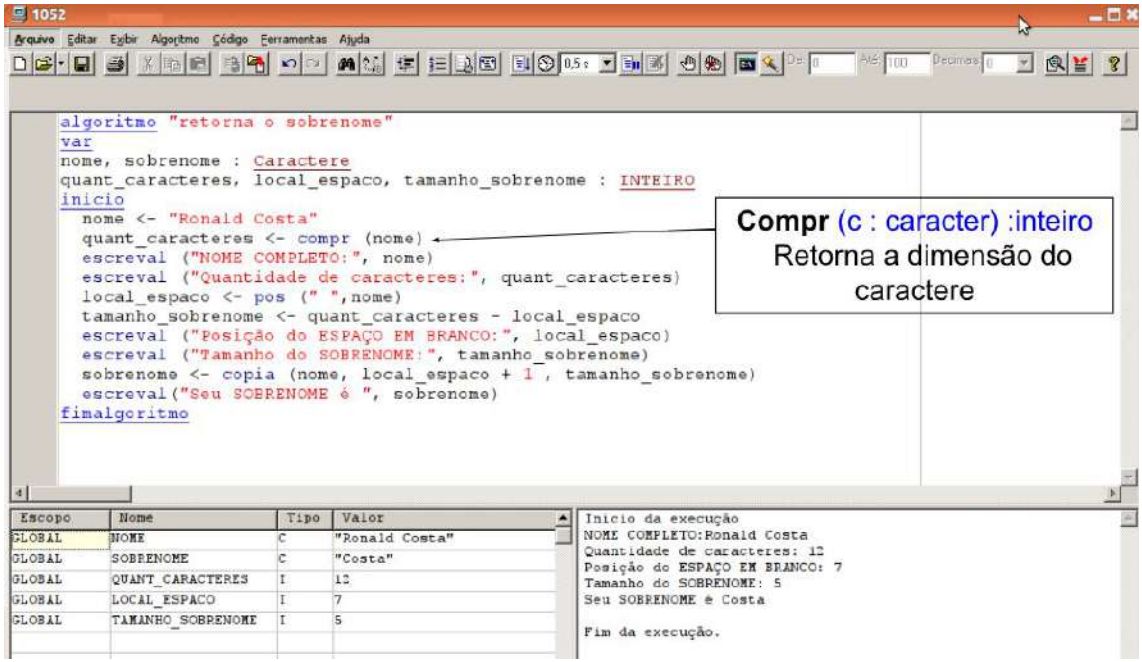
Vejamos uma lista de funções predefinidas do VISUALg.

Lista de funções predefinidas do VISUALg	
Abs (valor : real) :real	Valor absoluto
Arccos (valor : real) :real	Arco cosseno
Arcsen (valor : real) : rea	Arco seno
Arctan (valor : real) :real	Arco tangente
Asc (s : caracter) :inteiro	Retorna o código ASCII
Compr (c : caracter) :inteiro	Retorna a dimensão do caractere
Copia (c : caracter , posini, posfin : inteiro) :caracter	Copia um determinado caractere
Cos (valor : real) :real	Coseno
Cotan (valor : real) :rea	Co-tangente
Exp (<base>,<expoente>) :real	Potenciação
Grauprad (valor : real) :real	Converte grau para radiano
Int (valor : real):inteiro	Converte o valor em inteiro
Log (valor : real) :real	Logaritmo de base 10
Logn (valor : real) :real	Logaritmo natural (ln)
Maiusc (c : caracter) :caracter	Converte em maiúscula
Minusc (c : caracter) :caracter	Converte em minúscula

Lista de funções predefinidas do VISUALg	
Numpcarac (n : inteiro ou real) :caracter	Converte um número inteiro ou real para caracteres
Pi :real	Valor Pi
Pos (subc, texto: caracter) :inteiro	Retorna a posição de subc em texto
Quad (valor : real) :real	Elevado quadrado
Radpgrau (valor : real) :real	Converte radiano em grau
Raizq (valor : real) :real	Raiz quadrada
Rand :real	Gerador de números aleatórios entre 0 e 1
Randi (limite : inteiro) :inteiro	Gerador de números inteiros aleatórios com um limite determinado
Sen (valor : real) :real	Seno
Tan (valor : real) :real	Tangente

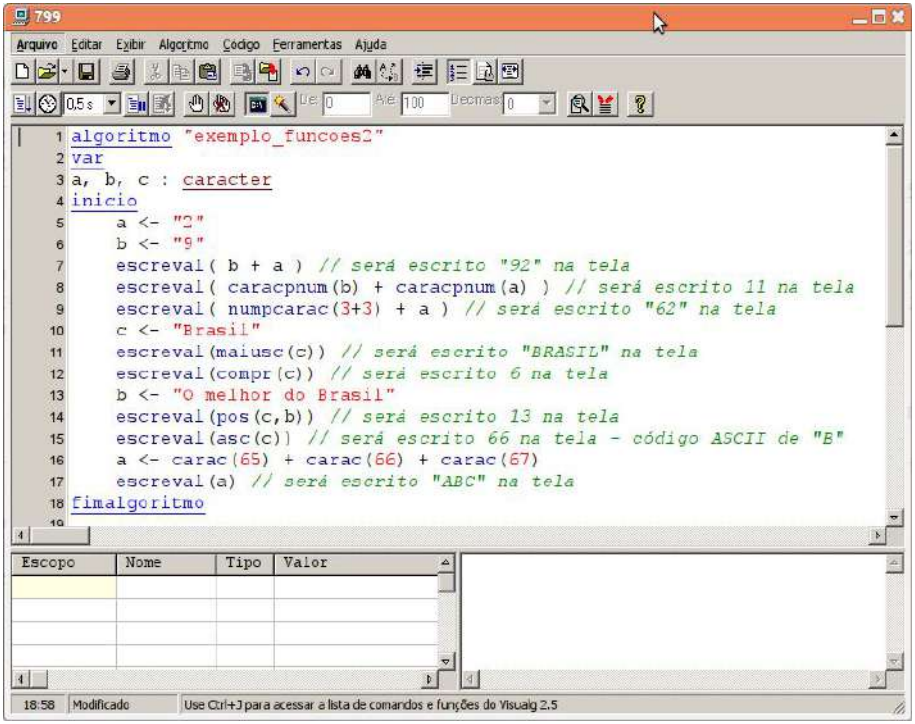
Agora que conhecemos novas funcionalidades (funções pré-definidas), vamos implementar algumas em VISUALg, conforme os exemplos de funções da Figura 23 a seguir.

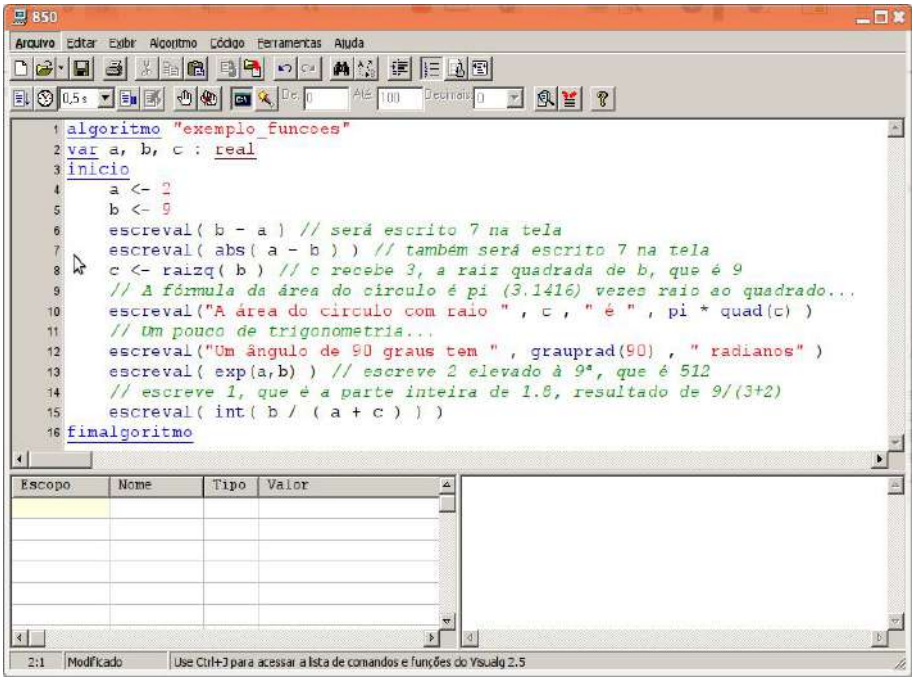
Figura 23 - Exemplos de código com funções pré-definidas



Fonte: Elaborada pelo autor.

Continua...





Fonte: Elaborada pelo autor.

Números aleatórios

Muitas vezes, a digitação de dados para o teste de um programa torna-se uma tarefa entediante. Com o uso do comando aleatório do VISUALg sempre que um comando leia for encontrado, a digitação de valores numéricos e/ou caracteres é substituída por uma geração aleatória.

Este comando não afeta a leitura de variáveis lógicas: com certeza, uma coisa pouco usual em programação. O VISUALg implementa algumas extensões às linguagens "tradicionais" de programação com o intuito principal de ajudar o seu uso apenas como ferramenta de ensino.

O comando aleatório tem a seguinte sintaxe:

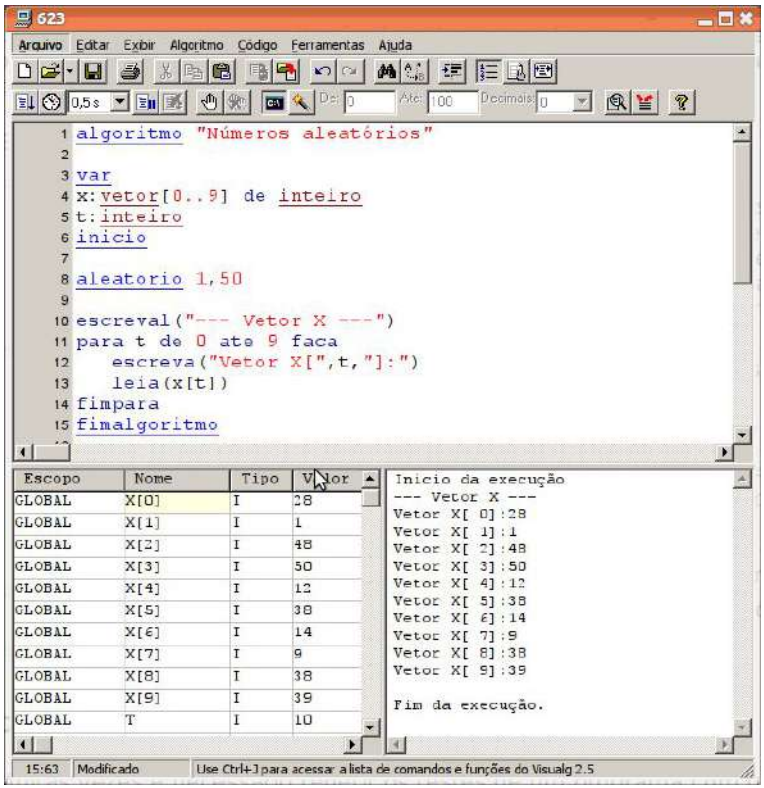
aleatorio [on]	Ativa a geração de valores aleatórios que substituem a digitação de dados. A palavra-chave *on* é opcional. A faixa padrão de valores gerados é de 0 a 100 inclusive. Para a geração de dados do tipo caractere, não há uma faixa pré-estabelecida: os dados gerados serão sempre strings de 5 letras maiúsculas.
aleatorio <valor1>[, <valor2 >]	Ativa a geração de dados numéricos aleatórios estabelecendo uma faixa de valores mínimos e máximos. Se apenas < valor1> for fornecido, a faixa será de 0 a <valor1> inclusive; caso contrário, a faixa será de <valor1> a <valor2> inclusive. Se <valor2> for menor que <valor1>, o VISUALg os trocará para que a faixa fique correta. Importante: <valor1> e <valor2> devem ser constantes numéricas e não expressões.
aleatorio off	Desativa a geração de valores aleatórios. A palavra-chave off é obrigatória.

Vamos refletir?

O VISUALg, plataforma que utilizamos para implementar o português estruturado em nossos exemplos, possui um conjunto de funções predefinidas. Essas funções são subprogramas que disponibilizam funcionalidades já prontas para o uso. Em linguagens de programação também é possível encontrar funções pré-definidas para reuso de código, normalmente por meio de bibliotecas.

Observe e implemente em VISUALg o exemplo de uso do comando aleatório conforme a Figura 24.

Figura 24 - Exemplo de código com funções pré-definidas



Fonte: Elaborada pelo autor.

Saiba mais!

O que são Funções e Procedimentos?

Hummm! Boa pergunta, não? Se você ainda está com alguma dúvida ou deseja aprofundar um pouco mais, vamos olhar esse site para entender esses conceitos um pouco melhor. Ok? Vamos explorar um pouco mais! Não desista. Seja persistente! Teste e modifique os códigos!

Acesse: https://dicasdeprogramacao.com.br/o-que-sao-funcoes-e-procedimentos/

Foi uma viagem e tanto. Muitos conceitos novos que vão se agrupando a tudo que já estudamos. Neste tópico aprendemos a trabalhar com funções e procedimentos. Observamos que a diferença entre essas estruturas está no retorno de valores. As funções sempre retornam valores. Empregamos a passagem de valores por valor e referência, um conceito bastante importante em programação. Foi possível compreender ainda que o Visuag, assim como as linguagens de programação, possui funções prédefinidas que facilitam o dia a dia do programador. Compreendemos que o reuso de código facilita o desenvolvimento e torna mais fácil a manutenção de softwares. Aproveite para consultar o glossário se tiver alguma dúvida de algum termo. Vamos partir para algumas questões de aprendizagem e para a revisão final do tópico. Não esqueça de olhar os sites e vídeos indicados que irão ampliar os seus horizontes de conhecimentos sobre o tema da unidade. Em nossos slides de apoio existem outros exemplos esclarecedores sobre a aplicação de funções e procedimentos. Aproveite para implementar. Vamos em frente para os próximos tópicos! Agora vamos implementar em C / C++ tudo que vimos em Lógica e Visualg! Agora sim vamos criar programas de verdade!

Vamos rever?

A modularização é uma característica muito importante no desenvolvimento de programas. Ela é um método utilizado para facilitar a construção de grandes algoritmos, através de sua divisão em pequenas etapas, que são os subprogramas. É a aplicação direta do jargão "Dividir para conquistar".

Um "Subprograma" é um programa que auxilia o programa principal através da realização de uma determinada subtarefa. Os subprogramas são chamados dentro do corpo do programa principal como se fossem comandos. É conveniente utilizá- los quando uma determinada tarefa é efetuada em diversos lugares no mesmo algoritmo. Ao invés de escrever-se um trecho diversas vezes, escreve-se um subprograma e este pode ser invocado (executado) diversas vezes. Existem dois tipos de subprogramas.

O primeiro é o procedimento (sub-rotina) que não retorna nenhum valor ao término de sua execução. E o segundo, a função que retorna um valor ou uma informação quando termina a sua execução. É possível perceber que a diferença entre procedimentos e função está apenas no retornar ou não um valor ao término de sua execução. A chamada de um subprograma ocorre quando invocamos o trecho de código pelo seu nome (identificador). Um algoritmo pode chamar um subprograma e este, por sua vez, poderá chamar outro e assim sucessivamente, transferindo o ponto de execução do algoritmo para o subprograma chamado. Ao término da execução do subprograma, o ponto de execução retorna, logo após o ponto inicial da chamada. Tanto a chamada quanto o término da execução de um subprograma podem ter ou não a passagem de valores por meio de uma variável, comumente denominada de parâmetro (na chamada) e retorno (no término da execução).

Sites indicados

1) Apoio Informática - Procedimentos em VISUALg
   http://www.apoioinformatica.inf.br/produtos/item/17-procedimentos

2) Apoio Informática - Funções em VISUALg
   http://www.apoioinformatica.inf.br/produtos/visualg/linguagem/item/18-funcoes

Audiovisuais indicados

1) Procedimentos - Curso de Algoritmos #12 - Gustavo Guanabara
   https://www.youtube.com/watch?v=KoNehy7rn8U

2) Funções - Curso de Algoritmos #13 - by Gustavo Guanabara
   https://www.youtube.com/watch?v=-nNx7e8GzHQ

3) 26 - Lógica de Programação - Procedimentos
   https://www.youtube.com/watch?v=u-y87XhfHmA

4) 27 - Lógica de Programação - Funções
   https://www.youtube.com/watch?v=NTPDUw16cm8

Questões de autoaprendizagem

Crie um algoritmo que tenha 3 procedimentos e uma função e execute as seguintes tarefas.

a) O primeiro procedimento fará a construção de um menu de opções para o sistema:

```
MENU
-------------------------
1 - Carregar dados
2 - Exibir dados
3 - Soma da Matriz
4 - Sair
-------------------------
Digite a opcao=>
```

b) O segundo procedimento irá receber valores para preencher um VETOR (matriz de 3x3) com dados digitados pelo usuário. A variável do vetor (matriz) será global.

c) O terceiro procedimento irá desenhar (exibir) a matriz na tela da seguinte forma (os valores são apenas exemplo):

```
Dados da Matriz
------------------------
 | 1 | 2 | 3
 | 4 | 5 | 6
 | 7 | 8 | 9

Pressione <ENTER> para continuar...
```

Questões de autoaprendizagem

d) Uma função “soma” que irá retornar para o programa principal o valor da soma de todos os elementos da matriz. O resultado será impresso na tela pelo programa principal.

e) Uma função “maior” que irá retornar para o programa principal, o maior valor contido entre todos os elementos da matriz. O resultado será impresso na tela pelo programa principal.

Gabarito das questões de autoaprendizagem

Abaixo, apresento uma resposta incompleta que você poderá tomar por base para finalizar o que falta no algoritmo.

```
algoritmo "Menu para Matriz"
var
matriz: vetor [0..2,0..2] de inteiro
opcao:inteiro

procedimento menu()
  var
  inicio
  limpatela
  Escreval("  MENU")
  Escreval ("------------")
  Escreval ("1 - Carregar dados")
  Escreval ("2 - Exibir dados")
  Escreval ("3 - Sair")
Fimprocedimento

procedimento carregar()
  var
  x,y:inteiro
  inicio
    limpatela
    escreval ("Carregando dados na Matriz")
    escreval ("--------------------------")
    para x de 0 ate 2 faca
      para y de 0 ate 2 faca
        escreva ("Digite MATRIZ[",x,",",y,"]:")
        leia (matriz[x,y])
      fimpara
    fimpara
Fimprocedimento
```

Continua...

Gabarito das questões de autoaprendizagem

```
procedimento exibir()
  var
  x,y:inteiro
  enter:caractere
  inicio
    limpatela
    escreval ("Dados da Matriz")
    escreval ("--------------------------")
    para x de 0 ate 2 faca
      para y de 0 ate 2 faca
        escreva (" | ",matriz[x,y])
      fimpara
      escreval
    fimpara
    Escreval("Pressione <ENTER> para continuar..."
    Leia (enter)
Fimprocedimento
```

```
inicio
  // Seção de Comandos
  enquanto opcao<>3 faca
   menu()
   Escreval ("--------------------------")
   Escreva ("Digite a opcao=>")
   leia (opcao)
   escolha opcao
   caso 1
     carregar()
   caso 2
     exibir()
   caso 3
     limpatela
     Escreva ("FIM !!!")
   outrocaso
     limpatela
     Escreva ("OPÇÃO INVÁLIDA")
   fimescolha
  fimenquanto
fimalgoritmo
```


Bem vindos ao desafio final ! Prontos para começar? O nosso assunto agora é programação em Linguagem C e C++. É tempo de programar!
Maravilha! E o que temos neste tópico ?
Neste tópico, primeiro, vamos explorar as possibilidades da linguagem de programação C. Tudo aquilo que aprendemos em lógica e em portuguêsestruturalserá agora testado e implementado em C. Vamos começar desde a estrutura básica de um programa em C até chegarmos as estruturas de sequência, seleção e repetição. Algumas coisas que aprendemos específicas do VISUALG não se aplicarão por completo em C, pois cada linguagem de programação tem suas peculiaridades. Porém a lógica continuará a mesma! Vamos programar!

Unidade 4

Linguagem C, Linguagem Orientada a Objetos com C++

Ao término desta unidade, esperamos atingir os seguintes objetivos:

- Codificar algoritmos nas linguagens de programação C e C++;
- Produzir programas legíveis, eficientes e corretos.

Vamos começar!

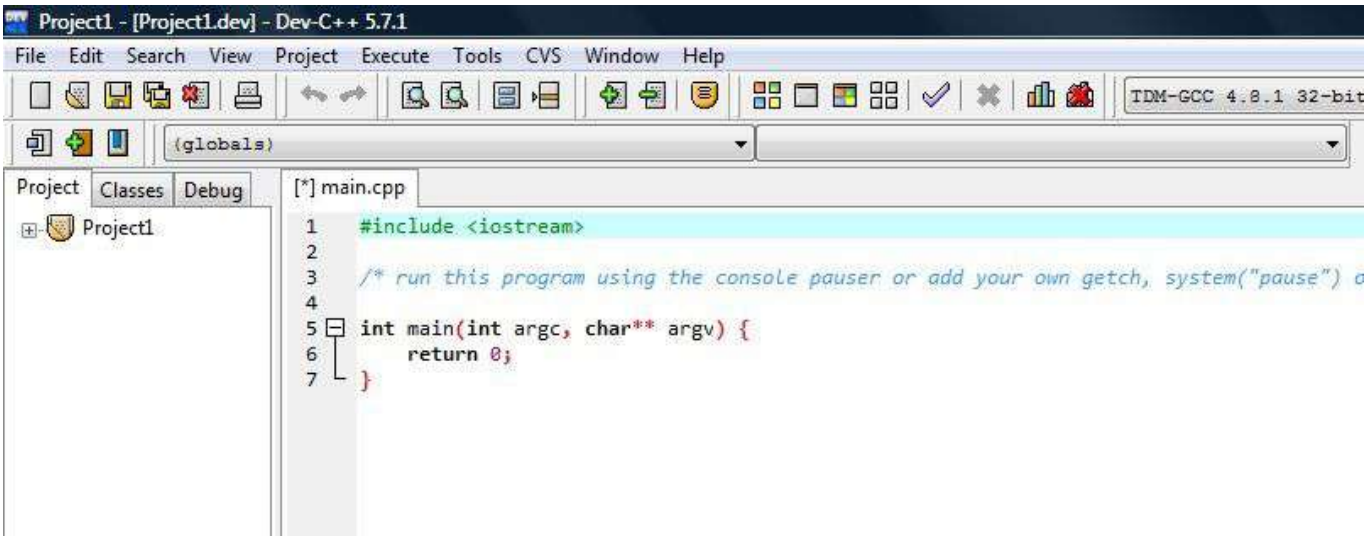
4.1 Preparando o ambiente de programação C

Antes de entrarmos na programação propriamente dita, é preciso escolher um ambiente de programação. Para a linguagem C / C++ existem vários disponíveis. Vamos apresentar alguns e você escolhe o que melhor atender a sua demanda.

a) Dev C++ (Windows)

É um ambiente bastante simples para ser utilizado no Windows. Na unidade 1, já demonstramos a sua instalação. Está disponível para download em: https://sourceforge.net/projects/orwelldevcpp.

Figura 1 - Tela do Dev C++



Fonte: Imagem retirada da internet.
Disponível em: https://commons.wikimedia.org/wiki/File:08_devC%2B%2B_TemplateProgramma.jpg.
Acesso em: 07 de jun.2022.

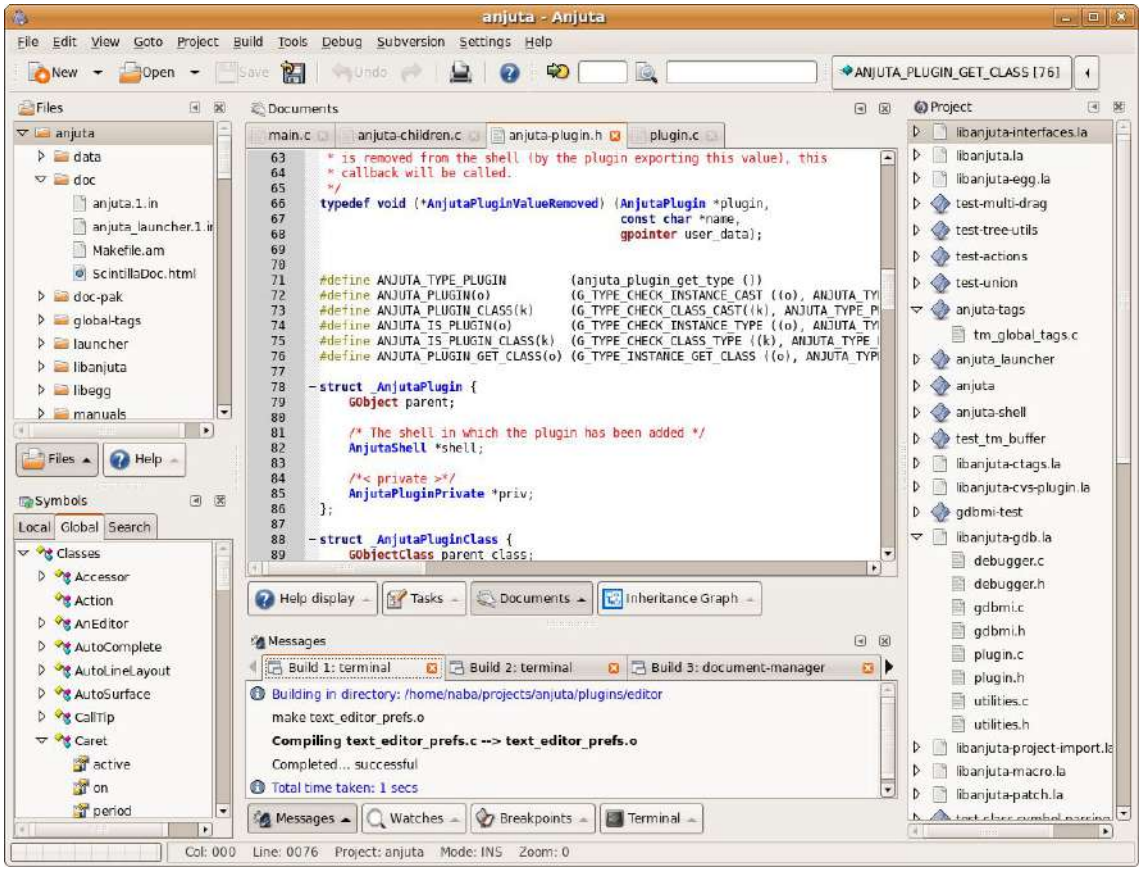
b) Anjuta (GNU/linux)

Ambiente semelhante ao Dev C++ só que para GNU/Linux. É bastante versátil e agradável. A sua instalação pode ser feita por pacotes no Debian ou Ubuntu. Está disponível para download em: http://anjuta.org/

No debian pode ser instalado com os comandos:

- apt-get install build-essentials
- apt-get install anjuta

Figura 2 - Tela do Anjuta

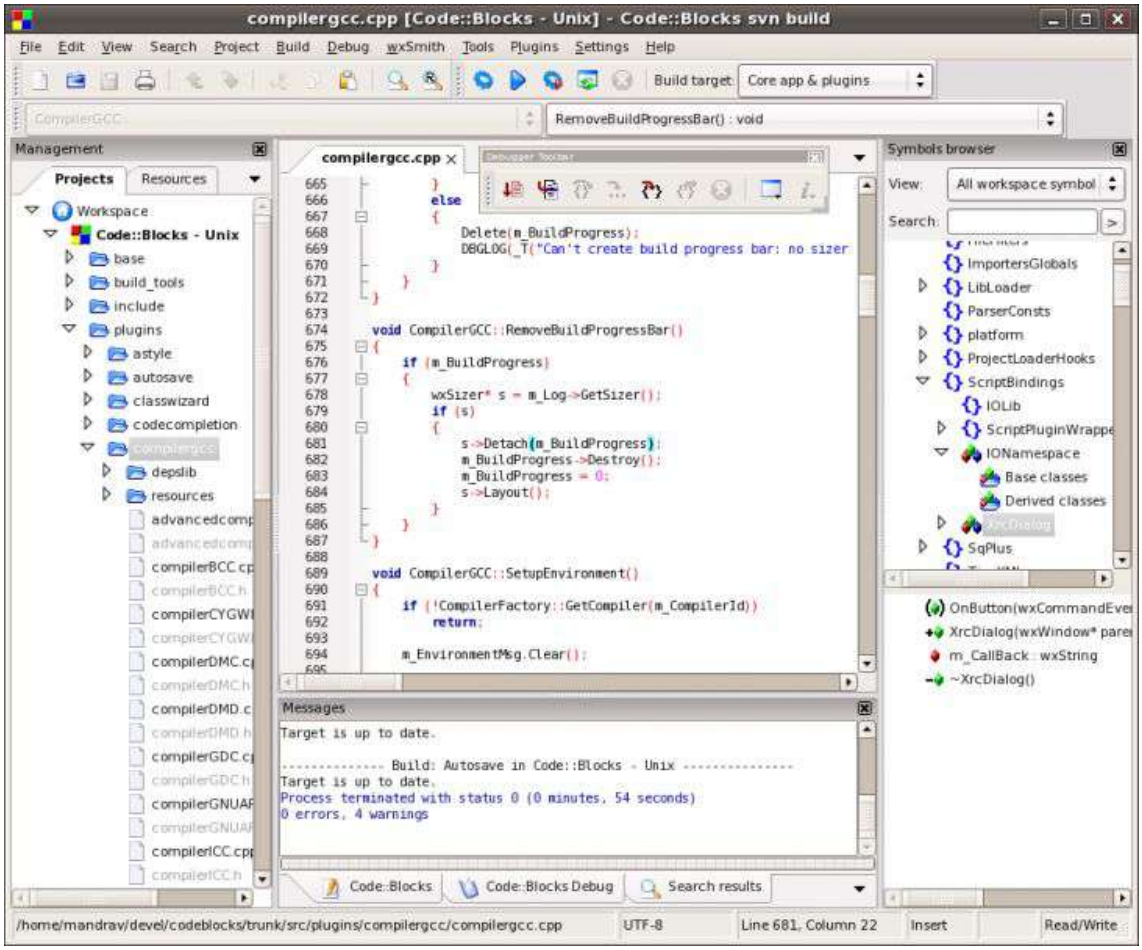


Fonte: Imagem retirada da internet. Disponível em: http://anjuta.org/wp-content/uploads/2014/06/anjuta-screenshot.png. Acesso em: 07 de jun.2022.

c) Codeblocks (GNU/linux)

Codeblocks é um IDE (Integrated Development Environment, ou Ambiente Integrado para Desenvolvimento) open source para a linguagem de programação C++. Trata-se de um projeto independente e criado com a colaboração de diversos desenvolvedores e programadores ao redor do mundo inteiro. Está disponível para download em: http://www.codeblocks.org/

Figura 3 - Tela do Codeblocks



Fonte: Imagem retirada da internet. Disponível em: <https://www.edivaldobrito.com.br/ide-codeblocks-no-linux-via-flatpak/>. Acesso em: 16 de ago.2022.

d) GCC / G++ (Terminal GNU/linux)

Outra forma bastante prática de compilar códigos em C / C++ é utilizar o compilador nativo do GNU/Linux. Utilizando um arquivo de texto que contenha o código fonte, gravado com um editor qualquer, basta acessar um terminal, digitar o comando de compilação e pronto.

1) No debian o gcc e o g++ podem ser instalados com os comandos:

- apt-get install build-essentials
- apt-get install gcc g++ nano

2) Depois crie um arquivo utilizando um editor do GNU/Linux. Neste caso, usamos o nano. Digite o código fonte mostrado na Figura 4.

```
$ nano modelo.c
```

Figura 4 - Editando um código fonte no terminal com nano

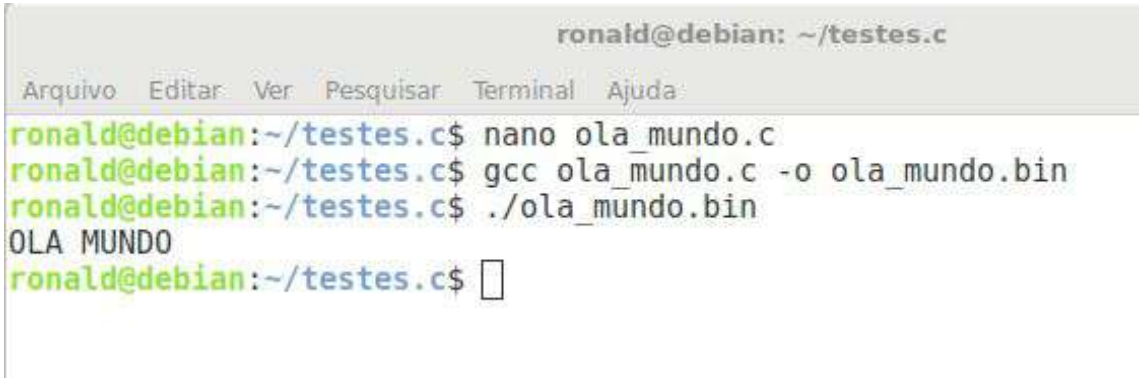
Fonte: Elaborada pelo autor.

3) Para salvar aperte CTRL+O, depois <ENTER>. Para sair CTRL+X.

4) Para compilar:
$ gcc <nome_arquivo_fonte> -o <nome_arquivo_binario_a_ser_gerado>

5) Para executar agora basta digitar:
$ ./<nome_do_arquivo_executavel>

Figura 5 - Compilando e executando

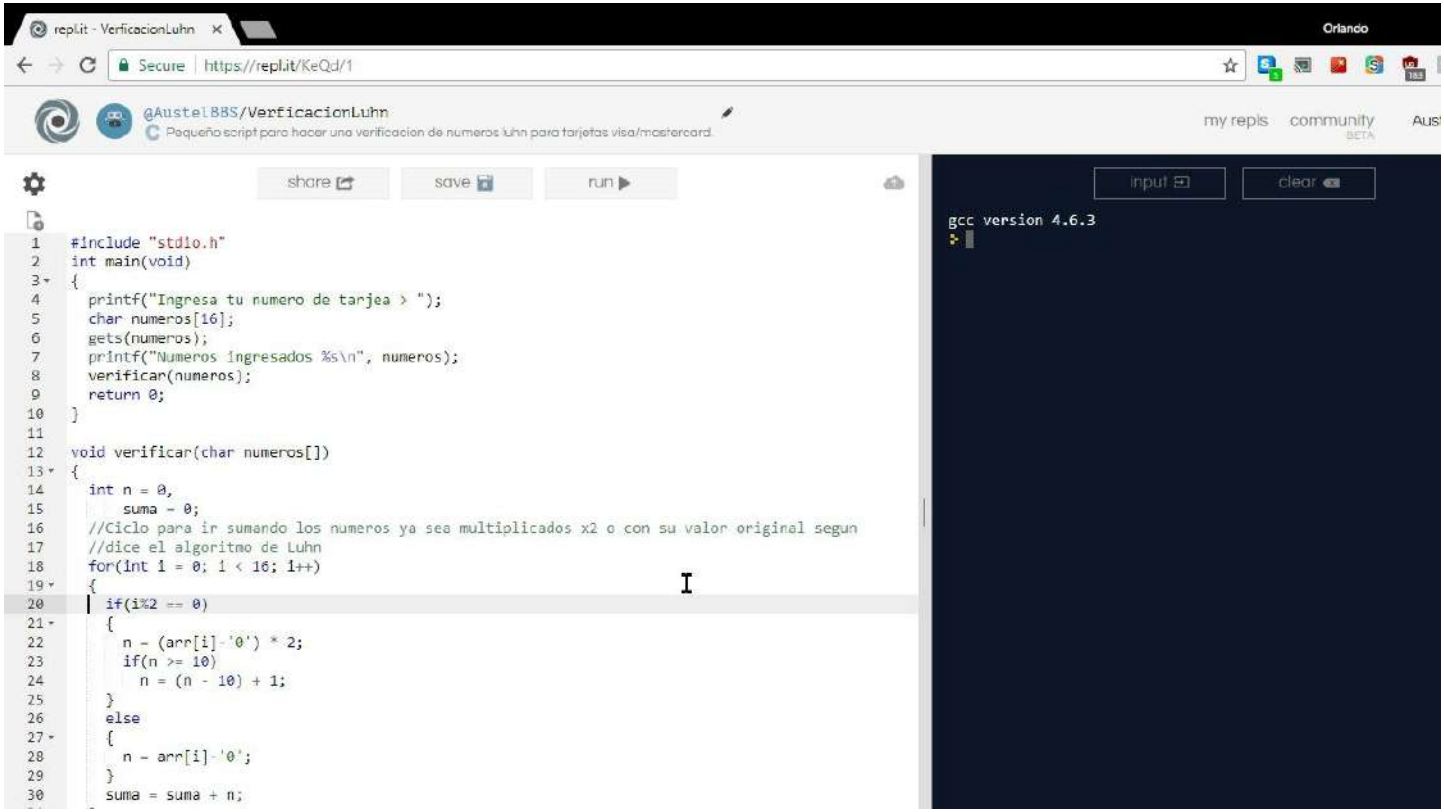


Fonte: Elaborada pelo autor.

e) Executando C/C++ no navegador (ONLINE) - repl.it

Outra forma bastante prática de executar códigos em C / C++ é utilizar um compilador *on-line*. Isso mesmo, você pode compilar ou executar programas diretamente no navegador em diversas linguagens diferentes. Com suporte a 30 linguagens de programação, o Replit é um trabalho de uma equipe pequena, porém muito bem feito. O editor permite configuração da identação e de seu visual. Possui 2 opções de tema (clara/escura). Se você criar sua conta *on-line*, poderá armazenar e compartilhar seus programas. Para utilizar basta acessar o endereço: https://repl.it.

Figura 6 - Compilando e executando



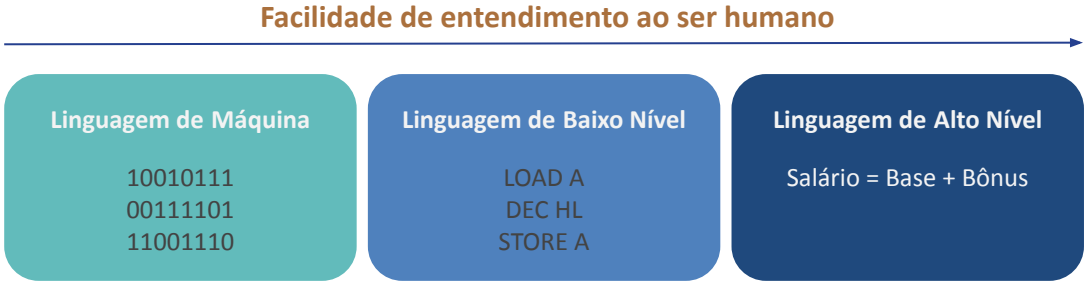
Fonte: imagem retirada da internet.
Disponível em: <http://i.ytimg.com/vi/YT1Uky0QhP4/maxresdefault.jpg>.
Acesso em: 02 de fev.2018.

Agora que o ambiente de programação foi escolhido e instalado, vamos dar início ao estudo da linguagem de programação C.

4.1.1 Estrutura básica de um programa em C

As linguagens de programação permitem a criação de programas empregando regras sintáticas e semânticas. O código fonte de um programa é o conjunto de palavras e códigos, organizado segundo as regras de cada linguagem. O código fonte para ser executado pelo processador do computador, primeiro deve ser traduzido e compilado para código de máquina (executável).

Figura 7 - Os níveis das linguagens de programação



Exemplos de trechos de código em diferentes linguagens

Fonte: Elaborada pelo autor.

Onde nasceu a Linguagem C?

A linguagem C nasceu a partir da Linguagem B, desenvolvida por Ken Thompson no final da década de 60.

A Linguagem C foi criada por Dennis M. Ritchie e Brian Kernighan em 1972, nos laboratórios Bell (EUA). Ela foi criada para o desenvolvimento do sistema operacional UNIX, o qual era escrito em Assembly (linguagem de baixo nível). A Linguagem C, assim como a Assembly, também possui instruções de baixo nível bem próximas à linguagem de máquina.

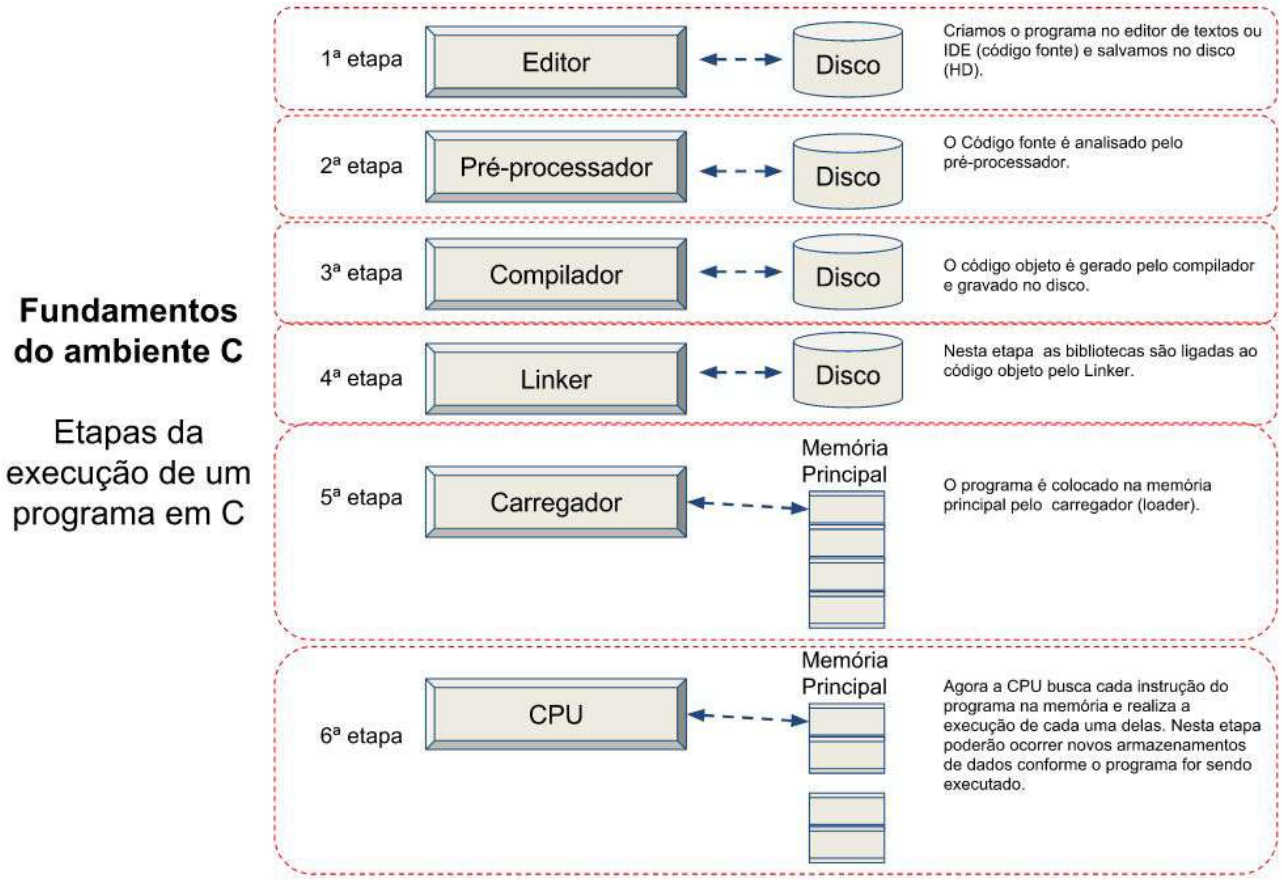
São características da linguagem C:

- É uma linguagem de alto nível, compilada e estruturada.
- É a linguagem do código fonte de diversos sistemas operacionais atuais.
- É uma linguagem muito popular e foi a base para a criação de outras linguagens mais atuais como C++ e Java.

O que é uma linguagem compilada?

Linguagem compilada é aquela que não é executada (interpretada) a partir de seu código fonte diretamente. A linguagem compilada passa por um processo de compilação, realizado pelo compilador, que interpreta o código fonte, gera um código objeto e, neste processo, realiza a ligação de todas as bibliotecas necessárias para o funcionamento do programa. Depois, converte o arquivo final em um executável (código binário) que poderá ser executado na arquitetura (sistema operacional) onde foi compilado. A figura 8 exemplifica o processo de compilação.

Figura 8 - Processo de compilação de um programa



Fonte: Elaborada pelo autor.

O que é uma linguagem estruturada?

Uma linguagem estruturada é aquela que permite o uso de módulos (procedimentos ou funções). Esses módulos são trechos de código reutilizável que serão chamados sempre que necessário.

Observe os dois exemplos abaixo. O código da esquerda está descrito em português estruturado (Lógica - VISUALg) e o da direita em Linguagem C. Podemos observar os dois códigos e começar a perceber as equivalências. Isso mesmo, tudo aquilo que aprendemos em VISUALg pode ser portado (migrado) para a linguagem C. É isso que vamos aprender neste tópico.

Figura 9 - A comparação entre lógica (VISUALg) e a linguagem C

Visual (lógica)

```
1  algoritmo "Calculando o Cubo"
2  // Função :
3  // Autor :
4  // Data : 28/11/2017
5  // Seção de Declarações
6  var
7     x, resultado: inteiro
8  inicio
9  // Seção de Comandos
10    escreval("Cubo de um numero")
11    escreval("Digite o numero:")
12    leia(x)
13    resultado<-x*x*x
14    escreval("Cubo de",x," = ", resultado)
15 fimalgoritmo
```

Linguagem C

```
1  #include <stdio.h>
2
3  int main() {
4   int x, resultado;
5   printf ("Cubo de um numero \n ");
6   printf ("Digite o numero:");
7   scanf ("%d", &x);
8   resultado=x*x*x;
9   printf ("\n Cubo de %d = %d \n",x,resultado);
10  return(0);
11 }
12
```

Fonte: Elaborada pelo autor.

A Linguagem C possui 32 palavras reservadas pelo padrão ANSI.

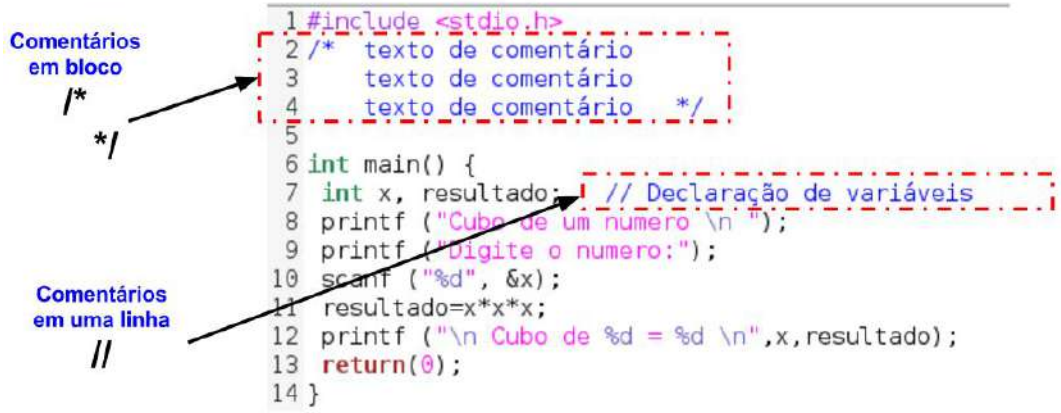
Palavras reservadas - Linguagem C - Padrão ANSI					
1	auto	12	else	23	long
2	break	13	typedef	24	register
3	case	14	while	25	unsigned
4	char	15	enum	26	return
5	const	16	extern	27	short
6	switch	17	float	28	signed
7	volatile	18	for	29	sizeof
8	continue	19	goto	30	static
9	default	20	union	31	struct
10	do	21	if	32	void
11	double	22	int	-	-

ANSI significa (American Standards Institute) - Instituto Americano de Padronização. Esta organização busca facilitar a padronização dos trabalhos dos seus membros.

Como fazer comentários no código em C?

A aplicação de comentários em C pode ser feita tanto por linhas como por blocos.

Figura 10 - Comentários



Fonte: Elaborada pelo autor.

4.1.2 Estrutura Básica de um programa em C

Um programa em C é composto por um conjunto de funções. A função pela qual o programa começa a ser executado chama-se main. Após cada comando em C, deve-se colocar um ‘;’ (ponto-e-vírgula). Um programa em C deve ser identado para que possa ser lido com mais facilidade.

Um programa básico em C possui os seguintes elementos:

```
#include <stdio.h>  int main
()
{
printif();“Olá mundo”);
return (0);
}
```

O #include especifica uma biblioteca chamada pelo código C. A linguagem C possui um conjunto de bibliotecas com instruções já prontas que permitem ao desenvolvedor não necessitar escrever códigos já prontos.

O pequeno código da página anterior apenas especifica a função main(). Ela é um conjunto de instruções que o programa executará em primeiro lugar. Inicialmente, colocaremos todos os códigos dentro de main. Não se preocupe, pois depois poderemos criar outras funções. O comando printf permite que seja exibido na tela o texto que está entre aspas. Todo comando interno deve ser finalizado com ponto-e-vírgula (;).

O arquivo com o código deve ser salvo com o nome desejado pelo desenvolvedor e com a extensão ".c". Pronto. Agora basta compilar e executar o código.

Vamos colocar a mão no código!

Escolha o seu compilador C e digite o código abaixo para testar o funcionamento. Experimente fazer modificações no programa.

```
1 #include <stdio.h>
2
3 int main() {
4  int x, resultado;
5  printf ("Cubo de um numero \n ");
6  printf ("Digite o numero:");
7  scanf ("%d", &x);
8  resultado=x*x*x;
9  printf ("\n Cubo de %d = %d \n",x,resultado);
10 return(0);
11 }
12
```

Lembre-se!
A função main será sempre executada em primeiro lugar. Ela é a função principal da linguagem C.

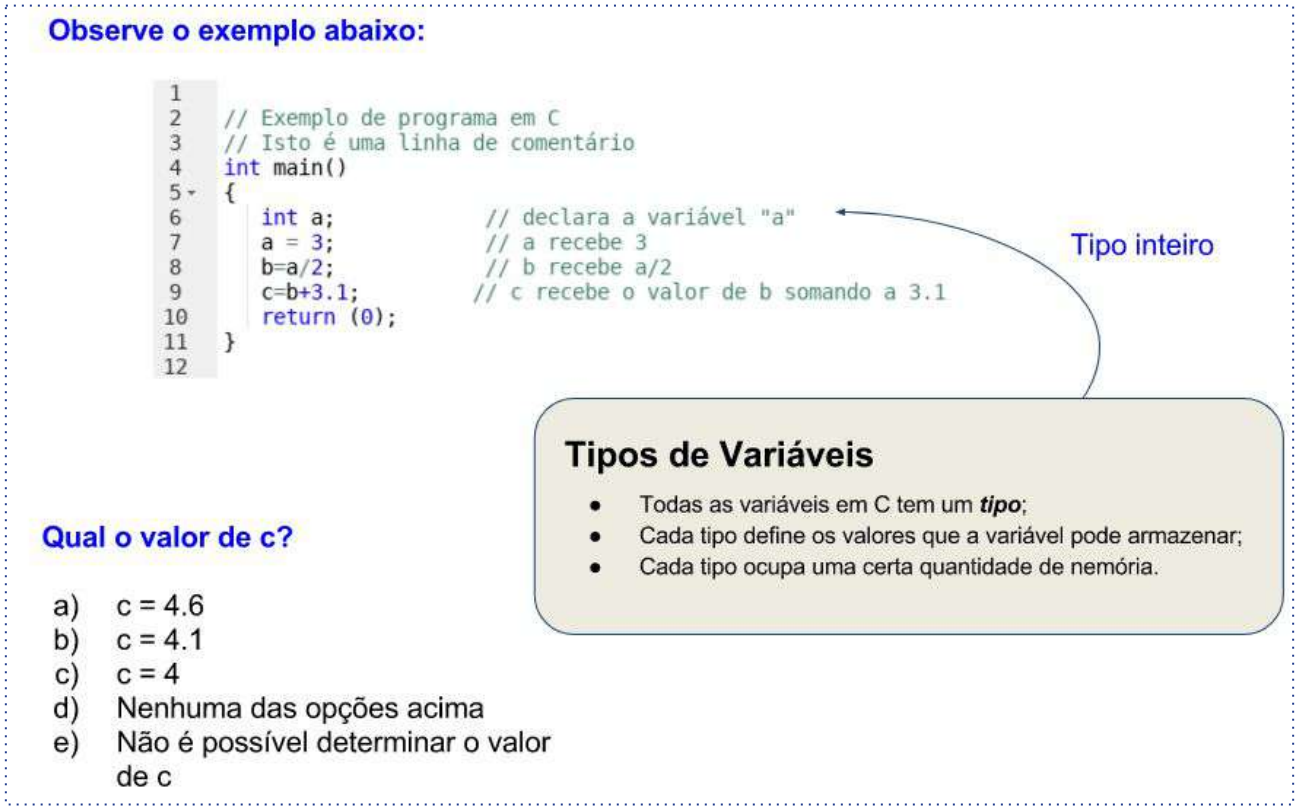
4.1.3 Variáveis, entrada e saída de dados

Aprendemos que Variáveis (ou Constantes) são espaços de memória, nos quais o computador armazena um determinado valor para uso posterior, normalmente na execução de um algoritmo ou programa. As linguagens de programação possuem tipos de dados que estão associados a variáveis e constantes. Algumas linguagens definem o tipo de dado da variável no momento da atribuição e não precisam da definição de tipo prévia na declaração da variável. A linguagem C precisa que todas as variáveis sejam declaradas!

Observe o trecho de código apresentado na Figura 11. Qual o valor da variável c?

Para responder a esta pergunta, teste o código no compilador. Digite o código e teste.

Figura 11 - Exemplos de variáveis



Fonte: Elaborada pelo autor.

Se você testou o código irá perceber que ocorreu um erro de compilação. Logo, a alternativa correta é a letra “e”. Mas o que aconteceu? Se você testou o código, terá observado que o compilador gerou um código de erro e que a explicação apresentada é:

gcc version 4.6.3

```
main.c: In function 'main':
main.c:8:4: error: 'b' undeclared (first use in this function)
   b=a/2;          // b recebe a/2
   ^
main.c:8:4: note: each undeclared identifier is reported only once for each function it appears in
main.c:9:4: error: 'c' undeclared (first use in this function)
   c=b+3.1;        // c recebe o valor de b somando a 3.1
   ^

exit status 1
```

Duas mensagens do compilador estão destacadas. As duas mensagens indicam que as variáveis b e c não foram declaradas e, por isso, não podem ser utilizadas pela primeira vez no código.

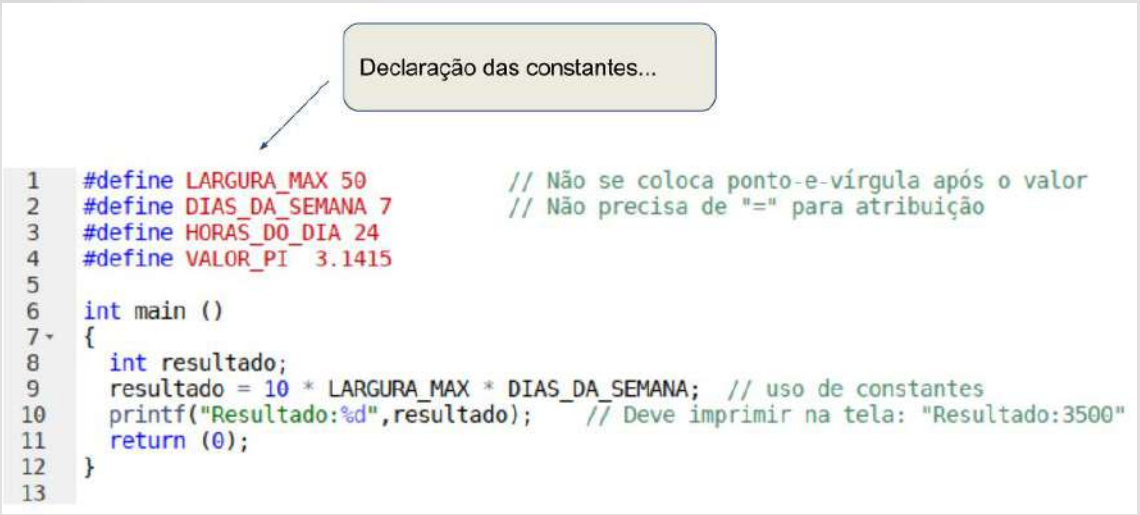
Constantes

As constantes são armazenadas na memória e não modificam seus valores durante a execução do programa. A criação de uma constante é feita com o comando #define, colocado no início do programa-fonte.

```
#define LARGURA_MAX 50        // Não se coloca ponto-e-vírgula após o valor
#define DIAS_DA_SEMANA 7      // no final da linha
#define HORAS_DO_DIA 24
#define VALOR_PI 3.1415
```

Vamos Refletir?

Digite o código e teste.



Variáveis

As variáveis são espaços de armazenamento na memória para um determinado tipo de dado. Toda variável possui um nome (seu identificador) e um tipo (tipo de dado armazenado). Toda variável deve ser explicitamente declarada (identificada por um nome e um tipo). Elas também podem ser declaradas em conjunto.

```
int x;    //declara uma variável do tipo int
float a,b,resultado;   //declaração de várias variáveis do tipo float
float c=10;    //declara e atribui valor a variável do tipo float
a=10;    //atribui valor a uma variável do tipo int
x=5.0;   //atribui valor a uma variável do tipo float
```

Vamos Refletir?

Digite o código e teste.

```c
#include <stdio.h>

int main ()
{
  int x;    //declara uma variável do tipo int
  int y;
  float a,b,resultado;   //declara várias variáveis do tipo float
  float c=10;
  a=10;    //atribui valor a uma variável do tipo int
  b=20;
  x=5.0;   //atribui valor a uma variável do tipo float
  y=5;

  // atribui valor a uma variável a partir
  // do valor de outra variável
  resultado = a+b+c+x+y;

  printf("Resultado:%f",resultado);    // Imprime na tela: "Resultado:50.000000"
  return (0);
}
```

Variáveis só conseguem armazenar valores de um mesmo tipo com que foram declaradas.

```
int x;   //declara uma variável do tipo int  (inteiro)
x=4.3   //a variável x armazenará apenas o valor 4
```

ATENÇÃO: para evitar problemas com acesso aos conteúdos indevidos (lixo por variável não inicializada), uma variável deve ter um valor inicial sempre definido.

Uma variável pode receber valor enquanto é definida (declarada) ou depois de declarada.

```
int x=10, y=15;   //declara e atribui valor
float c=6.7   //declara e atribui valor

int d   //declara
d=5   //atribui valor
```

Observe o exemplo abaixo que indica a necessidade de sempre se declarar e inicializar as variáveis. Neste caso, por sorte, a variável b contém "zero", mas como não foi inicializada poderia conter lixo!

Vamos Refletir?

Digite o código e teste.

```c
#include <stdio.h>

int main ()
{
  int a,b,resultado;   //declaração de várias variáveis do tipo int
  a=10;    //atribui valor a uma variável do tipo int

  // atribui valor a uma variável a partir
  // do valor de outra variável
  resultado = a+b;

  printf("Resultado:%d \n",resultado);    // Imprime na tela: "Resultado:10"
  //De onde saiu o valor de b???
  printf("b:%d \n",b);    // Imprime na tela: "b:0"

  return (0);
}
```

De onde saiu a variável "b"?

4.1.4 Operadores aritméticos e lógicos

Operadores

Empregamos operadores para realizar diversas operações. Com os operadores podemos realizar operações matemáticas, de comparação, lógicas e até mesmo em nível de bits entre variáveis.

Operadores aritméticos

Consideramos OPERADORES ARITMÉTICOS aqueles que atuam (operam) números, quer sejam valores, variáveis, constantes ou até chamadas de funções ou expressões que resultam em valores numéricos.

Geralmente são utilizados em conjunto com o operador de atribuição!

Figura 12 - Operadores Aritméticos

Operador	Significado	Exemplo
+	Adição de dois valores	a = x + y
-	Subtração de dois valores	a = x – y
*	Multiplicação de dois valores	a = x * y
/	Quociente de dois valores	a = x / y
%	Resto de uma divisão	a = x % y

Precedência dos operadores (ordem de execução)

Avaliado antes (esquerda para direita)

- (sinal de menos) , * , / , % , + , - (subtração)

Fonte: Elaborada pelo autor.

Vamos refletir?

Digite o código e teste.

```c
#include "stdio.h"

int main(void) {

  int a;
  double b,c;
  // a casa decimal em C é feita pelo "." (ponto)

  a=3.5;  // variável a recebe o valor 3 (ela é inteira - int)
  b=a/2.0;  // variável b recebe o valor 1.5 (ela é decimal - double)
  c=1/3+b;  // variável c recebe o valor 1.5 (a divisão 1/3 está sendo feita por inteiros)

  printf("a=%i \n",a);
  printf("b=%f \n",b);
  printf("c=%f \n",c);
  return 0;

  // O correto seria utilizar
  // c=1.0/3.0+b;
}

```

```
gcc version 4.6.3

a=3
b=1.500000
c=1.500000
```

Vamos refletir?

Digite o código e teste.

O operador de módulo, “%” aplica-se apenas para inteiros.

```c
#include "stdio.h"

int main(void) {

  int x,a;

  x=12;  // variável "x" recebe o valor 12
  a=x%2;  // variável "a" recebe o valor do resto da divisão de "x" por 2 (recebe 0 = par)

  printf("a=%i \n",a);

  return 0;
}
```

```
gcc version 4.6.3

a=0
```

Operadores de atribuição (= , += , -= , *= , /= , %=)

A atribuição é considerada pela linguagem C como uma expressão. Atribuir um valor a uma variável é armazenar este valor em seu endereço de memória.

```
i = 10;   // atribui o valor de 10 a variável i
i += 2;   // equivalente a   i = i +2;
x *= y + 1;  // equivalente a    x = x * ( y + 1 );
```

Vamos refletir?

Digite o código e teste.

```c
#include "stdio.h"

int main(void) {

  int x,y,i;

  x=12;  // variável "x" recebe o valor 12
  y=x+1;  // variável "y" recebe o valor de "x" mais 1 ( y recebe 13)

  i=x;   // variável "i" recebe o valor de x (i recebe 12)
  i += 2;  // variável "i" recebe o valor de i + 2 (i recebe 14)

  printf("x= %i * (%i + 1)  \n",x,y);

  x *= y + 1; // variável "x" recebe o valor de (x * (y+1)) (i recebe 156+12 = 168)

  printf("x=%i \n",x);

  return 0;
}
```

```
gcc version 4.6.3

x= 12 * (13 + 1)
x=168
```

Operadores de incremento (++) e decremento (--)

Este operador incrementa (acresce) ou decrementa (decresce) de uma unidade o valor da variável que foi empregado.

O incremento pode ser colocado antes ou depois da variável que foi utilizada.

```
n++;   //n está sendo incrementado em 1.  Igual  a n = n+1;
n--;   //n está sendo decrementado em 1.  Igual  a n = n-1;
```

Vamos refletir?

Digite o código e teste.

```c
#include "stdio.h"

int main(void) {

  int x,a,b,n;

  n=5;  // variável "n" recebe o valor 5
  x = n++;    //x recebe o valor de n (5), e depois n é incrementado em 1 (6)
  printf("n=%i   x=%i \n",n,x);

  x = ++n;    //n recebe o incremento de 1 (7), depois x recebe o valor de n (7)
  printf("n=%i   x=%i \n",n,x);

  a = 3;
  b = a++*2;    //b recebe o valor de a*3 (recebe 6), depois a é incrementado em 1 (vale 4)
  printf("a=%i   b=%i \n",a,b);

  return 0;
}
```

```
gcc version 4.6.3
n=6   x=5
n=7   x=7
a=4   b=6
```

Operadores relacionais (> , >= , < , <= , == , !=)

Os operadores relacionais realizam a comparação entre valores (magnitude), indicando se são valores iguais, diferentes, maiores ou menores. Na linguagem C, não existem valores booleanos (falso ou verdadeiro) e o resultado retornará 0 ou 1, representando os valores booleanos.

Figura 13 - Operadores relacionais

Operador	Significado	Exemplo
>	Maior do que	x > 5
>=	Maior ou igual a	x >= y
<	Menor do que	x < 5
<=	Menor ou igual a	x <= z
==	Igual a	x == 0
!=	Diferente de	x != y

ATENÇÃO! O símbolo de atribuição "=" é diferente, muito diferente, do operador relacional de igualdade "=="

Fonte: Elaborada pelo autor.

Vamos refletir?

Digite o código e teste.

```c
#include "stdio.h"

int main(void) {

  int a,b,c,d;

  c=23;  // variável "c" recebe o valor 23
  d=c+4;   //d recebe o valor de c (23) mais 4 (27)

  a = (c<20);   // a recebe 0 como resultado da comparação (falso = false = 0)
  b = (d>c);    // b recebe 1 como resultado da comparação (verdadeiro = true = 1)

  printf("a=%i   b=%i \n",a,b);

  return 0;
}
```

```
gcc version 4.6.3

a=0   b=1
```

Operadores lógicos (&& , || , !)

A realização das operações com operadores lógicos é sempre realizada da esquerda para a direita. A avaliação da expressão só termina quando seu valor pode ser conhecido (determinado). São utilizados para modelar situações em que não podemos utilizar apenas operadores aritméticos ou relacionais.

Por exemplo, a expressão matemática 0 < x < 10 indica que o valor de x deve ser maior do que 0 (zero) e também menor do que 10. Equivale a (x > 0) && (x < 10) .

Vamos refletir?

Operadores lógicos (&&, ||, !)

Operador	Significado	Exemplo
&&	Operador **E**	(x > 0) && (x < 10)
||	Operador **OU**	(a == 'F') || (b != 32)
!	Operador **NEGAÇÃO**	!(x == 10)

Tabela verdade

Os termos a e b representam o resultado de duas expressões relacionais

a	b	!a	!b	a && b	a || b
0	0	1	1	0	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	1	1

Vamos refletir?

Digite o código e teste.

```c
#include "stdio.h"

int main(void) {

  int a,b,c,d;

  c=23;  // variável "c" recebe o valor 23
  d=c+4;   //d recebe o valor de c (23) mais 4 (27)

  a = (c<20) || (d>c);  //  (0)ou(1) = (1) (verdadeiro)
  // a recebe 1 como resultado da comparação (verdadeiro = true = 1)

  b = (c<20) && (d>c);  //  (0)e(1) = (0) (falso)
  // b recebe 0 como resultado da comparação (falso = false = 0)

  printf("a=%i   b=%i \n",a,b);

  return 0;
}
```

```
gcc version 4.6.3

a=1   b=0
```

sizeof

Retorna o número de bytes ocupados por um tipo.

Conversão de tipo

A conversão de tipo é automática na avaliação de uma expressão. A conversão de tipo pode ser solicitada (requisitada) explicitamente.

Vamos refletir?

Digite o código e teste.

```c
#include "stdio.h"

int main(void) {

  int a,b,g,h;

  a = sizeof(float);    //armazena 4 em a (tamanho do float)
  printf("a=%i \n",a);

  b = sizeof(double);    //armazena 8 em b (tamanho do double)
  printf("b=%i \n",b);

  float c = 3;     //armazena 3.0 em c (converte automaticamente o int(3) para float(3.0)
  printf("c=%f \n",c);

  g = (int)3.5;       //3.5 é convertido e arredondado para "int" e armazendo em g
  printf("g=%i \n",g);

  h = (int)3.5 % 2;       //3.5 é convertido e arredondado para "int"
  // armazena 1 em h pois (3 % 2) tem resto 1
  printf("h=%i \n",h);

  return 0;
}
```

```
gcc version 4.6.3

a=4
b=8
c=3.000000
g=3
h=1
```

A função printf (saída)

É uma função utilizada para a saída de valores na tela (exibir) de acordo com um determinado formato.

A função printf deve receber pelo menos dois parâmetros separados por vírgula:

- um string de formato que define, através de caracteres especiais, os tipos dos dados a serem impressos e suas posições na linha de impressão;
- um dado a ser impresso.

Sintaxe: printf("%string_formato", dado);

Vamos refletir?

Digite o código e teste.

```c
#include "stdio.h"  // Necessário para usar a função printf
                    // A função printf exibe um ou mais dados na tela
int main(void) {

  // printf ("%tipo_de_saida",expressão)

  // printf ("%tipo1 %tipo2", expressão1, expressão2)

  // texto misturado com os valores das expressões ou variáveis
  // printf ("texto %tipo1 texto", expressão)

  printf("%s","Isto é uma string ....\n"); // note o '\n' no final da string;
  printf("%s","Outra string ....");
  printf("%s","Terceira string\n");

  int a, b;
  a = 10;
  b = 20;
  printf("Valor de a=%d e b=%d \n",a,b);

  return 0;
}
```

```
gcc version 4.6.3

Isto é uma string ....
Outra string ....Terceira string
Valor de a=10 e b=20
```

Especificadores de formato de saída (string de formato)

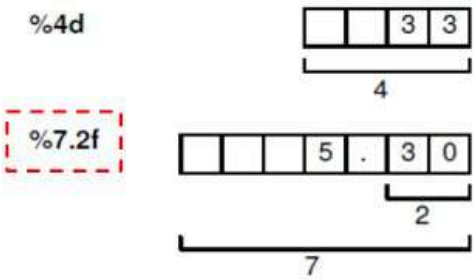
Algumas vezes, necessitamos formatar a saída de dados na tela de forma mais adequada. A função printf possui alguns especificadores de formato que podemos empregar.

Figura 14 - Especificadores de formato de saída (string de formato)

Alguns tipos de saída	
%c	escrita de um caractere (**char**)
%d ou %i	escrita de números inteiros (**int** ou **char**)
%u	escrita de números inteiros sem sinal (**unsigned**)
%f	escrita de número reais (**float** ou **double**)
%s	escrita de vários caracteres
%p	escrita de um endereço de memória
%e ou %E	escrita em notação científica

Para formatar de maneira diferente usar-se, junto com o ***%f*** uma especificação de quantas casas decimais se deseja que o número tenha. Especifica-se também o número total de caracteres do número a ser impresso.

Por exemplo: ***%7.2f*** especifica que se quer imprimir um **float** com **2 casas decimais** e com um **tamanho total** de **7** caracteres no total.



Fonte: Elaborada pelo autor.

Especificação de caracteres de escape (\ )

São constantes predefinidas. Elas permitem o envio de caracteres de controle não gráficos para dispositivos de saída. Isso também é empregado para realizar uma formatação diferenciada ou controlar funções e impressão de caracteres especiais.

Figura 15 - Especificação de caracteres de escape (\ )

Código	Comando
\a	som de alerta (bip)
\b	retrocesso (backspace)
\n	nova linha (new line)
\r	retorno de carro (carriage **return**)
\v	tabulação vertical
\t	tabulação horizontal
\'	apóstrofe
\"	aspa
\\	barra invertida (backslash)
\f	alimentação de folha (form feed)
\?	símbolo de interrogação
\0	caractere nulo (cancela a escrita do restante)

Fonte: Elaborada pelo autor.

Vamos refletir?

Digite o código e teste.

```c
#include <stdio.h>
int main()
{
    float NotaDaP1, NotaDaP2, Media, Numero;
    printf("Hello Word \n");
    printf("Hello\nWord \n");
    printf("Hello \\ Word \n");
    printf("\"Hello Word\"\n");
    printf("\n");

    NotaDaP1 = 6.6;  // Atribuição do Valores das médias
    NotaDaP2 = 8.2;
    Media = (NotaDaP1 + NotaDaP2) / 2.0;
    printf("Média Final : %6.3f", Media);
   // No momento da execução sinal %6.3f vai ser substituído
   // pelo valor da variável Media
         // Média Final :  7.400

    Numero = -2.5;
    printf("\n\n\n");
    printf("1234567890\n");
    printf("%7f\n", Numero);
    printf("%7.0f\n", Numero);
    printf("%7.3f\n", Numero);
    printf("%8.3f\n", Numero);
    printf("%9.3f\n", Numero);
    printf("\n");
    printf("%8.4f\n", Numero);
    printf("%8.1f\n", Numero);
    printf("%6.12f\n", Numero);
    return(0);
}
```

```
gcc version 4.6.3
Hello Word
Hello
Word
Hello \ Word
"Hello Word"

Média Final :  7.400


1234567890
-2.500000
     -2
 -2.500
  -2.500
   -2.500

 -2.5000
    -2.5
-2.500000000000
```

A função scanf (entrada)

É uma função utilizada para a entrada de valores por meio do teclado (receber) de acordo com um determinado formato que será armazenado em uma variável.

A função scanf deve receber, pelo menos, dois parâmetros separados por vírgula:

- um string de formato que define, através de caracteres especiais, os tipos dos dados a serem lidos e armazenados;
- um variável para armazenar o valor (o **&** indica que o valor lido será colocado no endereço de memória da variável utilizada).

Sintaxe: **scanf**("%string_formatos",&variáveis);

Especificadores de formato de entrada (string de formato)

A entrada também pode ser formatada de forma mais organizada com o emprego de especificadores de formato de entrada.

Figura 16 - Especificadores de formato de entrada

Alguns tipos de saída	
%c	leitura de um caractere (**char**)
%d ou %i	leitura de números inteiros (**int** ou **char**)
%f	leitura de número reais (**float** ou **double**)
%s	leitura de vários caracteres

Fonte: Elaborada pelo autor.

Vamos refletir?

Digite o código e teste.

```c
#include <stdio.h>
int main () {
   char str1[20], str2[30];
   printf("Enter name: ");
// scanf("%[A-Z a-z]",str1);  //limitar os caracteres lidos - todos os caracteres e o espaço
   scanf("%[^\n]s",str1);   //forçar o scanf ler a string até encontrar o [enter]
   setbuf(stdin, NULL);
   printf("Enter your website name: ");
   scanf("%[^\n]s",str2); // forçar o scanf ler a string até encontrar o [enter]
// scanf("%[A-Z a-z]",str2); //limitar os caracteres lidos - todos os caracteres e o espaço
   printf("Entered Name: %s\n", str1);
   printf("Entered Website:%s\n", str2);

   return(0);
}
```

```
ronald@debian: ~/testes.c
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
ronald@debian:~/testes.c$ mcedit scanf_char.c

ronald@debian:~/testes.c$ gcc -o scanf_char.bin scanf_char.c
ronald@debian:~/testes.c$ ./scanf_char.bin
Enter name: Ronald Costa
Enter your website name: Tudinho Livre
Entered Name: Ronald Costa
Entered Website:Tudinho Livre
ronald@debian:~/testes.c$
```

Vamos refletir?

Digite o código e teste.

```
1
2   // Exemplo com strings
3   #include <stdio.h>
4   #include <string.h>  // arquivo de cabeçalho para trabalhar com strings
5
6   int main()
7   {
8      char Nome[30];  // declara uma string que poderá armazenar até 29 caracteres !!
9      strcpy(Nome, "JOSÉ DA SILVA");  // atribui "Jose da Silva" para a variável Nome
10     printf("O funcionário %s foi tranferido", Nome);  // no lugar de %s aparecerá o
11                  // conteúdo da variável Nome
12
13     int num1, num2;
14     printf("\n");
15     printf("Insira dois numeros: ");
16     scanf("%d %d", &num1, &num2);
17     printf("\n");
18     printf("Você digitou: '%d' e '%d'", num1, num2);
19
20     return(0);
21  }
```

```
gcc version 4.6.3

O funcionário JOSÉ DA SILVA foi tranferido
Insira dois numeros:  10
 20

Você digitou: '10' e '20'
```

4.1.5 Estruturas de sequência, seleção e repetição

Estruturas de Seleção

Como fazer um programa tomar decisões?

Em alguns momentos, precisamos tomar decisões e seguir caminhos de execução diferentes na execução de um programa. Avalie a situação abaixo:

Imagine que nosso programa tem uma função para "qualificar" a temperatura:

- **se** a temperatura for menor do que (<20ºC), **então** está frio;
- **se** a temperatura estiver entre 20ºC e 30ºC, **então** está agradável;
- **se** a temperatura for maior do que 30ºC (>30ºC), **então** está quente;

Comando if e if else

O comando if ou if else são os comandos básicos para codificar uma tomada de decisão. A estrutura avalia uma condição inicial e de acordo com o resultado lógico dessa expressão toma a decisão se executa os seus blocos ou o bloco. Observe a sintaxe básica desse comando.

A forma geral do comando if é:

```
if (condição) {
< bloco de comando executados se VERDADEIRO >
}
```

A forma geral do comando if else é:

```
if (condição) {

< bloco de comando executados se VERDADEIRO >
} else {
< bloco de comando executados se FALSO >
}
```

Vamos refletir?

Digite os códigos e teste.

```
1
2   // Exemplo com strings
3   #include <stdio.h>
4
5   int main()
6   {
7     int num;
8     printf("Digite um número: \n");
9     scanf("%d",&num);
10
11    if (num > 10) {
12      printf("O número é maior que 10. \n");
13    }
14
15     return(0);
16  }
17
```

```
gcc version 4.6.3
>
Digite um número:
 11
O número é maior que 10.
>
```

```
1
2   // Exemplo com strings
3   #include <stdio.h>
4
5   int main()
6   {
7     int num;
8     printf("Digite um número: \n");
9     scanf("%d",&num);
10
11    if (num > 10) {
12      printf("O número É MAIOR que 10. \n");
13    }
14    else{
15      printf("O número NÃO É MAIOR que 10. \n");
16    }
17
18     return(0);
19  }
20
```

```
gcc version 4.6.3
>
Digite um número:
 10
O número NÃO É MAIOR que 10.
>
```

Imagine que nosso programa tem uma função para “qualificar” a temperatura:

- **se** a temperatura for menor do que (<20ºC), **então** está frio;
- **se** a temperatura estiver entre 20ºC e 30ºC, **então** está agradável;
- **se** a temperatura for maior do que 30ºC (>30ºC), **então** está quente;

Vamos refletir?

Digite o código e teste.

```
1
2   // Exemplo com strings
3   #include <stdio.h>
4
5   int main()
6   {
7     int temp;
8     printf("Digite a temperatura: \n");
9     scanf("%d",&temp);
10
11    if (temp < 10) {
12      printf("Temperatura muito fria. \n");
13    }
14    else if (temp < 20){
15      printf("Temperatura fria. \n");
16    }
17    else if (temp < 30){
18      printf("Temperatura agradável. \n");
19    }
20    else{
21      printf("Temperatura quente. \n");
22    }
23   return(0);
24  }
25
```

```
gcc version 4.6.3

Digite a temperatura:
 31
Temperatura quente.
```

Estrutura de bloco

A declaração de variáveis só pode ocorrer no início do corpo de uma função (principal ou não) ou de um bloco.

- **Escopo**: define onde e quando a variável pode ser usada.
- **Escopo global**: Fora de qualquer definição de função. Tempo de vida é o tempo de execução do programa
- **Escopo local**: Bloco ou função

O escopo de uma variável dentro de um bloco de código ou função é restrito a este espaço.

Uma variável declarada dentro de um bloco existe apenas dentro desse bloco (ou função). Ao término do bloco (ou função) ela deixa de existir.

Vamos refletir?

Digite o código e teste.

```c
#include <stdio.h>

//declaração de variáveis globais


// ---------------- Funçao main()-------------------
int main(void)
{
  //declaração das variáveis locais da main()

   return(0);
}
// -------------------------------------------------


void funcao1(variáveis locais de parâmetros)
{
   // declaração das variáveis locais da função1

   return;
}
```

Vamos refletir?

Digite o código e teste.

```c
#include<stdio.h>

//declaração de variáveis globais
float media, n1, n2;

//protótipo da função entrada
void entrada(void);

// ---------------- funçao main()--------------------
int main(void)
{
  //variável local
  char resposta;

  do
  {
    //chamada da função p/ entrada das notas
    entrada();
    //usando variáveis globais: media,n1,n2
    media = (n1 + n2) / 2;
    printf("\nMedia do aluno: %.2f\n", media);
    printf("\nDeseja calcular outra media? (s/n)");
    scanf("%c%*c",&resposta);
    /* No código acima o scanf irá ler um primeiro caráter e depois vai suprimir o seguinte (que
      normalmente é o ENTER), quer dizer que tal caráter não será armazenado no buffer de
      entrada.  */
  }
  while(resposta != 's');
  return(0);
}
// --------- fim da função main() -----

//função entrada de dados
//usa as variáveis globais n1 e n2
void entrada(void)
{
  printf("\nDigite a primeira nota: ");
  scanf("%f", &n1);
  printf("Digite a segunda nota: ");
  scanf("%f", &n2);
  return;
}
```

```
gcc version 4.6.3

Digite a primeira nota:  10
Digite a segunda nota:  20

Media do aluno: 15.00

Deseja calcular outra media? (s/n) n
```

Expressão Condicional

A expressão condicional não precisa ser uma expressão no sentido convencional. Pode ser o valor de uma variável sendo avaliada pelo compilador.

Uma expressão condicional é a avaliação de uma condição, variável ou valor pelo compilador que usa seu valor de retorno para tomar uma decisão.

Uma variável sozinha pode ser uma “expressão” e o compilador pode usar seu próprio valor para avaliar uma condição.

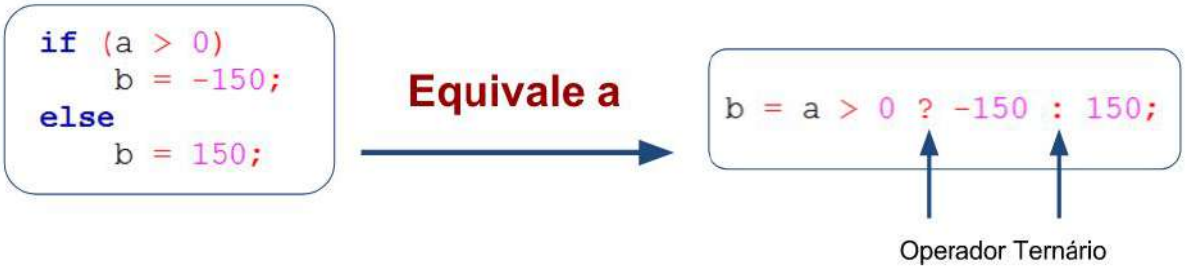
Figura 17 - Expressão condicional

Fonte: Elaborada pelo autor.

Operador Ternário (? :)

A expressão condicional ” ? : “ é uma simplificação do if-else utilizada tipicamente para testar condições.

Figura 17.1 - Expressão condicional



Fonte: Elaborada pelo autor.

Vamos refletir?

Digite o código e teste.

```c
#include "stdio.h"

int main(void) {

  int x,y,z;

  printf("\n ------ IF-ELSE ------ \n");
  printf("\n Digite o valor de x: ");
  scanf("%d", &x);
  printf("\n Digite o valor de y: ");
  scanf("%d", &y);
  if(x>y){
    z = x;
  }
  else{
    z = y;
  }
  printf("Maior = %d\n",z);

  printf("\n ------ OPERADOR-TERNÁRIO ------ \n");
  printf("\n Digite o valor de x: ");
  scanf("%d", &x);
  printf("\n Digite o valor de y: ");
  scanf("%d", &y);;
  z = x > y ? x : y ;
  printf("Maior = %d\n",z);

  return 0;
}
```

```
gcc version 4.6.3
>

 ------ IF-ELSE ------

 Digite o valor de x:  10

 Digite o valor de y:  20
Maior = 20

 ------ OPERADOR-TERNÁRIO ------

 Digite o valor de x:  10

 Digite o valor de y:  20
Maior = 20
>
```

Comando “ switch ”

A estrutura de seleção “switch” seleciona uma opção, dentre os diversos casos disponíveis, de acordo com a avaliação da expressão (valor) apresentada. Em linguagem C, a opção (OPn) deve ser um inteiro ou caractere.

Sintaxe:

```
switch( expressão )
  {
    case op1: bloco de comandos; break;
    case op2: bloco de comandos; break;
    . . .
    default: bloco de comandos; break;
  }
```

Equivale ao CASO que aprendemos em português estruturado (VISUALg). A declaração default é opcional e será executada apenas se a expressão que está sendo testada não for igual a nenhuma das constantes presentes nos case.

Vamos refletir?

Digite o código e teste.

Estruturas de Seleção (switch)

```c
#include "stdio.h"

int main(void) {

  float num1, num2;
  char op;
  printf("\nCalculadora de 4 operações \n");

  printf("\nDigite: numero   op   numero\n");
  scanf(" %f %c %f", &num1, &op, &num2);

  switch (op){
    case'+':  printf("=%f\n",num1+num2); break;
    case'-':  printf("=%f\n",num1-num2); break;
    case'*':  printf("=%f\n",num1*num2); break;
    case'/':  printf("=%f\n",num1/num2); break;
    default:  printf("Operador Inválido\n"); break;
  }
  return 0;
}
```

```
gcc version 4.6.3

Calculadora de 4 operações

Digite: numero   op   numero
 2
 +
 2
=4.000000
```

Estruturas de Repetição

- Como construir laços de repetição em meus programas?

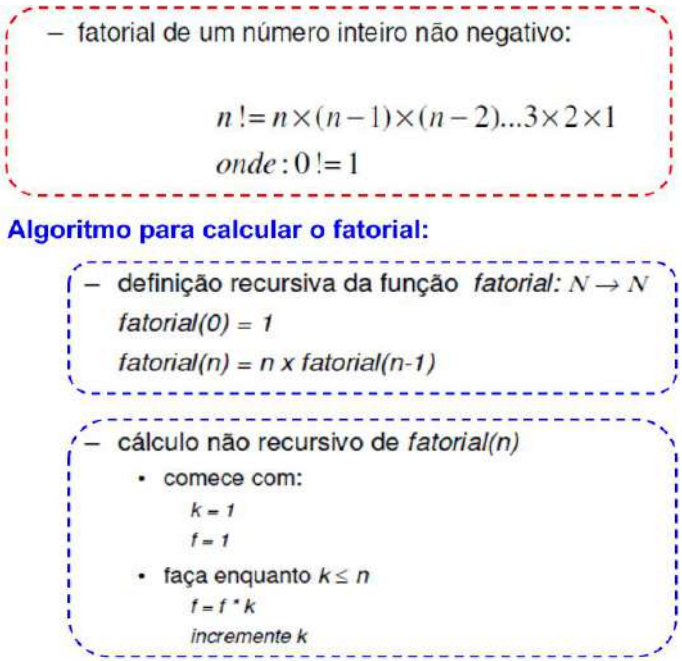
Agora que começa a parte divertida da programação! Vamos pensar utilizando exemplos práticos! Como calculamos o fatorial de um número?

Para começar a compreender como fazemos esse cálculo utilizando programação, é necessário relembrar primeiro como realizamos o cálculo do fatorial de um número matematicamente. Lembre-se é preciso primeiro entender a lógica do problema e sua resolução para depois realizar a codificação.

- Lembrando os conceitos de FATORIAL

Importante relembrar esse processo de cálculo de um fatorial, pois o utilizamos em vários exemplos de estruturas de repetição.

Figura 18 - Como resolver um fatorial: matematicamente e logicamente.



Fonte: Elaborada pelo autor.

Comando “ while ”

Estrutura de repetição que realiza a execução do bloco de comandos de código, enquanto o resultado da expressão (condição) avaliada for verdadeira. É uma repetição com teste no início da estrutura.

Sintaxe:

```
while ( condição )
{
<bloco de comandos>
}
```

Equivale ao ENQUANTO que aprendemos em português estruturado (VISUALg).

Vamos refletir?

Digite o código e teste.

```c
#include "stdio.h"

int main(void) {

  int k,n;
  long int f = 1;

  printf("\n ------ FATORIAL ------ \n");
  printf("\n Digite um nº inteiro não negativo: ");
  scanf("%d", &n);

  // Calcula o fatorial com WHILE
  k=1;
  while (k <= n){
    printf("Fatorial = %d x",f);
    f = f * k;
    printf("%d = %d \n",k,f);
    k = k + 1;
  }

  return 0;
}
```

```
gcc version 4.6.3
>

 ------ FATORIAL ------

 Digite um nº inteiro não negativo:  5
Fatorial = 1 x1 = 1
Fatorial = 1 x2 = 2
Fatorial = 2 x3 = 6
Fatorial = 6 x4 = 24
Fatorial = 24 x5 = 120
>
```

Comando “ for ”

Estrutura de repetição que realiza a execução do bloco de comandos de código, começando na inicialização e repetindo com o incremento até que o resultado da expressão (condição) avaliada seja falso.

Sintaxe:

```
for ( inicialização; condição; incremento )
    {
       <bloco de comandos>
    }
```

Se a condição é verdadeira, continua a execução do bloco de comandos, incrementando o valor de inicialização. Após executar o bloco de comandos, testa novamente a condição.

Vamos refletir?

Digite o código e teste.

```c
#include "stdio.h"

int main(void) {

  int k,n;
  long int f = 1;

  printf("\n ------ FATORIAL ------ \n");
  printf("\n Digite um nº inteiro não negativo: ");
  scanf("%d", &n);

  // Calcula o fatorial com WHILE
  k=1;
  while (k <= n){
    printf("Fatorial = %d x",f);
    f = f * k;
    printf("%d = %d \n",k,f);
    k = k + 1;
  }

  return 0;
}
```

```
gcc version 4.6.3
>

 ------ FATORIAL ------

 Digite um nº inteiro não negativo:  5
Fatorial = 1 x1 = 1
Fatorial = 1 x2 = 2
Fatorial = 2 x3 = 6
Fatorial = 6 x4 = 24
Fatorial = 24 x5 = 120
>
```

Vamos refletir?

Digite o código e teste.

```c
#include "stdio.h"

int main(void) {

  int k,n;
  long int f = 1;

  printf("\n ------ FATORIAL ------ \n");
  printf("\n Digite um nº inteiro não negativo: ");
  scanf("%d", &n);

  // Calcula o fatorial com FOR

  for (k = 1; k <= n; k+1){    // Atenção ao ERRO NESTA LINHA (k+1)!
  // Ele gera um loop infinito!  O correto deveria ser k=k+1 ou k++ para incrementar o k
    printf("Fatorial = %d x",f);
    f = f * k;
    printf("%d = %d \n",k,f);
  }

  return 0;
}
```

ERRO !!!
Loop Infinito !!!

```
gcc version 4.6.3


 ------ FATORIAL ------

 Digite um nº inteiro não negativo:  5
Fatorial = 1 x1 = 1
Fatorial = 1 x1 = 1
Fatorial = 1 x1 = 1
Fatorial = 1 x1 = 1
```

Um loop infinito é quando uma estrutura de repetição nunca é finalizada e o programa nunca termina a sua execução. Ficará eternamente executando se não for interrompido manualmente. Isso normalmente ocorre por um erro de lógica na condição de parada ou com a variável de controle dessa estrutura de repetição.

Comando “ do while ”

Estrutura de repetição que realiza a execução do bloco de comandos de código primeiro (uma vez) e depois testa a expressão (condição) avaliada. Se a condição for verdadeira repete o processo novamente (executa e testa). É uma repetição com teste no final da estrutura.

Sintaxe:

```
do
 { <bloco de comandos> }
while ( <condição> );
```

Equivale ao FAÇA - ENQUANTO que aprendemos em português estruturado (VISUALg).

Vamos refletir?

Digite o código e teste.

```c
#include "stdio.h"

int main(void) {

  int k,n;
  long int f = 1;

  printf("\n ------ FATORIAL ------ \n");

  // Calcula o fatorial com DO WHILE
  do{
    printf("\n Digite um nº inteiro não negativo: ");
    scanf("%d", &n);
  }
  while (n<0);

  for (k=1; k<=n; k++)  {
    printf("Fatorial = %d x",f);
    f *= k;
    printf("%d = %d \n",k,f);
  }

  return 0;
}
```

```
gcc version 4.6.3
>

 ------ FATORIAL ------

 Digite um nº inteiro não negativo:  -1

 Digite um nº inteiro não negativo:  5
Fatorial = 1 x1 = 1
Fatorial = 1 x2 = 2
Fatorial = 2 x3 = 6
Fatorial = 6 x4 = 24
Fatorial = 24 x5 = 120
>
```

Comando “ break ”

Comando que realiza a interrupção de um laço de repetição, ou seja, ele termina a execução de uma estrutura de repetição.

Comando “ continue ”

Comando que termina a iteração corrente (execução do momento) e passa para o próximo incremento (volta) da estrutura de repetição.

Vamos refletir?

Digite o código e teste.

```c
#include "stdio.h"

int main(void) {

  int k;
  printf("\nBREAK \n");

  for (k=0; k<10; k++)  {
    if (k==5){
      break;
    }
    printf("%d ,",k);
  }
  printf("\nFIM \n");
  return 0;
}

```

```
gcc version 4.6.3
:

BREAK
0 ,1 ,2 ,3 ,4 ,
FIM
:
```

Vamos refletir?

Digite o código e teste.

```c
#include "stdio.h"

int main(void) {

  int k;
  printf("\nCONTINUE \n");

  for (k=0; k<10; k++)  {
    if (k==5){
      continue;
    }
    printf("%d ,",k);
  }
  printf("\nFIM \n");
  return 0;
}
```

```
gcc version 4.6.3


CONTINUE
0 ,1 ,2 ,3 ,4 ,6 ,7 ,8 ,9 ,
FIM
```

Não se esqueça! A primeira preocupação deve ser em conhecer muito bem o problema, pois assim será possível refletir sobre ele e, desta forma, encontrar uma solução segura e eficaz para a sua resolução.

Saiba mais!

Que tal dar uma espiada em alguns tutoriais de linguagem C?

Achou complicado? É sempre interessante assistir vídeos e explicações diferentes sobre um determinado conteúdo. Isso ajuda a melhorar a compreensão. Vamos espiar um vídeo?

Não desista. Seja persistente! Programe!

Veja o vídeo do Curso de programação em C
para iniciantes (Pixel Tutoriais)

Acesse:

https://www.youtube.com/watch?v=3_kvlGGgfW8&index=1&list=PL65GXbMNWBWZ85GBjUBuG6lwjO8WwihKH

Vamos rever?

As linguagens de programação permitem a criação de programas empregando regras sintáticas e semânticas.

O código fonte de um programa é o conjunto de palavras e códigos, organizado segundo as regras de cada linguagem. O código fonte para ser executado pelo processador do computador, primeiro, deve ser traduzido e compilado para código de máquina (executável).

A Linguagem C foi criada por Dennis M. Ritchie e Brian Kernighan em 1972, nos laboratórios Bell (EUA). Ela foi criada para o desenvolvimento do sistema operacional UNIX, o qual era escrito em Assembly (linguagem de baixo nível). A Linguagem C, assim como a Assembly, também possui instruções de baixo nível, bem próximas à linguagem de máquina.

A linguagem compilada passa por um processo de compilação, realizado pelo compilador, que interpreta o código fonte, gera um código objeto e, neste processo, realiza a ligação de todas as bibliotecas necessárias para o funcionamento do programa. Depois, converte o arquivo final em um executável (código binário) que poderá ser executado na arquitetura (sistema operacional) onde foi compilado.

Uma linguagem estruturada é aquela que permite o uso de módulos (procedimentos ou funções). Esses módulos são trechos de código reutilizável que serão chamados sempre que necessário. Um programa em C é composto por um conjunto de Funções. A função pela qual o programa começa a ser executado chama-se main. Após cada comando em C, deve-se colocar um ";" (ponto-e-vírgula). Um programa em C deve ser identado para que possa ser lido com mais facilidade.

Vamos rever?

Variáveis (ou Constantes) são espaços de memória nos quais o computador armazena um determinado valor para uso posterior, normalmente, na execução de um algoritmo ou programa. As linguagens de programação possuem tipos de dados que estão associados a variáveis e constantes. A linguagem C precisa que todas as variáveis sejam declaradas! A criação de uma constante é feita com o comando #define colocado no início do programa-fonte.

Com os operadores, podemos realizar operações matemáticas, de comparação, lógicas e até mesmo em nível de bits entre variáveis. A função printf (saída) é uma função utilizada para a saída de valores na tela (exibir) de acordo com um determinado formato. A função scanf (entrada) é uma função utilizada para a entrada de valores por meio do teclado (receber) de acordo com um determinado formato que será armazenado em uma variável.

O comando if ou if else são os comandos básicos para codificar uma tomada de decisão. A estrutura avalia uma condição inicial e, de acordo com o resultado lógico dessa expressão, toma a decisão se executa os seus blocos ou bloco. O escopo de uma variável dentro de um bloco de código ou função é restrito a este espaço. Uma variável declarada dentro de um bloco existe apenas dentro desse bloco (ou função). Ao término do bloco (ou função) ela deixa de existir.

Uma expressão condicional é a avaliação de uma condição, variável ou valor pelo compilador que usa seu valor de retorno para tomar uma decisão. Uma variável sozinha pode ser uma “expressão” e o compilador pode usar seu próprio valor para avaliar uma condição.

Vamos rever?

A estrutura de seleção “switch” seleciona uma opção, dentre os diversos casos disponíveis, de acordo com a avaliação da expressão (valor) apresentada.

O comando *while* é uma estrutura de repetição que realiza a execução do bloco de comandos de código enquanto o resultado da expressão (condição) avaliada for verdadeira. É uma repetição com teste no início da estrutura. O comando for é uma estrutura de repetição que realiza a execução do bloco de comandos de código, começando na inicialização e repetindo com o incremento até que o resultado da expressão (condição) avaliada seja falso. O comando do *while* é uma estrutura de repetição que realiza a execução do bloco de comandos de código primeiro (uma vez) e depois testa a expressão (condição) avaliada. Se a condição for verdadeira, repete o processo novamente (executa e testa). É uma repetição com teste no final da estrutura. O *break* é um comando que realiza a interrupção de um laço de repetição, ou seja, ele termina a execução de uma estrutura de repetição. O continue é um comando que termina a interação corrente (execução do momento) e passa para o próximo incremento (volta) da estrutura de repetição.

Sites indicados

1) Linguagem C - IME-USP https://www.ime.usp.br/~slago/slago-C.pdf
2) Linguagem C: Completa e Descomplicada
https://kupdf.net/download/linguagem-c-completa-e-descomplicada_5ae0f496e2b6f5cc553bb08d_pdf
3) Apostila de Introdução à Introdução à Linguagem C
https://edisciplinas.usp.br/pluginfile.php/4211311/mod_resource/content/1/Apostila-de-Introdu%C3%A7%C3%A3o-%C3%A0-Linguagem-C.pdf
4) Apostila de Linguagem C (Conceitos Básicos)
http://www.facom.ufu.br/~gustavo/ED1/Apostila_Linguagem_C.pdf

Audiovisuais indicados

1) Curso de C - eXcript
https://www.youtube.com/playlist?list=PLesCEcYj003SwVdufCQM5FIbrOd0GG1M4
2) Curso - Primeiros passos com a Linguagem C
https://www.youtube.com/playlist?list=PLbEOwbQR9lqxHno2S-IiG9-lePyRNOO_E
3) Curso de Linguagem C (ANSI)
https://www.youtube.com/playlist?list=PLZ8dBTV2_5HTGGtrPxDB7zx8J5VMuXdob
4) Linguagem C - Curso de Programação Completo para Iniciantes e Profissionais
https://www.youtube.com/playlist?list=PLrqNiweLEMonijPwsHckWX7fVbgT2jS3P

“Nunca abandone o 3F (Foco, Força e Fé)!”
Prof. Ronald Costa

Questões de autoaprendizagem

a) Desenvolver um algoritmo que leia um número inteiro e verifique se o número é divisível por 5 e por 3 ao mesmo tempo.

b) Desenvolver um algoritmo para ler o número de uma sala de aula, sua capacidade e o total de alunos matriculados e imprimir uma linha mostrando o número da sala, sua capacidade, o número de cadeiras ocupadas e sua disponibilidade indicando se a sala está lotada ou não.

c) Fazer um algoritmo que leia um número indeterminado de linhas contendo cada uma a idade de um indivíduo. A última linha, que não entrará nos cálculos, contém o valor da idade igual a zero. Calcule e escreva a idade média deste grupo de indivíduos.

d) Faça um programa que imprima, na tela, todos os números pares de um intervalo informado pelo usuário.

Gabarito das questões de autoaprendizagem

a)

```c
/* Desenvolver um algoritmo que leia um número
inteiro e verifique se o número é divisível
por 5 e por 3 ao mesmo tempo. */

#include<stdio.h>
#include<math.h>

int main()
{
 int numero;
 printf("Digite o número:") ;
 scanf("%d",&numero);

 if (((numero % 5)==0) && ((numero % 3)==0)){
   printf("O NUMERO E DIVISIVEL\n");
 }
 else{
   printf("O NUMERO NAO E DIVISIVEL\n");
 }
 return 0;
}
```

Gabarito das questões de autoaprendizagem

b)

```c
#include<stdio.h>
#include<math.h>

int main()
{
 int numerosala, capacidade, alunosmatriculados;
 printf("\nEntre o Nr. da SALA:");
 scanf(" %d",&numerosala);
 printf("\nEntre a capacidade da SALA:");
 scanf(" %d",&capacidade);
 printf("\nEntre o Nr. de alunos matriculados:");
 scanf(" %d",&alunosmatriculados);

 if (alunosmatriculados >= capacidade)
 {
  printf("SALA = %d\n",numerosala);
  printf("CAPACIDADE = %d\n",capacidade);
  printf("CADEIRAS OCUPADAS = %d\n",alunosmatriculados);
  printf("SALA LOTADA\n");
 }
 else if (alunosmatriculados < capacidade)
   {
    printf("SALA = %d\n",numerosala);
    printf("CAPACIDADE = %d\n",capacidade);
    printf("CADEIRAS OCUPADAS = %d\n",alunosmatriculados);
    printf("SALA NAO LOTADA\n");
   }
 return 0;
}
```

Gabarito das questões de autoaprendizagem

c)

```
1
2   #include<math.h>
3   #include<stdio.h>
4   #include<string.h>
5
6   int main(){
7    int idade, soma, count;
8    float media;
9    soma = 0;
10   count = 0;
11   printf("\nDigite a idade:");
12   scanf("%d",&idade);
13
14   while (idade > 0){
15       soma += idade;
16       count++;
17       printf("\nDigite a idade:");
18       scanf("%d",&idade);
19  }
20  media = (float)soma / (float)count;
21  printf("IDADE MEDIA = %.2f\n",media);
22  return 0;
23  }
```

Gabarito das questões de autoaprendizagem

d)

```
1
2   #include<stdio.h>
3   int main(){
4     int cont,valor1,valor2;
5     printf("Digite um valor: ");
6     scanf("%d",&valor1);
7     printf("Digite outro valor: ");
8     scanf("%d",&valor2);
9     if(valor2<valor1){
10      cont=valor1;
11      valor1=valor2;
12      valor2=cont;
13    }
14    for(cont=valor1;cont<=valor2;cont++){
15      if(cont%2==0){
16        printf("%d ",cont);
17      }
18    }
19    return(0);
20  }
```

4.2 Preparando o ambiente de programação C++

a) G++ (Terminal GNU/linux)

Para compilar códigos em C++ podemos utilizar o compilador nativo do GNU/Linux. Basta criar um arquivo de texto que contenha o código fonte e gravar com um editor de textos qualquer. O processo é simples. Tudo que tem a fazer é acessar um terminal (shell), digitar o comando de compilação e pronto.

1) Para instalar o G++ no Debian GNU/Linux utilizamos os comandos:

   - apt-get install build-essentials
   - apt-get install gcc g++ nano

2) Depois crie um arquivo utilizando um editor do GNU/Linux. Neste caso, usamos o nano. Digite o código fonte mostrado na Figura 19.

```
$ nano testa_ambiente.cpp
```

Figura 19 - Editando um código fonte no terminal com nano

Listagem 1. testa-ambiente.cpp

```
#include <iostream>

using namespace std;

int main(void){
  char s[40];
  cout << "Digite seu nome: ";
  cin >> s;
  cout << "Seu nome eh" << s << "\n";
  return 0;
}
```

Fonte: Elaborada pelo autor.

3) Para salvar aperte CTRL+O depois <ENTER>. Para sair CTRL+X.
4) Para compilar e executar basta digitar os comandos demonstrados na Figura 20.

Figura 20 - Compilando e executando

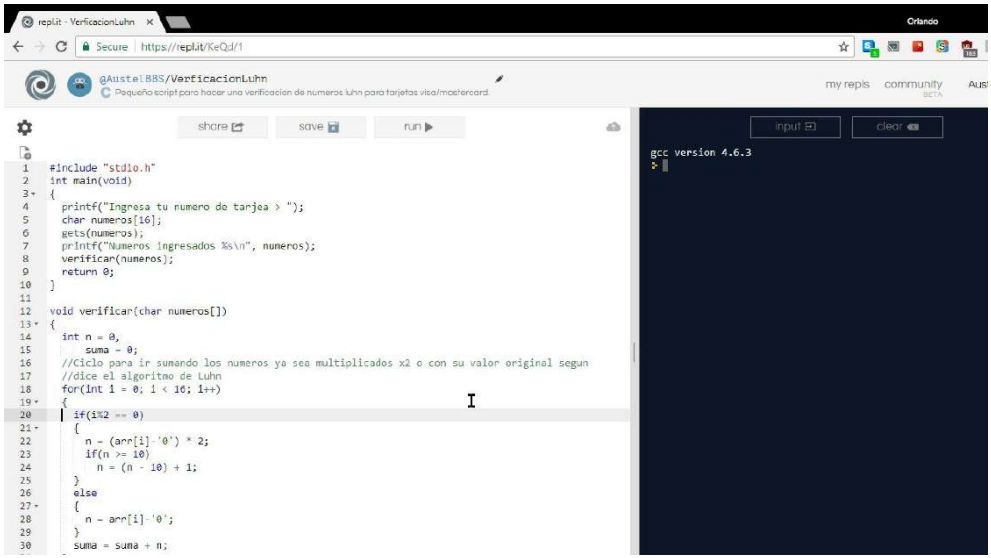
```
$ g++ testa-ambiente.cpp -o testa-ambiente
$ ./testa-ambiente

Digite seu nome: Jose
Seu nome é Jose
```

Fonte: Elaborada pelo autor.

b) Executando C++ no navegador (ONLINE) - repl.it

Outra forma bastante prática de executar códigos em C++ é utilizar um compilador online. Para utilizar com C++ basta acessar o endereço: https://repl.it/languages/cpp

Figura 21 - Compilando e executando



Fonte: imagem retirada da internet. Disponível em: <https://i.ytimg.com/vi/YT1Uky0QhP4/maxresdefault.jpg>. Acesso em: 02 de fev.2018.

4.2.1 Programação orientada a objetos

A Programação Orientada a Objetos (POO) é um paradigma de programação. Atualmente, a POO é largamente empregada na área de desenvolvimento de *software*. De acordo com AGUILAR (2008), a POO é o paradigma de programação mais utilizado no mundo de desenvolvimento de software e da engenharia de software do século XXI.

Na orientação a objetos, avançamos para uma nova abordagem que contém classes, objetos, atributos e métodos, dentre outros conceitos que nos ajudam a descrever o mundo real. É um processo de otimização da produção e manutenção de código no desenvolvimento de sistemas. Os primeiros estudos sobre POO (Programação Orientada a Objetos) ocorreram no início na década de 60 no Instituto de Tecnologia de Massachusetts (MIT), Estados Unidos.

A programação orientada a objetos nos permite (fundamenta-se em) associar objetos do mundo real para o ambiente de programação. Em nosso dia a dia, conseguimos distinguir um enumerado de coisas no mundo real, como, por exemplo, animais, casas, carros, pessoas, árvores etc. Cada objeto do mundo real pode ser descrito por meio de suas características e informações.

As linguagens de programação orientada a objetos possuem um conjunto claro de propriedades que as definem:

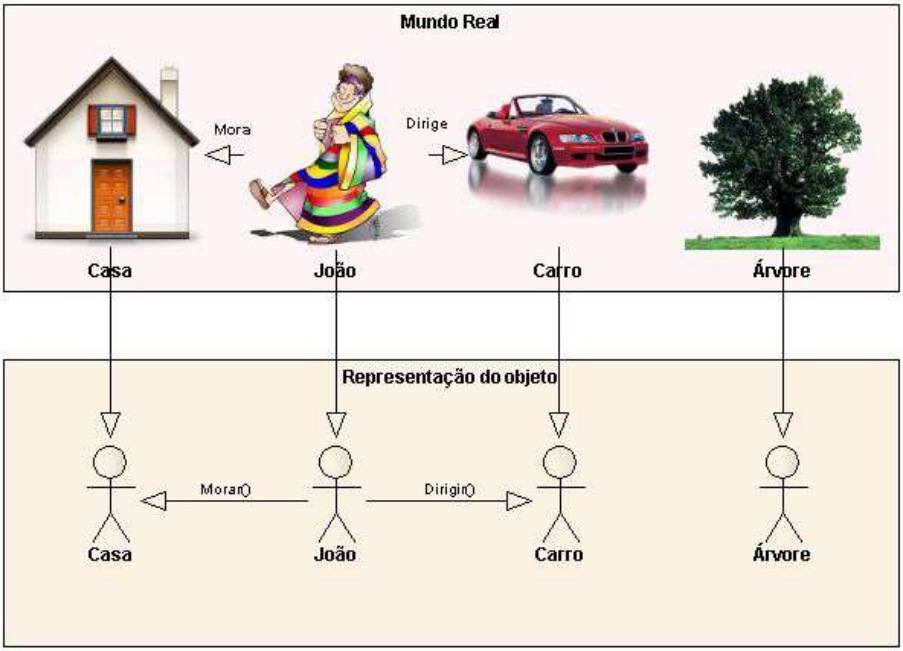
- Abstração (tipos abstratos de dados e classes);
- Encapsulamento de dados;
- Ocultação de dados;
- Herança; e
- Polimorfismo.

Neste tópico, será possível explorar todas essas características na Linguagem de Programação C++, pois ela suporta a orientação a objetos e, mesmo assim, continua compatível com C e a programação estruturada. C++ é uma extensão da linguagem C.

Abstração

Esse processo consiste em levar em conta apenas os aspectos importantes de um determinado ponto de vista, desconsiderando os aspectos restantes. No campo da programação, consiste em diferenciar entre as propriedades externas de uma entidade e os detalhes de sua composição interna. O processo de abstração possui diversos graus, denominados níveis de abstração. De maneira geral, é um meio de reduzir a complexidade.

Figura 22 - Abstração



Fonte: Imagem retirada da internet. Disponível em: <http://techblog.desenvolvedores.net/tag/construtores-e-destrutores/>. Acesso em: 01 de fev.2018

4.2.2 Classes e objetos

Classes

O ser humano se relaciona com o mundo através do conceito de objetos. Estamos sempre identificando qualquer objeto ao nosso redor. Para isso lhe damos nomes e, de acordo com suas características, lhes classificamos em grupos, ou seja, classes.

Figura 23 - Exemplo de classe

Fonte: Elaborada pelo autor.

A unidade fundamental de programação em orientação a objetos (POO) é a “classe”, e ela contém:

- Atributos (propriedades): determinam o estado do objeto;
- Métodos (comportamento): semelhantes a procedimentos (funções) em linguagens convencionais, são utilizados para manipular os atributos. É uma sub-rotina que é executada por um objeto ao receber uma mensagem (chamada a esse método).
- Cada método (função) tem uma assinatura (seu identificador - nome, tipo para o valor de retorno e sua lista de argumentos com tipo e nome também).
- A sobrecarga de métodos é quando dois métodos da classe tem o mesmo nome, porém com assinaturas (argumentos) diferentes.

As classes proveem a estrutura para a construção de objetos.

Objetos

Objetos do mundo real possuem duas características: estado e comportamento. É importante registrar que Objetos são instâncias das Classes!

Exemplos de objetos:

Cachorros → estado: nome, cor, raça
comportamento: latir, correr

Bicicletas → estado: marcha atual, velocidade atual
comportamento: trocar marcha, aplicar freios

Identificar o estado e o comportamento de objetos do mundo real é o primeiro passo para começar a pensar em POO (programação orientada a objetos).

Observe um objeto e pergunte:

Quais os possíveis estados que esse objeto pode estar?
Quais os possíveis comportamentos que ele pode executar?

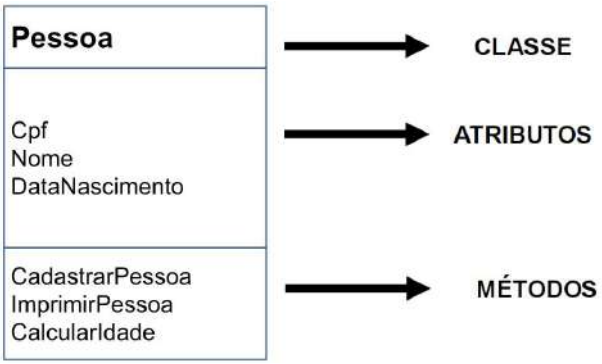
A Figura 24 apresenta dois exemplos de classes. No primeiro, a classe "Aluno" e duas instâncias "Ronald" e "Helena" com seus atributos (dados). Podemos dizer que são dois objetos (Ronald e Helena. Observe que a classe possui atributos e métodos.

Figura 24 - Exemplos de classes, objetos e instâncias

Fonte: Elaborada pelo autor.

Vamos observar um exemplo da classe Pessoa com seus atributos e métodos na Figura 25. A classe apresenta três atributos e três métodos.

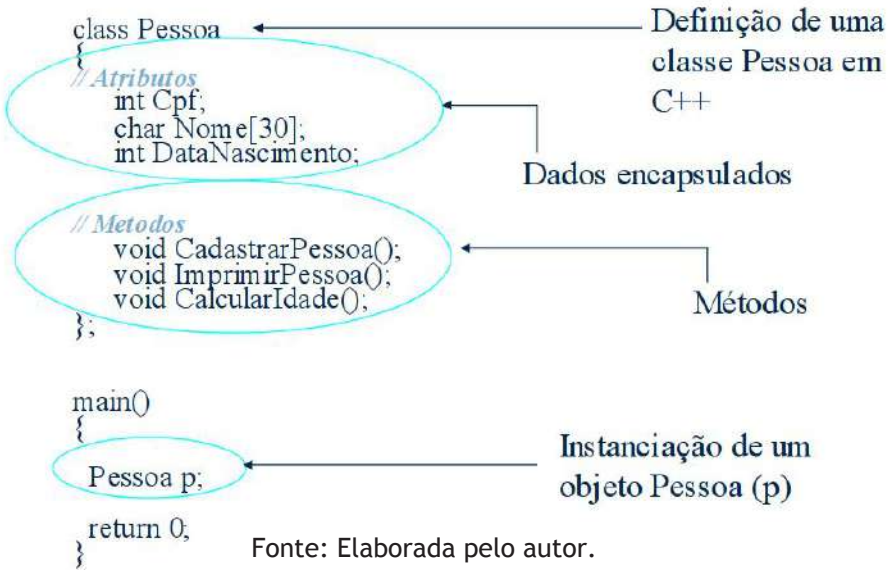
Figura 25 - Classe Pessoa



Fonte: Elaborada pelo autor.

A implementação de uma classe em C++ deve ser feita antes da declaração da função principal main(). Neste caso, ela terá o escopo global e poderá ser utilizada em qualquer parte do código. A declaração começa com a palavra reservada class. Observe, na Figura 26, o exemplo de declaração da classe Pessoa em C++.

Figura 26 - Implementação em C++ da classe Pessoa



Fonte: Elaborada pelo autor.

Bora praticar? Vamos colocar a mão no código!

Escolha o seu compilador C++ e digite o código abaixo para testar o funcionamento. Experimente fazer modificações no programa.

Use o Repl.it para testar online: https://repl.it/languages/cpp

```cpp
#include <iostream>
class Pessoa     //Definição de uma classe Pessoa
{
  //Atributos   (Dados encapsulados)
  int Cpf;
  char Nome[30];
  int DataNascimento;

  //Metodos
  void CadastrarPessoa();
  void ImprimirPessoa();
  void CalcularIdade();
};
int main() {
  Pessoa p;    //Instancia de um objeto Pessoa (p)
  return (0);
}

```

Os objetos são a abstração de uma entidade do mundo real e que possuem características (valores). Eles são instâncias de uma determinada classe. Sob o ponto de vista da programação orientada a objetos, um objeto não é muito diferente de uma variável normal.

Um programa baseado no paradigma de orientado a objetos é composto por um conjunto de objetos que interagem entre si (chamadas aos métodos - mensagens). Os objetos de software (de programação) são conceitualmente similares a objetos do mundo real: eles consistem do estado (atributos) e do comportamento (métodos) relacionados ao objeto.

Um objeto armazena seu estado em campos (variáveis) e expõe seu comportamento através de métodos (funções).

Não se esqueça! No paradigma de POO, o programa é formado por um conjunto de classes que são modelos para criação de objetos.

As classes e objetos têm membros que são dados privados e métodos de acesso que são as funções.

Quando um trecho de código quer fazer uma operação sobre um objeto, ele emite uma mensagem requisitando a operação utilizando a chamada de função (método).

Encapsulamento

O encapsulamento é princípio de projeto pelo qual cada componente de um programa deve agregar toda a informação relevante para sua manipulação como uma unidade (uma cápsula).

- Os atributos de uma classe são acessíveis apenas pelos métodos da própria classe.
- Outras classes só podem acessar os atributos de uma classe invocando os métodos públicos.
- Restringe a visibilidade do objeto, mas facilita o reuso.

Ocultação da Informação

A ocultação da Informação é o princípio pelo qual cada componente deve manter oculta sob sua guarda uma decisão de projeto única. Para a utilização desse componente, apenas o mínimo necessário para sua operação deve ser revelado (tornado público).

Não se esqueça! Encapsulamento é justamente o fato dos dados não estarem acessíveis diretamente, mas apenas através dos métodos permitidos na classe.

Visibilidade das Classes

\+ Public

Quem tem acesso à classe tem acesso também a qualquer membro com visibilidade public. É raro ter atributos públicos, mas é comum ter métodos públicos.

\- Private

O membro private não é acessível fora da classe. A intenção é que apenas quem escreve a classe possa usar esses membros.

\# Protected

O membro protected é acessível à classe e as suas subclasses. A intenção é dar acesso aos programadores que estenderão a sua classe.

Construtor

O construtor é uma estrutura especial da classe (um método). Seu objetivo é definir a configuração inicial de uma classe. Ele é utilizado quando existem atributos da classe que são essenciais para o funcionamento do objeto, porém são atributos de uma instância e, assim, variam de acordo com cada objeto.

Destrutor

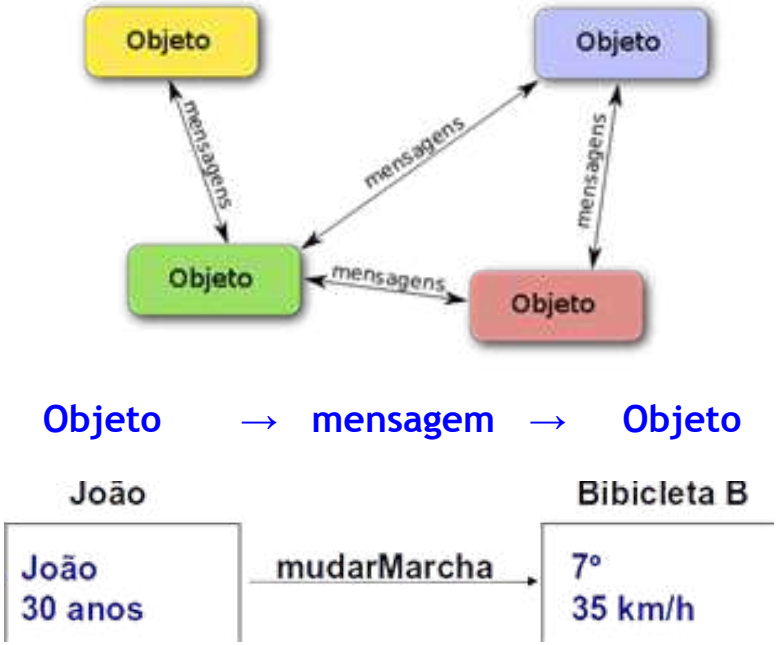
O destrutor é uma estrutura especial da classe (um método). Ele é responsável por limpar a memória ou atributos utilizados na execução da classe.

4.2.3 Invocação de métodos

A chamada de métodos é a chamada das funções do objeto que permitem modificar o seu comportamento. Os métodos normalmente são públicos para permitir a comunicação com os objetos. Os membros (atributos - dados) privados de uma classe só podem ser acessados pelos métodos (funções) da própria classe.

Os métodos (funções-membro do objeto) operam no estado interno de um objeto e servem como mecanismo de comunicação entre objetos.

Figura 27 - Comunicação entre os objetos



Fonte: Elaborada pelo autor.

Vamos praticar?

Escolha o seu compilador C++ e digite o código abaixo para testar o funcionamento. Experimente fazer modificações no programa.

Use o Repl.it para testar online: https://repl.it/languages/cpp

```cpp
#include <iostream>
using namespace std;
class Pessoa     //Definição de uma classe Pessoa
{
  private:
    char Nome[30];
    char Cpf[11];
    int AnoNasc;
  public:
    void CadastrarPessoa()
    {
      cout<<"\nNome:";
      cin.getline(Nome,30);
      cout<<"\nCPF:";
      cin.getline(Cpf,12);
      cout<<"\nAno Nascimento:";
      cin>>AnoNasc;
    };
    void ImprimirPessoa()
    {
      cout<<"\n------------------\n";
      cout<<Cpf<<"\n";
      cout<<Nome<<"\n";
      cout<<AnoNasc;
      cout<<"\n------------------\n";
    };

};
```

```cpp
int main() {
  Pessoa p;    //Instancia de um objeto Pessoa (p)
  p.CadastrarPessoa();
  p.ImprimirPessoa();
  return (0);
}
```

```
gcc version 4.6.3

Nome: Linux Tux

CPF: 00011122233

Ano Nascimento: 1973

------------------
00011122233
Linux Tux
1973
------------------
```

A linha 2 indica o escopo para entrada de argumentos em C++.

```
using namespace std;
```

A linha 5 indica que os atributos (dados) da classe Pessoa são privados. Ela está protegendo os dados de acesso direto.

```
private:
```

A linha 9 indica que os métodos da classe Pessoa são públicos.

```
public:
```

A classe Pessoa possui o método (função) chamado CadastrarPessoa() que não recebe nenhum parâmetro e não retorna nenhum valor.

```
void CadastrarPessoa()
   {
    cout<<"\nNome:";                 // Imprime a variável na tela (semelhante ao printf)
    cin.getline(Nome,30);            // armazena o valor em Nome (semelhante ao scanf)
    cout<<"\nCPF:";
    cin.getline(Cpf,12);             // armazena o valor em Cpf (semelhante ao scanf)
    cout<<"\nAno Nascimento:";
    cin>>AnoNasc;                    // armazena o valor em AnoNasc (semelhante ao scanf)
   };
```

A classe Pessoa possui o método (função) chamado ImprimirPessoa() que não recebe nenhum parâmetro e não retorna nenhum valor.

```
void ImprimirPessoa()
   {
    cout<<"\n-----------------\n";      // Imprime na tela (semelhante ao printf)
    cout<<Cpf<<"\n";                   // Imprime a variável na tela
    cout<<Nome<<"\n";
    cout<<AnoNasc;
    cout<<"\n-----------------\n";
   };
```

Dentro da função principal main(), fazemos uso da classe Pessoa para instanciar um objeto p e depois utilizamos os métodos (funções) do objeto para carregar e exibir os seus dados (atributos).

```
int main()                    // função principal
{
  Pessoa p;                   // Instancia do objeto (p) com base na Classe Pessoa
  p.CadastrarPessoa();        // chama o método CadastrarPessoa do objeto p
  p.ImprimirPessoa();         // chama o método ImprimirPessoa do objeto p
  return (0);
}
```

Vamos praticar?

Escolha o seu compilador C++ e digite o código abaixo para testar o funcionamento. Experimente fazer modificações no programa.

Use o Repl.it para testar online: https://repl.it/languages/cpp

```
1  /*
2   Controle de Acesso:
3   - O membro privado (dado) é exclusivamente acessível a métodos da mesma classe
4   */
5
6  #include <iostream>
7  #include <string>
8  using namespace std;
9
10 class Aluno
11   {
12   private:
13       string nome;
14   public:
15       void set_name(string s) {nome = s; };
16       void print() {cout << nome; };
17   } ;
18
19 int main(int argc, char**argv)
20   {
21   cout << "Dados privados e acesso atraves de funcoes\n";
22   Aluno a;
23   // a.nome = "Ronald";  // ERRO! "nome" é um membro privado!
24   a.set_name("Ronald");
25   a.print();
26   return(0);
27   }
28
```

Observe que, no código acima, a linha 23 está comentada (de propósito).

```
a.nome = "Ronald";
```

Não podemos acessar o atributo (dado) da classe diretamente, pois ele é privado (private).

Observe o erro ao lado se tentarmos acessar o dado privado.

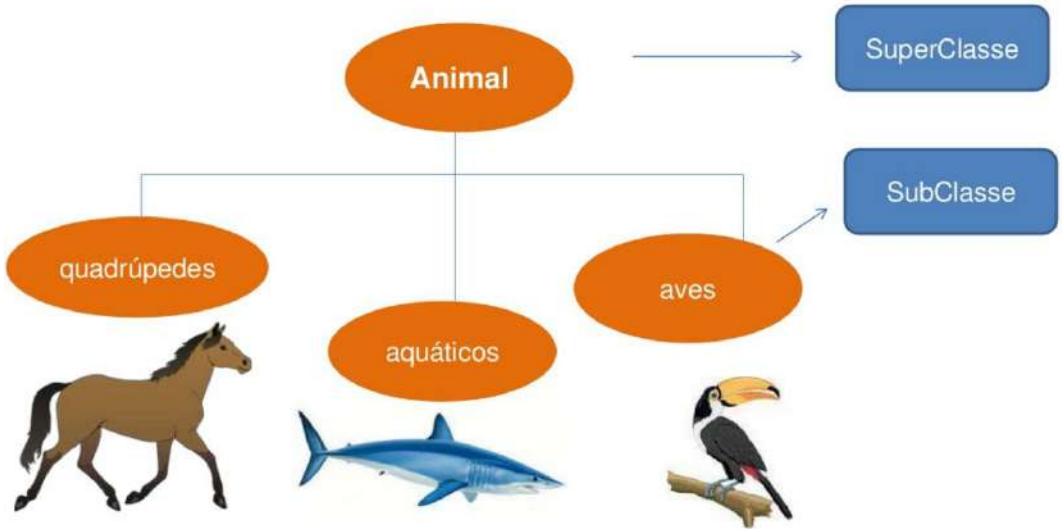
```
gcc version 4.6.3
 >
main.cpp: In function 'int main(int, char**)':
main.cpp:23:5: error: 'std::__cxx11::string Aluno::nome' is
private within this context
   a.nome = "Ronald";  // ERRO! "nome" é um membro privado!
     ^~~~
main.cpp:13:14: note: declared private here
       string nome;
              ^~~~

exit status 1
 >
```

4.2.4 Herança, polimorfismo e ligação dinâmica

Herança

A herança é o fatoramento de características comuns de diversas classes em uma classe base (superclasse). Ela é o mecanismo pelo qual uma subclasse herda todas as propriedades da superclasse e acrescenta suas próprias e exclusivas características. No processo de herança, as propriedades da superclasse não precisam ser repetidas em cada subclasse. A principal aplicação da herança é a reutilização de código.

Figura 28 - Exemplo de herança



Uma superclasse pode ser herdada por diversas subclasses. Nesta imagem, a classe Animal é herdada pelas classes: quadrúpedes, aquáticos e aves. Desta forma, a superclasse Animal possui as mesmas características que as outras subclasses.

Fonte: Elaborada pelo autor.

O mecanismo de herança faz a definição de classes mais especializadas a partir de classes básicas.

Superclasse: fornece membros a outras classes | **Subclasse**: herda membros da subclasse

Sobreposição

Refere-se à redefinição de métodos na hierarquia da herança, de forma que estes métodos implementam definições diferentes (mais especializadas) nos subtipos das classes herdeiras.

Observe as três classes abaixo na Figura 29. Elas possuem características comuns e podem ser fatoradas (agrupadas em uma classe comum). Essas semelhanças (itens comuns) estão SUBLINHADAS para destacar.

Figura 29 - Fatoração de características comuns

PROFESSOR	
Dados	**Operações**
Nome	AtualizarEndereço
Endereço	IncluirDadosPessoais
Grau	AlterarGrau
Disciplinas	AlterarDisciplinas
Área	IncluirArea
Nasc	LerDadosPessoais
NumDisc	LerDadosAcademicos

ALUNO	
Dados	**Operações**
Nome	AtualizarEndereço
Endereço	IncluirDadosPessoais
Matrícula	Matricular
Curso	IncluirNotas
Disciplinas	LerDadosPessoais
Notas	LerCurso
Nasc	LerNotasDisc
NumDisc	LerMatricula

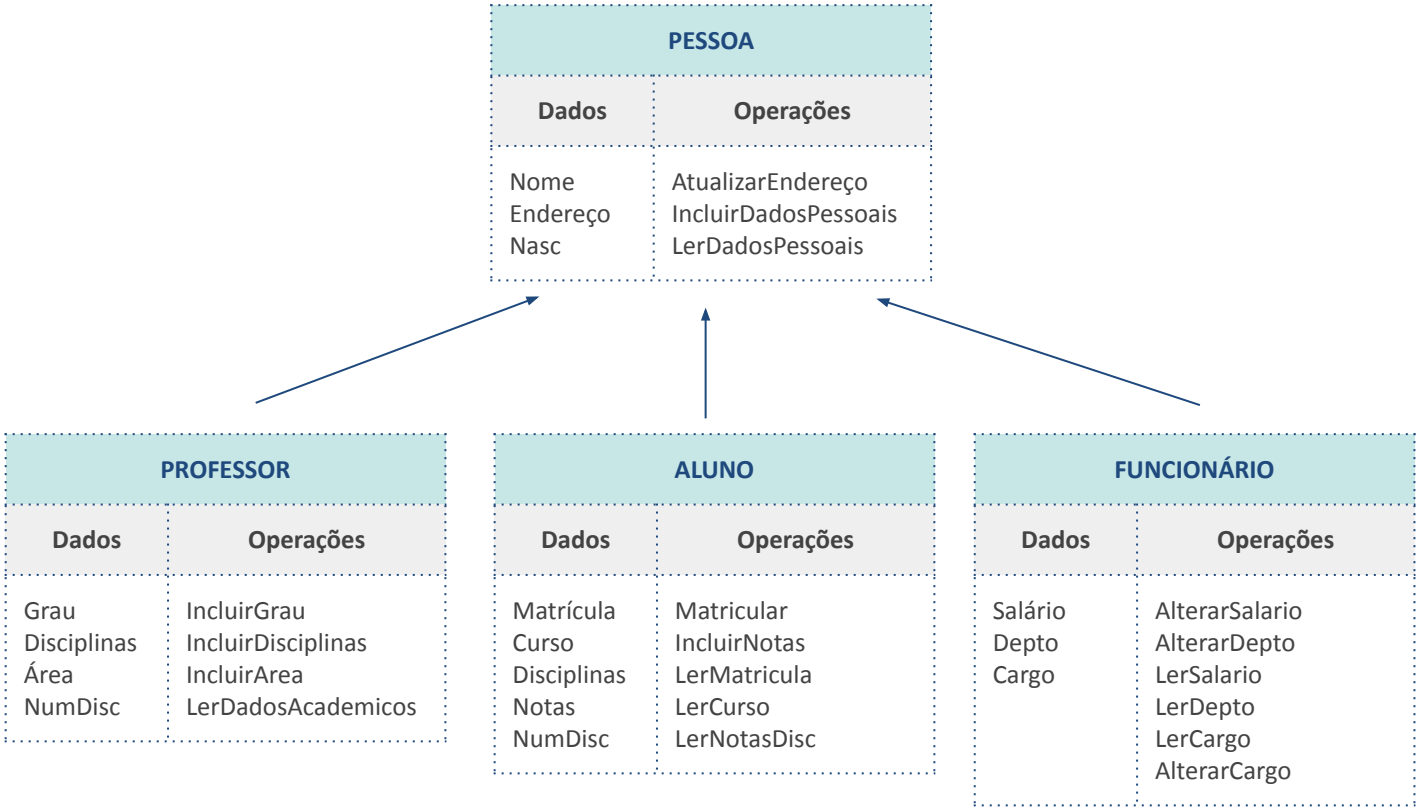
FUNCIONÁRIO	
Dados	**Operações**
Nome	AtualizarEndereço
Endereço	IncluirDadosPessoais
Salário	AlterarSalario
Depto	IncluirDepto
Nasc	LerDadosPessoais
Cargo	LerSalario LerDepto LerCargo ArlterarCargo

Fonte: Elaborada pelo autor.

A partir dessas três classes, vamos gerar uma SUPERCLASSE utilizando o conceito de HERANÇA.

Observe agora, na Figura 30, a mesma estrutura, utilizando herança, na qual pessoa é uma superclasse e professor, aluno e funcionário são três subclasses de pessoa.

Figura 30 - Superclasse Pessoa



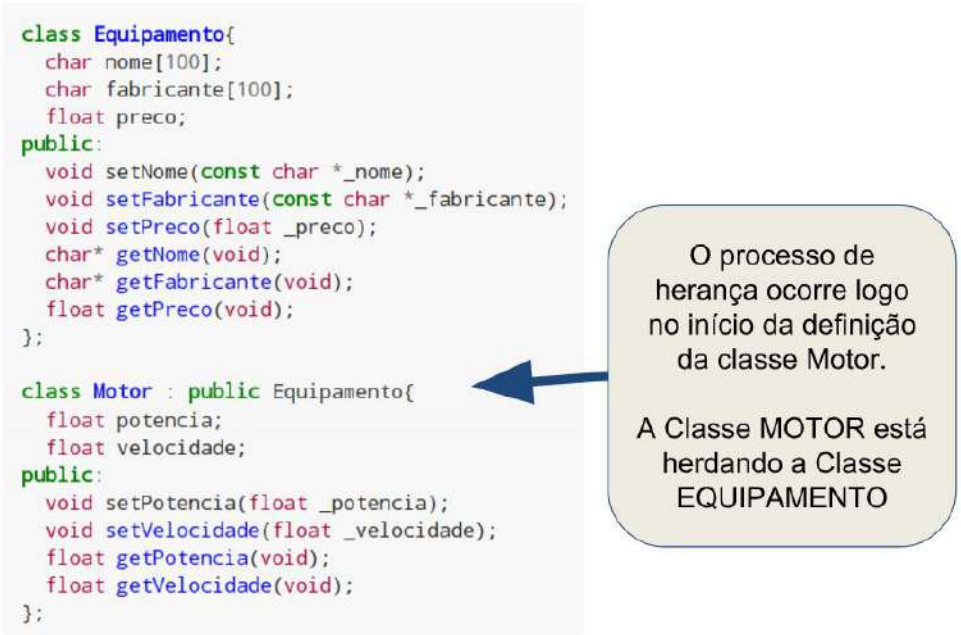
Fonte: Elaborada pelo autor.

Vamos observar, agora, um exemplo prático de Herança em C++ na Figura 31. O processo de herança está materializado na linha:

```
class Motor : public Equipamento {
```

A Classe Motor está herdando todas as características da Classe Equipamento.

Figura 31 - Exemplo de herança em C++



Fonte: Elaborada pelo autor.

Polimorfismo

O polimorfismo permite que referências de tipos de classes mais abstratas representem o comportamento das classes concretas que referenciam. Assim, um mesmo método pode apresentar várias formas, de acordo com seu contexto.

O Polimorfismo torna possível reimplementar métodos de uma classe básica na classe mais especializada e, posteriormente, tratar diversos objetos diferentemente especializados através dos mesmos métodos.

A especificação de classes, cuja única função é definir conjuntos de métodos, dá origem ao conceito de classes abstratas e as diferencia dos tipos concretos que são aqueles que realmente implementam os métodos.

Ligação dinâmica

O polimorfismo ocorre quando uma mesma mensagem, chegando a objetos diferentes, provoca respostas diferentes. Ligação dinâmica é a chave para fazer polimorfismo.O Compilador não gera o código para chamar o método em tempo de compilação, em vez disso, cada vez que se aplica um método a um objeto, o compilador gera código para calcular que método deve ser chamado, usando informações de tipo do objeto. Na ligação dinâmica, as bibliotecas são carregadas e descarregadas da memória conforme solicitadas durante a execução do programa.

Vamos praticar?

Escolha o seu compilador C++ e digite o código abaixo para testar o funcionamento. Experimente fazer modificações no programa.

Use o Repl.it para testar online: https://repl.it/languages/cpp

Exemplo 1 - Implemente e teste o código. Faça alterações nos atributos e métodos.

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   class Veiculo {
6     protected:
7      int rodas;
8      float peso;
9     public:
10     void criar (int in_rodas, float in_peso){
11       rodas = in_rodas;
12       peso = in_peso;
13     };
14     int get_rodas (void){
15       return (rodas);
16       };
17     float get_peso (void){
18       return (peso);
19       };
20     float carga_por_roda (void){
21       return (peso/rodas);
22     };
23  };
24
25  class Carro:public Veiculo {
26    int carga_de_passageiros;
27    public:
28     void criar(int in_rodas, float in_peso, int pessoas = 4){
29       rodas = in_rodas;
30       peso = in_peso;
31       carga_de_passageiros = pessoas;
32     };
33     int passageiros(void){
34       return (carga_de_passageiros);
35     };
36  };
37
```

(continua...)

```
38  class Caminhao:public Veiculo {
39   int carga_de_passageiros;
40   float carga;
41   public:
42    void criar_caminhao(int quantos = 2, float carga_maxima = 24000.0){
43      carga_de_passageiros = quantos;
44      carga = carga_maxima;
45    };
46    float eficiencia(void){
47      return (carga_de_passageiros-(peso/carga));
48    };
49    int passageiros(void){
50       return (carga_de_passageiros);
51    };
52  };
53
54  int main()
55  {
56  Veiculo monociclo;
57
58  monociclo.criar(1, 12.5);
59  cout << "O monociclo tem " << monociclo.get_rodas() << " rodas.\n"<<endl;
60  cout << "O monociclo carrega " << monociclo.carga_por_roda() << " Kg em uma única roda.\n";
61  cout << "O monociclo pesa " << monociclo.get_peso() << " Kg.\n\n";
62
63  Carro sedan;
64
65  sedan.criar(4, 3500.0, 5);
66  cout << "O sedan carrega " << sedan.passageiros() << " passageiros.\n";
67  cout << "O sedan pesa " << sedan.get_peso() << " Kg.\n";
68  cout << "O sedan está carregando " << sedan.carga_por_roda() << " Kg por roda.\n\n";
69
70  Caminhao bau;
71
72  bau.criar(18, 12500.0);
73  bau.criar_caminhao(1, 33675.0);
74  cout << "O bau pesa " << bau.get_peso() << " Kg.\n";
75  cout << "A eficiência do bau é de " << 100.0 * bau.eficiencia() << " %.\n";
76  return (0);
77  }
```

```
gcc version 4.6.3
:
O monociclo tem 1 rodas.

O monociclo carrega 12.5 Kg em uma única roda.
O monociclo pesa 12.5 Kg.

O sedan carrega 5 passageiros.
O sedan pesa 3500 Kg.
O sedan está carregando 875 Kg por roda.

O bau pesa 12500 Kg.
A eficiência do bau é de 62.8805 %.
:
```

Vamos praticar?

Exemplo 2 - Implemente e teste o código. Faça alterações nos atributos e métodos.

```
1   #include <iostream>
2   #include <cstring>
3
4   using namespace std;
5
6   class Equipamento{
7     char nome[100];
8     char fabricante[100];
9     float preco;
10  public:
11    void setNome(const char *_nome);
12    void setFabricante(const char *_fabricante);
13    void setPreco(float _preco);
14    char* getNome(void);
15    char* getFabricante(void);
16    float getPreco(void);
17  };
18
19  class Motor : public Equipamento{
20    float potencia;
21    float velocidade;
22  public:
23    void setPotencia(float _potencia);
24    void setVelocidade(float _velocidade);
25    float getPotencia(void);
26    float getVelocidade(void);
27  };
28
29  void Equipamento::setNome(const char *_nome){
30    strcpy(nome,_nome);
31  }
32
33  void Equipamento::setFabricante(const char *_fabricante){
34    strcpy(fabricante,_fabricante);
35  }
36
37  void Equipamento::setPreco(float _preco){
38    preco=_preco;
39  }
40
```


(continua...)

Vamos praticar?

```
41  char* Equipamento::getNome(void){
42    return nome;
43  }
44
45  char* Equipamento::getFabricante(void){
46    return fabricante;
47  }
48
49  float Equipamento::getPreco(void){
50    return preco;
51  }
52
53  void Motor::setPotencia(float _potencia){
54    potencia=_potencia;
55  }
56
57  void Motor::setVelocidade(float _velocidade){
58    velocidade=_velocidade;
59  }
60
61  float Motor::getPotencia(void){
62    return potencia;
63  }
64
65  float Motor::getVelocidade(void){
66    return velocidade;
67  }
68
69  int main(void){
70    Motor m;
71    m.setFabricante("ACME");
72    m.setPreco(23.45);
73    m.setNome("Speedatron");
74    m.setPotencia(130);
75    m.setVelocidade(280);
76    cout << m.getFabricante() << "\n"
77         << m.getPreco() << "\n"
78         << m.getNome() << "\n"
79         << m.getPotencia() << "\n"
80         << m.getVelocidade() << "\n";
81  }
```

```
gcc version 4.6.3

ACME
23.45
Speedatron
130
280
```

Figura 32 - Humor

Fonte: Imagem retirada da internet. Disponível em: <https://vidadeprogramador.com.br/wp-content/uploads/2011/11/tirinha344.png>. Acesso em: 2 de fev.2018.

Saiba mais!

Você ainda está em dúvida em como instalar um ambiente para desenvolver com POO e C++?

Assim como qualquer linguagem de programação, precisamos editar os códigos em C++ de alguma maneira. Qualquer editor simples pode ser empregado para editar o código-fonte desenvolvido em C++. Ocorre que desenvolver utilizando uma IDE (Ambiente Integrado de Desenvolvimento) é muito mais prático.

Acompanhe no site do professor Agostinho Brito, um resumo completo da Programação Orientada a Objetos e C++ e aproveite para aprofundar a leitura sobre como preparar o ambiente de desenvolvimento em C++.

Todo o esforço vale a pena. Conhecimento nunca é demais.

Acesse o site:

https://agostinhobritojr.github.io/tutorial/cpp/

Chegamos ao fim da nossa disciplina! Parabéns. Foi muito legal compartilhar conhecimentos com você. Integramos em diversas atividades os principais conceitos da programação e da lógica. Neste tópico trabalhamos os conceitos básicos da Programação Orientada à Objetos como classes, objetos, atributos e métodos. Percebemos que os objetos são instâncias de uma classe, assim como seus atributos representam os dados os métodos (funções) representam o seu comportamento. Observamos como ocorre a comunicação entre os objetos. Esse processo de comunicação é feito por chamadas aos métodos públicos e as ações realizadas são as mensagens. Aprendemos como criar Classes e Objetos com a Linguagem C++. Utilizamos construtores e destrutores de classes e instanciamos objetos. Também compreendemos como funciona o encapsulamento, a herança e o polimorfismo. Finalizando este tópico realizamos implementação de programas que envolvem classes em objetos em C++. Aproveite para consultar o glossário se tiver alguma dúvida de algum termo. Vamos partir para algumas questões de aprendizagem e para a revisão final do tópico. Não esqueça de olhar os sites e vídeos indicados que irão ampliar os seus horizontes de conhecimentos sobre o tema da unidade. Gostaria de agradecer o seu esforço e dedicação para chegar até o final do curso. Tenha certeza que os resultados virão. Acredite em seu potencial e dedique-se que o resultado virá. Boa sorte e até uma próxima oportunidade.

Vamos rever?

POO é o paradigma de programação mais utilizado no mundo de desenvolvimento de software e da engenharia de software do século XXI. Essa nova abordagem que contém classes, objetos, atributos e métodos, dentre outros conceitos, nos ajudam a descrever o mundo real.

A programação orientada a objetos nos permite (fundamenta-se em) associar objetos do mundo real para o ambiente de programação. Cada objeto do mundo real pode ser descrito por meio de características e informações próprias. As linguagens de programação orientadas a objetos possuem um conjunto claro de propriedades que as definem: abstração (tipos abstratos de dados e classes); encapsulamento de dados; ocultação de dados; herança e polimorfismo. A unidade fundamental de programação em orientação a objetos (POO) é a "classe" e ela contém: atributos (propriedades): determinam o estado do objeto; métodos (comportamento): semelhantes a procedimentos (funções) em linguagens convencionais, são utilizados para manipular os atributos. As classes proveem a estrutura para a construção de objetos. Objetos são instâncias das classes!

Sob o ponto de vista da programação orientada a objetos, um objeto não é muito diferente de uma variável normal. Um programa orientado a objetos é composto por um conjunto de objetos que interagem entre si. Um objeto armazena seu estado em campos (variáveis) e expõe seu comportamento através de métodos (funções). No paradigma de POO, o programa é formado por um conjunto de classes que são modelos para criação de objetos.

Vamos rever?

As classes e objetos têm membros que são dados privados e métodos de acesso que são as funções. Quando um trecho de código quer fazer uma operação sobre um objeto, ele emite uma mensagem requisitando a operação, utilizando a chamada de função (método).

Encapsulamento é justamente o fato de os dados não estarem acessíveis diretamente, mas apenas através dos métodos permitidos. Herança é o mecanismo pelo qual uma subclasse herda todas as propriedades da superclasse e acrescenta suas próprias e exclusivas características. O polimorfismo ocorre quando uma mesma mensagem chegando a objetos diferentes provocam respostas diferentes.

Sites indicados

1) Cartilha de Programação em C para ARDUINO - João Alexandre
https://edisciplinas.usp.br/pluginfile.php/4287607/mod_resource/content/3/Cartilha_de_programacao_Arduino.pdf

2) Linguagem C++ - Notas de Aula
http://www.inf.ufpr.br/ci208/NotasAula.pdf

3) Curso de Linguagem C - CPDEE / UFMG
http://www.pucsp.br/~so-comp/cursoc/c.html

4) Curso C++
http://excript.com/curso-cpp.html

5) Guia de Referência Rápida - C++
https://pt.wikibooks.org/wiki/Refer%C3%AAncia_r%C3%A1pida_de_C%2B%2B

Audiovisuais indicados

1) Curso C++
https://www.youtube.com/playlist?list=PLesCEcYj003QTw6OhCOFb1Fdl8Uiqyrqo

2) Curso de C++ - CFBCursos
https://www.youtube.com/playlist?list=PLx4x_zx8csUjczg1qPHavU1vw1IkBcm40

3) Curso Programação em C/C++
https://www.youtube.com/playlist?list=PLC9E87254BD7A875B

4) Partiu C++
https://www.youtube.com/playlist?list=PLG6ZmQok0kq8EitPWzqmTxJfdt4MyHtUM

Questões de autoaprendizagem

a) Digite o código abaixo e teste. Altere o programa para criar mais dois atributos na Classe Aluno (CPF e Identidade). Ajuste o código para permitir o carregamento e a exibição dos dados corretamente. Crie mais um objeto "b" a partir da Classe Aluno.

```
/*  Consistência:
 - Controle de acesso restrito permite verificações de consistência a serem realizadas
 */
#include <iostream>
#include <string>
using namespace std;
// Enum é um vetor de valores onde cada item tem um índice começando em 0..
enum Cabelo { qualquer,loiro, castanho, caracol, grisalho }; //lista de valores

class Aluno
  {
  private:
      string nome;
      int idade;
      Cabelo cabelo;
  public:
      void set_idade(int i) {idade = i; };
      void set_name(string s) {nome = s; };
      int get_idade() { return idade; };
      void set_cabelo(Cabelo c);
      void print() ;
  } ;

void Aluno::set_cabelo(Cabelo c)
  {
  if (idade <35 && c==grisalho)
    {
    cout << "erro" << endl;
    cabelo = qualquer;
    }
  else cabelo = c;
  };
```

(a questão a) continua na próxima página)

Questões de autoaprendizagem

```
33
34  void Aluno::print()
35    {
36    cout << "Nome: " << nome << endl;
37    cout << "Idade: " << idade << " anos\n";
38    cout << "Cabelo: " << cabelo << endl;
39    };
40
41  // Função Principal
42  int main()
43    {
44    Aluno a;   //Cria o objeto "a"  - Instância da Classe Aluno
45
46    a.set_name("Giovani");   //seta nome (carrega valor chamando o método)
47    a.set_idade(30);  // seta a idade chamando o método
48    a.set_cabelo(grisalho);  // tenta setar o valor de cabelo,
49                             // mas como idade<35 e cabelo=grisalho não aceita
50                             // então carrega cabelo como "qualquer (valor 0)"
51    a.print();
52    return(0);
53    }
```

Questões de autoaprendizagem

b) Digite e teste o programa que implementa a Classe Aluno. Modifique o código apresentado e crie mais um objeto “b” a partir da Classe Aluno. Adicione mais um atributo idade na Classe Aluno e faça o carregamento e impressão correta dos dados.

```
/*
 Controle de Acesso:
 - public e private são como labels que controlam o acesso aos membros
 - funcoes em uma classe são chamadas de métodos
 - métodos podem ter seu corpo definido dentro da definição da classe
   nesse caso, devem ser funções inline
 - caso contrário, podem ter seu corpo definido depois com a sintaxe:
    nome_da_classe::nome_da_função
 Exercícios:
 - inclua mais membros privados representando mais dados
 - inclua mais membros publicos representando mais operacoes
 */

#include <iostream>
using namespace std;
class Aluno
  {
  private:
      string nome;
  public:
      void set_name(string s) {nome = s; };
      void print() {cout << nome; };
  } ;

int main(int argc, char**argv)
  {
  cout << "Dados privados e acesso atraves de funcoes\n";
  Aluno a;
  // a.nome = "Giovani";  // ERRO: nome eh membro privado
  a.set_name("Giovani");
  a.print();
  }
```

Glossário

- algoritmo - um algoritmo é uma sequência finita de instruções bem definidas e não ambíguas; cada uma das quais pode ser executada mecanicamente num período de tempo finito e com uma quantidade de esforço finita.
- alto nível - são aquelas que estão mais próximas da linguagem humana, ou seja, são mais fáceis de compreensão. São mais simples, pois utilizam uma sintaxe estruturada, assim seu código é mais legível.
- arrays (ou vetor) – estruturas de dados unidimensionais ou multidimensionais.
- baixo nível - são linguagens escritas usando as instruções do microprocessador do computador. São complexas tanto para compreensão como para a criação de programas.
- biblioteca de classes – programas já desenvolvidos que contêm funcionalidades para utilização de outros programas.
- breakpoint - quando se executa um programa via ambiente de programação, você pode escolher a opção de depuração para encontrar erros e uma das maneiras de se fazer isto é colocar, no meio do programa, um ponto de parada, o chamado breakpoint.
- C – uma linguagem de programação popular; o equivalente hexadecimal ao número doze no sistema decimal.
- C# (C sharp) – linguagem de programação orientada a objetos da Microsoft, derivada de C. C++ - uma linguagem de programação orientada a objetos.

Glossário

- campo - uma série de caracteres formando uma unidade de informação; o espaço para entrada de dados no banco de dados.
- caracter – é uma unidade de informação correspondente a um símbolo escrito (uma letra, um número etc.) ou algo semelhante. Representado por %c ou char – (Ex: 1, 2, A, B, C): server para armazenar informações alfanuméricas, porém, com esse tipo não se pode realizar operações aritméticas e sim apenas trabalhar informações.
- carregar - transferir algo enquanto se prendendo a ele; comunicar, passar adiante.
- carregar; carga - abrir um programa ou documento; a quantidade de processos sendo realizada.
- cascata - tipo de desenvolvimento onde estágios são completados sequencialmente e passos anteriores não se sobrepõem com posteriores e não podem ser facilmente modificados.
- CASE (Engenharia de Software Auxiliada por Computador) - a tecnologia e as técnicas do uso de software para auxiliar ou automatizar o desenvolvimento de software.
- caso de uso - técnica para capturar os requisitos funcionais de um sistema.
- ciclo de desenvolvimento - ciclo para desenvolvimento de software composto de cinco fases individuais.
- ciclo de vida - todos os estágios pelos quais passam projetos e produtos, da concepção inicial ao término.

Glossário

- classe - definição do que uma rotina programada faz; em programação orientada a objetos, é um construto usado para agrupar objetos semelhantes.
- Classe abstrata - aquela definida para generalizada para que atenda a diversos objetivos. Classe base - nome dado a uma classe herdada por outra(s) classe(s).
- classe coleção - em programação orientada a objetos, uma classe que pode armazenar outros objetos.
- Classes concretas - são diferentes das abstratas por conterem implementações; no caso, as abstratas podem conter métodos abstratos e estes não possuem corpo. As abstratas também não podem ser instanciadas, já as concretas podem sê-las.
- Classe derivada - nome dado a uma classe que herda características de outra classe. Classe filha - o mesmo que classe derivada.
- Classe pai - o mesmo que casse base.
- código de máquina - linguagem de baixo nível que manipula o hardware diretamente, em especial a unidade central de processamento.
- código de retorno - mensagem que informa o status e o sucesso de um pedido de dados.
- código fonte - texto original de um programa ou documento que deve ser compilado antes de ser executado ou visualizado.
- código, codificar - instruções em linguagem de programação para um programa ou página na web que podem ser lidas e compiladas; escrever código.

Glossário

- comando - uma instrução para realizar uma ação.
- comentário - uma resposta a uma mensagem em um weblog ou fórum público na web, postado junto à mensagem original; uma nota explanatória no código-fonte para leitores humanos.
- compilador - um programa para compilação. São programas que fazem a tradução de todas as instruções de um programa fonte, gerando um programa executável.
- compilar - transformar dados escritos em uma linguagem de computador em um formato que pode ser executado. É o processo que o compilador executa. Ele verifica a existência de erros de sintaxe no código fonte e depois gera um programa executável que contém o binário equivalente ao código fonte que recebeu.
- comunidade virtual - uma comunidade cujos membros se conhecem e se relacionam principalmente através de computadores.
- Constante - É um tipo abstrato de dado que é armazenado, porém, esse dado não pode ser alterado, ou seja, não sobre nenhuma variação no decorrer do tempo.
- construtor - método especial em linguagens orientadas a objetos, usado para a criação de objetos. contador de programa - contador de registro que aponta para a próxima instrução a ser executada. do while (laço repita) – repita uma instrução até que uma condição definida no final do laço do seja verdade.
- Else (Então) – saída de uma condição IF.

Glossário

- Elseif (Senão SE) – estrutura lógica de decisão dentro de outra estrutura lógica de decisão.
- Erro de lógica - ocorre durante a execução de um programa (código fonte), devido a um erro de lógica na execução dos comandos (instruções).
- Erro de sintaxe - ocorre durante o processo de compilação do programa, quando o programador, ao editar o seu programa fonte, não respeita alguma regra de sintaxe da linguagem de programação.
- Função - uma sequência de passos, ações e atividades designadas a um usuário ou programa. Fluxograma - são formas de expressar a lógica por meio de símbolos.
- For (laço para) – esse laço contém uma condição inicial, uma condição final e um passo.
- Framework - modelos abstratos de um determinado tipo de aplicação. Eles trazem toda a funcionalidade básica de um tipo de aplicação que pode ser utilizada conforme a necessidade.
- Herança - recurso da programação orientada a objetos que permite a uma determinada classe herdar características de outra classe.
- IDE - Integrated Development Environment (Ambiente de desenvolvimento integrado), como o ambiente se apresenta para o usuário, seus menus, suas funções e facilidades.
- Identificadores - são os nomes que damos as variáveis.

Glossário

- Inteiro - tipos de dados para armazenar informações numéricas de números inteiros positivos ou negativos. Representado por %d ou int – (Ex: 1, -1).
- Interface - são modelos para a construção de outras classes.
- Interpretadores - são programas que interpretam cada instrução do arquivo de código fonte do programa, executando-o dentro de um ambiente de programação. Como exemplo de linguagem interpretada temos o PHP.
- IF (ou SE) – estrutura lógica de condição. Verifica-se se uma condição é Verdade ou Falsa e toma-se uma decisão. Se a condição for verdadeira, executa-se um conjunto de instruções, se ela for falsa, executa-se outro conjunto de instruções.
- Linguagem de máquina - é a linguagem básica dos computadores. Formada por 0 e 1. Não é uma linguagem simples e legível para um programador.
- Linguagem de programação - arcabouço (conjunto) de regras sintáticas e semânticas empregadas para a construção de um programa.
- Lógico – representa Verdadeiro ou Falso: do tipo lógico, aceita apenas dois valores, Verdadeiro ou Falso.
- Matrizes - estruturas de dados multidimensionais.
- Métodos - funções que determinam qual o comportamento que uma classe terá.
- Modificador - altera a situação normal de um comando ou instrução, assim como são os modificadores de acesso (public, private, protected).
- Objetos - tudo na POO é objeto; os objetos; são exemplares de uma classe qualquer.

Glossário

- Operadores - símbolos que representam operações aritméticas, lógicas ou relacionais. São normalmente empregados em expressões e condições em estruturas de repetição ou decisão.
- Override - indica que o método com este escopo será sobrescrito em alguma classe derivada.
- Pseudocódigo - uma forma de imitar a linguagem de programação através de uma linguagem natural.
- Polimorfismo - significa executar tarefas de formas diferentes. Private - Pode ser acessado apenas pela própria classe.
- Programa - é um conjunto de instruções (comandos) que descrevem uma determinada tarefa a ser executada por um computador.
- Programar - é a ação de escrita, teste e manutenção de um programa. Protected - pode ser acessado pelas subclasses da classe base.
- Public - método de acesso que permite ao membro definido ser acessado de qualquer outra classe ou método.
- Sobreposição - quando um método é reescrito em uma classe derivada, sendo que seu nome permanece o mesmo (aquele definido na classe base).
- Real - quando se quer armazenar informações numéricas que pertençam ao conjunto de números reais, porém nesse caso aceita números decimais. Representado por %f ou float – (Ex: 5.50, 1.57).

Glossário

- String - utilizado em programação e em linguagens formais, uma cadeia de caracteres (também conhecida como samblagem ou string) é uma sequência ordenada de caracteres (símbolos) escolhidos a partir de um conjunto pré-determinado. Em programação, cada símbolo armazenado na memória é representado por um valor numérico. Uma variável declarada com tipo de dado cadeia geralmente armazena um número pré-determinado de caracteres. Representado por %s ou str (Ex: 1, 2, 3, A, B, C).
- switch (significa escolha) – avalia o valor da variável X e caso X seja igual ao valor 1, executa instrução 1, se for igual ao valor 2, executa instrução 2, se for igual ao valor 3, executa instrução 3 e assim sucessivamente.
- Vetores ou arrays - estruturas de dados unidimensionais ou multidimensionais.
- Variável - é um tipo abstrato de dado que é armazenado e pode ser alterado em algum instante no decorrer de um certo tempo durante o processamento de dados no computador.
- while (laço enquanto) – enquanto uma condição for verdade, continua repetindo a instrução.

Referências

BROWN, Tim. Design Thinking. **Uma Metodologia Poderosa para Decretar o Fim das Velhas Ideias.** Alta Books, 2017.

FLANAGAN, David. **JavaScript - O Guia Definitivo.** Bookman, 2012.

FREEMAN, Eric. FREEMAN, Elisabeth. **Use a Cabeça! HTML com CSS e XHTML.** Alta Books, 2008.

MELO, Adriana. ABELHEIRA, Ricardo. **Design Thinking and Thinking Design - Metodologia, Ferramentas e Uma Reflexão Sobre o Tema.** Novatec, 2015.

MORRISON, Michael. **Use a Cabeça! Javascript.** Alta Books, 2008.

SHARP, Helen. ROGERS, Yvonne. PREECE, Jennifer. **Design de Interação - Além da Interação Homem-computador,** 3ª Ed., 2013.

SILVA, Maurício Samy. **A Fundamentos de Html5 e Css3.** Novatec, 2015. WATRALL, Ethan. **Use a Cabeça! Web Design.** Alta Books, 2009.

Fundamentos de Webdesign

Salvador Alves de Melo Júnior

Mensagem de Boas-Vindas

Seja bem-vindo (a) ao módulo Fundamentos de Web Design!!!

Você irá se divertir bastante desenvolvendo páginas web, utilizando os conceitos de usabilidade, escrevendo códigos em HTML e JavaScript, formatando tudo com o CSS.

Este capítulo não visa formar web designers profissionais, e sim, apresentar importantes conceitos, que serão úteis no desenvolvimento de sistemas e na customização de templates.

Sabe o que são templates? Neste contexto, são modelos prontos de sites, que já vêm com todo o básico que você precisa em termos de HTML, CSS e JavaScript. Assim, basta você colocar o seu conteúdo e adaptar a formatação, usando os fundamentos de WebDesign que você verá neste curso.

Você que adora acessar a internet, ver o seu youtuber preferido, ou acessar a sua página do Facebook, já parou para pensar como aquelas páginas são feitas?

Bom, tudo começa com uma ideia, que se junta às ideias de outras pessoas criativas. Essa criatividade deve ser traduzida em uma interface que seja fácil e agradável de aprender e de usar. A isso chamamos de usabilidade, um dos tópicos deste curso.

No final, tudo precisa ser codificado de forma que o seu navegador entenda. Essa codificação pode estar dividida em camadas. Nesse caso, podemos utilizar o HTML, o CSS e o JavaScript.

Você pode estar pensando... é muita coisa!!! Que nada, com o tempo você verá que tudo é muito mais diversão que trabalho ou estudo. Mas apenas para você ter uma ideia, vou apresentar os principais conceitos.

O HTML é responsável por apresentar o conteúdo da sua página, independente do estilo ou do comportamento da aplicação. Em outras palavras, é um texto no qual você define o conteúdo da sua página, que é interpretado pelo seu navegador, sem se importar se a cor da fonte será em vermelho ou azul.

O CSS define o estilo da sua página, sem se importar com o conteúdo. Em outras palavras, o CSS é um texto no qual você pode definir, por exemplo, que todas as tabelas terão uma fonte com a cor vermelha, independente de qual tabela será usada.

O JavaScript controla o comportamento de algum elemento da sua página, permitindo uma maior interação com o usuário, sem ter que obrigatoriamente passar pelo servidor. Então você pode, por exemplo, alertar o usuário que ele não pode deixar em branco um campo de formulário de cadastro.

Espero que você aproveite a oportunidade e construa páginas incríveis usando a técnica aqui aprendida e a sua criatividade!!

Unidade 1 Arquitetura de Informação

Olá! Leitor,
Para iniciar o estudo sobre Web Design é essencial que você conheça os aspectos básicos relacionados à Arquitetura de Informação. Portanto, nesta primeira unidade você estudará os conceitos de usabilidade, acessibilidade e legibilidade. Também descobrirá um pouco sobre os fundamentos e padrões de desenho para Web. Além disso, você vai conhecer sobre interfaces para sites, web e portais.

Ao final desta unidade, você será capaz de:

- Definir os limites de atuação profissional em Web Design;
- Identificar o que precisa fazer para desenvolver uma página que seja fácil de usar e de aprender;
- Descrever os conceitos de usabilidade, acessibilidade e legibilidade;
- Apontar os fundamentos e padrões de desenho para Web; e
- Caracterizar os aspectos que configuram as interfaces para sites, web e portais.

Vamos nessa? Vem comigo navegar nesse fantástico universo da Arquitetura da Informação.

1.1 Usabilidade

Para início de conversa...

Você já parou para pensar nos equipamentos que você usa no dia a dia? Quais deles são realmente fáceis de usar? Quais deles você aprendeu a usar com facilidade?

Você pode ter certeza, todos os equipamentos que são fáceis de usar foram projetados tendo o usuário em mente.

Veja dois exemplos de um controle remoto de TV. Qual deles parece ser mais fácil de usar?

Imagem 1 - Controle remoto TV
Fonte: https://support.apple.com/pt-br/HT205329

Imagem 2 - Controle remoto TV
Fonte: https://pixabay.com/pt/vectors/controle-remoto-tv-televis%C3%A3o-146212/

Esperto como você é, já deve ter percebido que um controle remoto com menos botões, mais leve e menor, que faz coisas semelhantes, é muito mais fácil de usar e muito mais fácil de aprender a usar.

Os mesmos problemas vistos nos equipamentos podem ser identificados em sites considerados ruins na internet. Vejamos, a seguir, um exemplo bom e um ruim:

- Exemplo positivo:
  O site de busca da Google tem uma finalidade principal, fazer pesquisas na internet, então sua página, simples e direta, reflete bem isso.

Imagem 3 - Tela inicial do Google
Fonte: www.google.com

- Exemplo negativo:
  O site de Compras Governamentais têm evoluído bastante nos últimos tempos. É um importante portal governamental, tanto para os gestores, quanto para os fornecedores. Entretanto, esse portal também é responsável pela divulgação e a realização das licitações. Desafio você a localizar, em menos de 3 minutos, as licitações que foram abertas, na área de informática, nos últimos 30 dias.

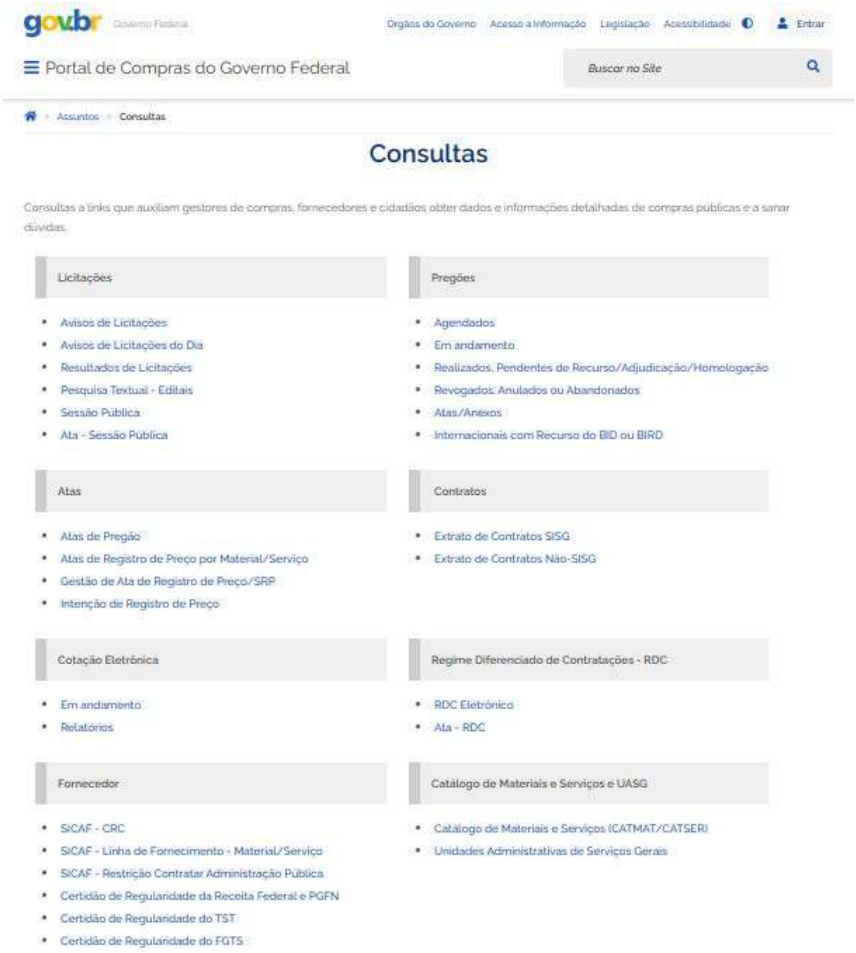


Imagem 4 - Tela do portal de compras do Governo Federal
Fonte: https://www.gov.br/compras/pt-br/assuntos/consultas-1/capa-consulta

Você conseguiu perceber o problema? Sabe é qual o grande problema dos exemplos negativos? Não foram projetados tendo o usuário em mente, e sim, o próprio desenvolvedor ou o gestor do ambiente. Problemas desse tipo interferem diretamente no que chamamos de **usabilidade.**

Vamos refletir?

Eu falei que um site tem uma boa interface, se a maioria dos usuários achá-la agradável e fácil de usar. Eu disse também que para que isso aconteça, a mesma tem que ser desenvolvida pensando no usuário. Isso parece óbvio, mas em muitos casos, o desenvolvedor (profissional que trabalha com web designer), cria uma página imaginando o que o usuário quer ver, sem perguntar para os usuários.

Entendeu? Por isso, digo que nem sempre o óbvio é feito.

É fácil dizer que uma página deve ter boa usabilidade, mas como fazer isso na prática? Será que existe algum documento que ajude o desenvolvedor a criar páginas com boa usabilidade?

A resposta é “Sim”. Esse documento é a Norma NBR 9241-11 .
Essa norma define que **usabilidade é**:

> A medida na qual um produto pode ser usado por usuários específicos para alcançar objetivos específicos com eficácia, eficiência e satisfação em um contexto específico de uso. Em outras palavras, qualquer interface tem uma boa usabilidade quando é fácil de usar e de aprender e quando faz aquilo que se propõe.

Saiba mais!

A ABNT – Associação Brasileira de Normas Técnicas – é o Fórum Nacional de Normalização. As Normas Brasileiras, cujo conteúdo é de responsabilidade dos Comitês Brasileiros (ABNT/CB) e dos Organismos de Normalização Setorial (ONS), são elaboradas por Comissões de Estudo (ABNT/CE), formadas por representantes dos setores envolvidos, delas fazendo parte: produtores, consumidores e neutros (universidades, laboratórios e outros).

Os Projetos de Norma Brasileira, elaborados no âmbito dos ABNT/CB, circulam para Consulta Pública entre os associados da ABNT e demais interessados.

A NBR 9241 consiste das seguintes partes, sob o título geral de Requisitos ergonômicos para trabalho de escritório com computadores. E a parte 11 refere-se às orientações sobre usabilidade.

Fonte: http://www.labiutil.inf.ufsc.br/cpqd-capacitacao/iso9241-11F2.doc

1.2 Acessibilidade

Quando eu disse que uma página deve ser fácil de usar, eu também me referi às pessoas com necessidades especiais, ou seja, devem ser considerados os critérios de acessibilidade. E você sabe o que é acessibilidade?

Segundo a Secretaria Especial dos Direitos da Pessoa com Deficiência, **acessibilidade** é:

> "Um atributo essencial do ambiente que garante a melhoria da qualidade de vida das pessoas. Deve estar presente nos espaços, no meio físico, no transporte, na informação e comunicação, inclusive nos sistemas e tecnologias da informação e comunicação, bem como em outros serviços e instalações abertos ao público ou de uso público, tanto na cidade como no campo".

Imagem 5 - Acessibilidade
Fonte:
https://pixabay.com/pt/illustrations/acessibilidade-deficiência-1682903/

Mas como fazer um site acessível? E como verificar sua acessibilidade?

Existem diversas referências de acessibilidade que podem ajudar você na construção de uma interface que seja, ao mesmo tempo, de alta usabilidade e acessível às pessoas com necessidades especiais.

O Governo Federal disponibiliza um site denominado eMAG - Modelo de Acessibilidade em Governo Eletrônico, que "tem o compromisso de ser o norteador no desenvolvimento e a adaptação de conteúdos digitais do Governo Federal, garantindo o acesso a todos".

Mesmo que você não esteja desenvolvendo um sistema para o Governo Federal, esse site ainda assim é uma referência, pois as recomendações do eMAG "permitem que a implementação da acessibilidade digital seja conduzida de forma padronizada, de fácil implementação, coerente com as necessidades brasileiras e em conformidade com os padrões internacionais".

Além disso, deve-se seguir as recomendações do World Wide Web Consortium (W3C), que é uma comunidade internacional que desenvolve padrões com o objetivo de garantir o crescimento da web.

Saiba mais!

Para mais informações acesse:
http://emag.governoeletronico.gov.br/

Saiba mais!

Sabe quem lidera o W3C?

Nada menos que o Tim Berners-Lee, o inventor da web!!!

O W3C desenvolve protocolos e diretrizes, de modo a garantir o crescimento da rede mundial de computadores (World Wide Web). Uma das maneiras de manter o crescimento é ajudar todas as pessoas a acessarem a rede mundial, inclusive as pessoas com necessidades especiais.

Tim Bernes-Lee
https://upload.wikimedia.org/wikipedia/commons/c/c2/Tim_Berners-Lee_2012.jpg

Saiba mais!

Modelo de Acessibilidade em Governo Eletrônico (eMAG)

O Ministério do Planejamento, Orçamento e Gestão, por meio da Secretaria de Logística e Tecnologia da Informação, elaborou um documento chamado "Modelo de Acessibilidade em Governo Eletrônico (eMAG)", com o compromisso de ser o norteador no desenvolvimento e na adaptação de conteúdos digitais do Governo Federal, garantindo o acesso a todos.

As recomendações do eMAG permitem que a implementação da acessibilidade digital seja conduzida de forma padronizada, de fácil implementação, coerente com as necessidades brasileiras e em conformidade com os padrões internacionais. É importante ressaltar que o eMAG trata de uma versão especializada do documento internacional WCAG (Web Content Accessibility Guidelines: Recomendações de Acessibilidade para Conteúdo Web) voltado para o governo brasileiro, porém o eMAG não exclui qualquer boa prática de acessibilidade do WCAG.

Fonte: https://www.gov.br/governodigital/pt-br/acessibilidade-digital/eMAGv31.pdf

Saiba mais!

A W3C disponibiliza uma cartilha, que apresenta vários critérios de usabilidade que você deve observar quando for desenvolver o seu site. A referida cartilha destaca os seguintes objetivos:

- O que é acessibilidade? Usar uma linguagem simples e de fácil entendimento, de modo que todos possam compreender o assunto acessibilidade;
- Quais são os grupos beneficiados com a acessibilidade? A cartilha da W3C apresenta os diferentes grupos de usuários com necessidades especiais e suas respectivas dificuldades quando tentam acessar a rede mundial;
- Como fazer um site acessível? A W3C apresenta recomendações e diretrizes que devem ser seguidas por desenvolvedores, de modo a minimizar ou eliminar as dificuldades de acesso dos grupos indicados no item anterior;
- Como avaliar um site quanto à acessibilidade? A referida cartilha apresenta orientações a respeito dos procedimentos que devem ser adotados para avaliar a acessibilidade de um site web;
- Como exigir acessibilidade? A W3C apresenta em sua cartilha algumas orientações sobre como devem proceder para cobrar a acessibilidade em sites web.

Fonte:
https://www.w3c.br/pub/Materiais/PublicacoesW3C/cartilha-w3cbr-acessibilidade-web-fasciculo-I.html

1.3 Legibilidade

Agora, vamos estudar mais um aspecto fundamental da arquitetura da informação: Legibilidade.

Você já percebeu que alguns sites usam cores que dificultam a leitura?

Eles usam uma combinação de cores de fundo que atrapalha a identificação das letras do texto apresentado no site, ou usam letras muito pequenas. Esses são problemas de legibilidade.

Segundo o dicionário Michaelis, legibilidade é a “qualidade de legível”.

Em outras palavras, um site é legível quando todas as informações apresentadas na tela podem ser lidas sem dificuldade.

Como que você garante legibilidade? Comprando um óculos para o leitor?? (risos)

É claro que não! Então, o que é preciso para garantir a legibilidade em um site?

Continue comigo, siga em frente que eu vou te explicar!

Para garantir a legibilidade em um site, você deve ficar atento, por exemplo, ao brilho do caractere, ao tamanho da fonte, ao contraste letra/fundo, ao espaçamento entre entre as linhas, ao espaçamento entre os parágrafos.

Veja o exemplo a seguir:

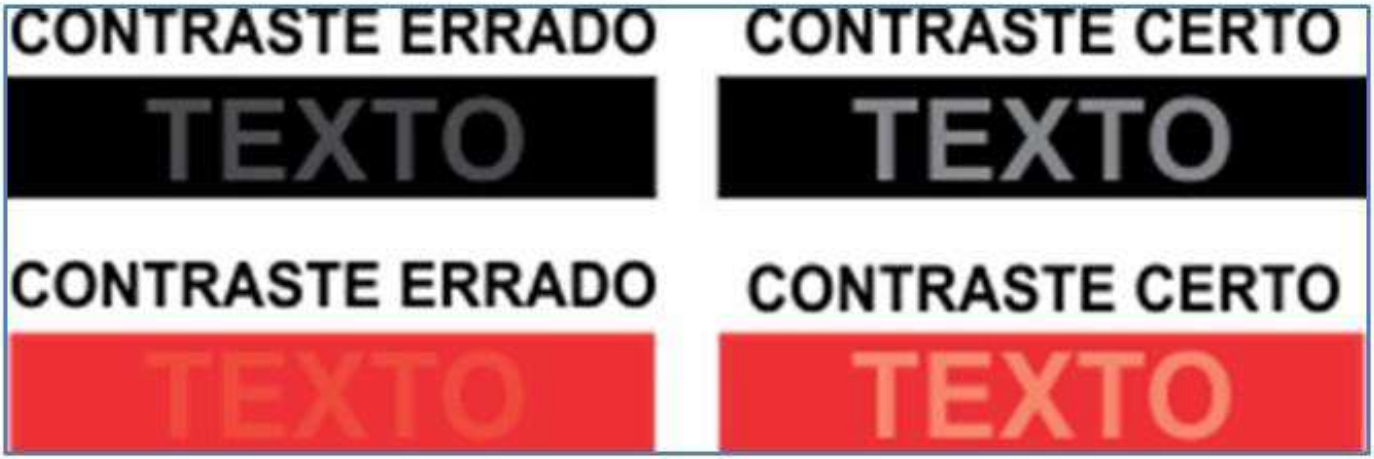


Imagem 6 - Legibilidade
Fonte: http://institucional.zapgrafica.com.br/wp-content/uploads/2012/05/dicas_especiais_efeitos_02.png

Quando você procura que seu site tenha um bom desempenho, deve facilitar a leitura da informação apresentada. Para isso, o desenvolvedor do site deve levar em conta duas características dos usuários: a primeira diz respeito ao cognitivo, que está relacionada com o processo de aquisição de conhecimento; a segunda está relacionada à percepção pelos sentidos.

Saiba mais!

O LabIUtil da UFSC (Universidade Federal de Santa Catarina) foi o laboratório de usabilidade criado em 1995 e desativado no final de 2003, quando apoiou empresas brasileiras produtoras de software interativo que buscavam a melhoria da usabilidade dos sistemas que produziam. Hoje, mesmo desativado, continua a disponibilizar uma lista de verificação de qualidades ergonômicas do software. O referido laboratório fez as seguintes recomendações gerais:

- Títulos devem ser centralizados;
- Rótulos devem estar em letras maiúsculas;
- Cursores devem se apresentar distintos dos outros itens;
- Quando o espaço para o texto for limitado, mostrar poucas linhas longas ao invés de muitas linhas curtas;
- Exibir texto contínuo em colunas largas de, ao menos, 50 caracteres por linha;
- A justificação à direita deve ser empregada se puder ser obtida por espaçamento, desde que sejam mantidos espaçamentos proporcionais constantes entre e nas palavras e espaçamento consistente entre palavras de uma linha;
- Ao exibir um material textual, mantenha as palavras intactas, com o mínimo de hífens.

Disponível em: http://www.labiutil.inf.ufsc.br/index.html

1.4 Fundamentos e Padrões de Desenho para Web

As imagens utilizadas nos sites têm evoluído bastante nas últimas décadas. Antes a gente podia exibir apenas textos. Agora podemos inserir fotos, áudios e vídeos. A velocidade de conexão evoluiu também, o que permite você ver a sua série favorita online.

Você conhece todos os principais formatos de imagens utilizados atualmente? Existem dezenas de formatos de imagens.

Vamos refletir?

Você verá que existem muitas possibilidades de aplicação. Por um lado isso é bom, pois lhe dá maior flexibilidade. Por outro lado, pode te deixar em dúvida sobre qual formato usar. Fique tranquilo!

O mais importante é saber as características de cada formato, e analisar as possibilidades de acordo com as necessidades do seu projeto.

Observe a tabela a seguir que apresenta uma comparação entre as características dos principais formatos de imagens, considerando a mesma imagem, com o tamanho de 840 x 560.

Tabela comparativa entre as características dos principais formatos de imagens

Formato	Principais Usos	Características	Tamanho do Arquivo
Bitmap (bmp)	Geral	• um dos formatos mais antigos e mais simples • trabalha com milhões de cores • arquivos muito grandes • não permite o efeito de fundo transparente	1361 KB
GIF	Animações e ícones	• trabalha com apenas 256 cores • permite o efeito de fundo transparente • arquivos pequenos (compactados sem perdas)	90 KB
JPEG JPG (1)	Geral	• boa qualidade de imagem • trabalha com imagens de 24 bits(ou cerca de 16,8 milhões de cores) • arquivos pequenos (compactados com perdas[1])	130 KB
PNG	Animações e ícones	• trabalha com milhões de cores • permite o efeito de fundo transparente • arquivos pequenos (compactados sem perdas) • exibe detalhes com mais nitidez que a JPEG, pois sua compactação é sem perdas • utilização apoiada pela W3C	384 KB

Tabela 1 - Características dos principais formatos de imagens
Fonte: elaborado pelo autor

(continua...)

RAW	Uso profissional de fotografia	• guarda todos os dados da imagem, tal como captada pelo sensor da câmera fotográfica, sem aplicação de efeitos ou ajustes	Muito grande (tamanho depende da máquina fotográfica)
SVG	Imagens estáticas e animadas	• imagens vetoriais • formato aberto • permite o efeito de fundo transparente • podem ser ampliadas ou reduzidas sem causar perda de qualidade	503 KB
TIFF	imagens médicas e industriais	• grande quantidade de cores • excelente qualidade de imagem • suporta o uso de camadas • permite o efeito de fundo transparente	428 KB
WebP	Geral	• formato de imagem desenvolvida pela Google • tamanho reduzido • rápido carregamento • boa qualidade de imagem • concorrente do JPEG	28 KB

Tabela 1 - Características dos principais formatos de imagens
Fonte: elaborado pelo autor

[1] JPEG: realiza a compactação da imagem original, com perdas de informação. Porém, se efetuada dentro de certos limites, as perdas não são perceptíveis ao olho humano.

1.5 Interfaces para Sites Web e Portais

A pesquisadora Camila Brandão da PUC-SP, destaca que apesar da importância de qualquer interface contemplar as necessidades e desejos dos usuários, existem muitos produtos digitais de difícil uso e pouco prazerosos para os usuários, representando uma baixa usabilidade.

A referida pesquisadora referência também os três principais problemas de interfaces mal idealizadas, a saber:

- Ignorância em relação aos usuários: não sabe ao certo quais são as características do usuário da interface e nem o que a mesma deveria ter para deixá-los felizes;

- O conflito de interesse entre as necessidades dos usuários e as prioridades de construção do produto digital: é comum ter um conflito entre o que pode ser feito, com o tempo disponível, com a equipe contratada e da forma mais fácil;

- Falta de processo para entender as necessidades dos usuários: nem sempre são estabelecidos processos para realmente entender o que o usuário final da interface precisa.

Saiba mais!

Como podemos criar sites que sejam agradáveis e fáceis de usar?

Abordaremos duas propostas básicas

- Design Thinking: a ideia principal é pensar numa solução utilizando diferentes pontos de vista, de forma colaborativa e coletiva, envolvendo todas as pessoas relacionadas com o projeto, desde os patrocinadores até os usuários finais. E tudo fica melhor se envolver pessoas com vivências e formações diferentes, em equipes multidisciplinares. Assim, teremos diferentes modos de ver os problemas, testando diversas ideias e encontrando possíveis soluções.

Leia este material da InovaGov sobre "Design Thinking". Disponível em: http://inova.gov.br/biblioteca/livro-confianca-criativa/

- User Experience (UX): o ponto focal dessa abordagem é como tornar a experiência do usuário da interface mais fácil e útil. Ou seja, focar na usabilidade, com a apresentação do conteúdo disposta de forma lógica, do ponto de vista da experiência do usuário.

Assista a este vídeo da Canaltech Startups e saiba um pouco mais sobre "O que é UX (User Experience)? E o que NÃO é UX?". Disponível em: https://www.youtube.com/watch?v=2hfu_F4cDRQ .

Questões de autoaprendizagem

Vamos pensar e praticar...

Vamos supor que você seja ligado no mundo da internet. Nesse caso, talvez seja fácil imaginar uma nova interface para um novo tablet, que acessaria sua série favorita do Netflix (Stranger Things??) ou os vídeos do seu Youtuber preferido (Whindersson Nunes??). Mas vamos tirar você da sua zona de conforto.

Gostaria que você imaginasse uma nova interface do tablet para um público específico, o idoso. Imagine as funcionalidades que esse tablet deveria ter e faça um esboço da tela inicial.

Solução...

Entendeu que se colocar no lugar do outro, no caso, o usuário, é necessário e deve ser feito com cuidado? Então, você precisa ouvir o usuário, conhecer o perfil da pessoa que irá utilizar, saber onde serão utilizados, conhecer as atividades que as pessoas estão realizando quando interagem com o produto (que pode ser um site) e imaginar como você poderia fazer para otimizar a interação do usuário com o sistema.

Uma proposta de solução seria a proposta a seguir...

Imagem 7 - TabSênior
Fonte: https://canaltech.com.br/tablet/conheca-o-primeiro-tablet-para-idosos-do-brasil-97290/

Vamos rever!

Nesta unidade você descobriu o que seria um bom projeto de interface. Você viu que um bom projeto deve ter o usuário como foco central. Parece óbvio, mas quantas vezes você já entrou em um site ruim, difícil de usar e chato para aprender.

Você também estudou que o conceito de usabilidade tem evoluído continuamente. Essa usabilidade envolve também aspectos de acessibilidade, garantindo o acesso às pessoas com necessidades especiais. E envolve também critérios de legibilidade.

Nesta unidade, você também conheceu alguns padrões de formato de imagens. Talvez você nunca tenha usado alguns desses formatos, mas agora já possível utilizar novas formas e fazer coisas bonitas, diferentes e ao mesmo tempo usuais, acessíveis e legíveis sempre com foco no usuário. Pode ter certeza!

Por fim, identificamos algumas estratégias para elaborar interfaces agradáveis e fáceis de usar.

Sites indicados

CARTILHA: ACESSIBILIDADE NA WEB - W3C BRASIL. Disponível em: <https://www.w3c.br/pub/Materiais/PublicacoesW3C/cartilha-w3cbr-acessibilidade-web-fasciculo-I.html>. Acesso em: 22 fev. 2021.

Laboratório de Utilizabilidade da Informática - Home Page. Disponível em: <http://www.labiutil.inf.ufsc.br/index.html>. Acesso em: 22 fev. 2021.

Modelo de editoração de Normas ABNT – LabIUtil. Disponível em: <http://www.labiutil.inf.ufsc.br/cpqd-capacitacao/iso9241-11F2.doc>. Acesso em: 22 fev. 2021.

Glossário

- Acessibilidade
  É um atributo essencial do ambiente que garante a melhoria da qualidade de vida das pessoas. Deve estar presente nos espaços, no meio físico, no transporte, na informação e comunicação, inclusive nos sistemas e tecnologias da informação e comunicação, bem como em outros serviços e instalações abertos ao público ou de uso público, tanto na cidade como no campo.

  Fonte:
  https://www.gov.br/mdh/pt-br/navegue-por-temas/pessoa-com-deficiencia/programas/acessibilidade

- Legibilidade
  Diz respeito às características lexicais das informações apresentadas na tela que possam dificultar ou facilitar a leitura desta informação (brilho do caractere, contraste letra/ fundo, tamanho da fonte, espaçamento entre palavras, espaçamento entre linhas, espaçamento de parágrafos, comprimento da linha, etc.).

  Fonte: http://www.labiutil.inf.ufsc.br/CriteriosErgonomicos/LabIUtil2003-Crit/140legib_A.html

- Usabilidade
  Estudo ou a aplicação de técnicas que proporcionem a facilidade de uso de um dado objeto, no caso, um sítio.

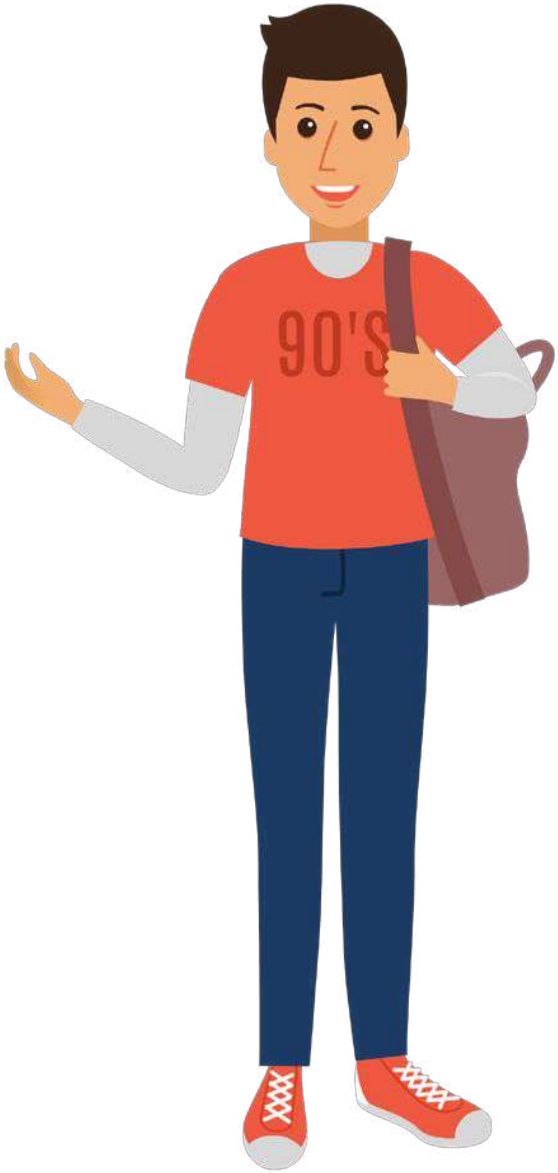
  Fonte: http://epwg.governoeletronico.gov.br/cartilha-usabilidade

Unidade 2

HyperText Markup Language - HTML

Olá! Leitor,
Que bom que você continua seus estudos!

Na primeira unidade, você aprendeu que devemos desenvolver interfaces que sejam agradáveis e fáceis de usar, que podem ser um site, uma página web. Para criar essa página, você deve usar uma linguagem específica, que permita que o navegador a reconheça. Assim, nesta segunda unidade aprenderemos uma dessas linguagens utilizadas para a construção de páginas web: HTML, uma abreviação para HyperText Markup Language, que em português podemos traduzir para Linguagem de Marcação de Hipertexto.



É importante esclarecer que você não será um desenvolvedor HTML profissional apenas com este curso, pois o objetivo não é esse, e sim fornecer o conhecimento necessário para customizar templates e criar páginas mais simples, o conhecimento básico que todo desenvolvedor web deve ter.

O estudo desta unidade 2, possibilitará a você alcançar os seguintes objetivos de aprendizagem:

- Citar as versões do HTML;
- Descrever os principais ambientes de desenvolvimento web;
- Definir os elementos básicos de uma página HTML;
- Relacionar às respectivas marcações suas funções na formatação básica de texto;
- Criar link entre páginas; e
- Inserir imagens, vídeos e tabelas em uma página HTML.

2.1 Histórico do HTML

Quando todo mundo começou a usar o HTML, as coisas estavam um pouco confusas, sem um padrão. Então, em 1990 o HTML foi formalizado como uma linguagem, seguindo determinados padrões. Nessa mesma linha, em 1994, Berners-Lee criou o famoso World Wide Web Consortium, também conhecido com W3C, que é um consórcio com cerca de 400 membros, cujo objetivo principal é a padronização da World Wide Web, também conhecida como rede mundial de computadores ou simplesmente web.

Segundo a W3C, sempre se buscou novos recursos para o HTML, que continuam a ser introduzidos para ajudar os autores de aplicativos da Web. Desse modo, novos elementos continuam a ser introduzidos com base na pesquisa sobre as práticas de autoria prevalecentes.

Com isso, com todas as contribuições, o HTML tem evoluído muito ao longo dos anos, acompanhando as novas necessidades e a evolução das tecnologias. Resumidamente, na tabela ao lado, apresentamos as principais versões do HTML.

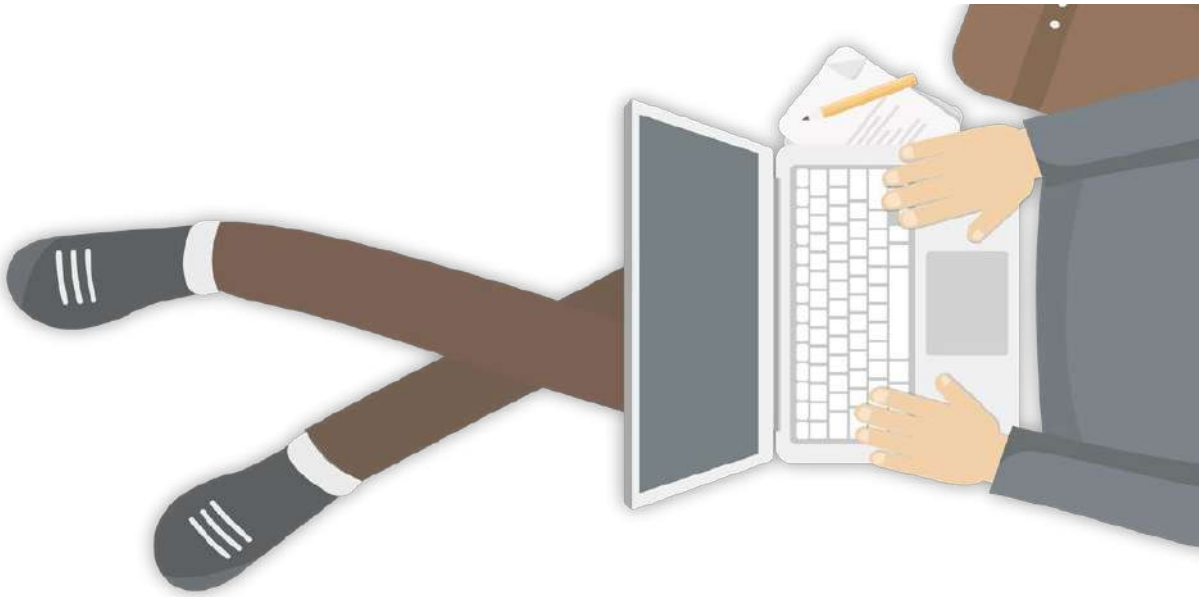
Versão	Ano
HTML 5.2	2017
HTML 5.1	2016 e 2017 (segunda edição)
HTML 5	2014
XHTML	2000
HTML 4.01	1999
HTML 3.2	1997
HTML 2.0	1995
HTML	1991

Tabela 1: versões do HTML
Fonte: elaborado pelo autor

Saiba mais!

O Consórcio World Wide Web (W3C) é um consórcio internacional, no qual organizações filiadas, uma equipe em tempo integral e o público trabalham juntos para desenvolver padrões para a Web. Liderado pelo inventor da web, Tim Berners-Lee e o CEO Jeffrey Jaffe, o W3C tem como missão conduzir a World Wide Web para que atinja todo seu potencial, desenvolvendo protocolos e diretrizes que garantam seu crescimento de longo prazo.

Fonte: http://www.w3c.br/Home/WebHome



Você sabia?

De onde surgiu essa ideia de internet? Será que um dia todo mundo acordou e passou a usar o WhatsApp?

Esse homem ao lado é um físico britânico, cientista da computação e professor do MIT chamado Timothy John Berners-Lee, que criou o HTML com o objetivo inicial de compartilhar suas pesquisas com seus colegas. Naquela época a página da internet não ia além de textos, com pouca interatividade, sem imagens ou vídeos. Mas foi o início de uma tremenda revolução no modo como nos comunicamos.

Será que você será o próximo Berners-Lee? Só o tempo dirá.

Tim Bernes-Lee
https://www.csaspeakersindia.in/media/pictures/profile/TIMBER.png

2.2 Editores

Eu já falei para você um pouco sobre a evolução do HTML.

Agora, para colocar em prática, vamos desenvolver um código HTML e vamos criar uma página web.
Por onde você deve começar? Afinal do que você precisa para criar um código HTML?

A rigor, basta ter um editor de texto simples (bloco de notas) para criar um código HTML (neste caso você deve salvar como um arquivo .html) e um navegador para visualizar o resultado do referido código. Existem dezenas de ambientes de desenvolvimento. Vou apresentar para você os principais.

Nome	Breve descrição	Onde encontrar
Adobe Dreamweaver	Este é provavelmente o mais famoso programa para desenvolvimento de páginas web.	http://www.adobe.com/br/products/dreamweaver.html
Notepad++	É um editor de código-fonte gratuito, regido pela Licença GPL[2] .	https://notepad-plus-plus.org/
Sublime Text	Semelhante ao Notepad++, porém com uma interface melhorada.	https://www.sublimetext.com/
Brackets	Um editor de códigos leve e limpo com ótimas funções de apoio.	http://brackets.io/
NetBeans	É um ambiente de desenvolvimento integrado gratuito e de código aberto para desenvolvedores de software de várias linguagens.	https://netbeans.org/downloads/

Tabela 2: ambientes de desenvolvimento HTML
Fonte: elaborado pelo autor

[2] GNU General Public License (Licença Pública Geral GNU), GNU GPL ou simplesmente GPL.
Fonte: http://www.gnu.org/

Testando os exemplos

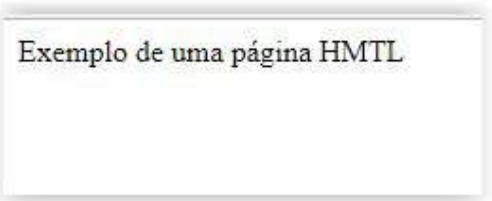
Para entender a teoria sobre HTML, é muito importante testar os exemplos e praticar. Vou apresentar como fazer esses testes, de uma maneira muito simples. Para isso, peço que você siga estes passos

1. Abra o aplicativo chamado “bloco de notas”
2. Escreva o código a seguir
3. Salve o arquivo com o nome exemplo.html
4. Dê dois cliques em cima do arquivo exemplo.html
5. O navegador padrão do seu computador será aberto e você verá o resultado.

Código HTML

```
<html>
  <body>
    Exemplo de uma página HMTL
  </body>
</html>
```

Resultado no navegador



2.3 Definições Iniciais

Agora que você já sabe que precisa praticar, vamos aprender alguns conceitos iniciais.

Por exemplo, como um navegador sabe o que é um texto simples e o que é um código HTML?

Os códigos HTML ficam entre as marcações (ou também chamadas tags) < e >, que geralmente são apresentadas aos pares, ou seja, <marcação> e </marcação>.

Exemplo:

<html> ... </html>

Entretanto, há algumas exceções para o funcionamento aos pares, que eu vou explicar à medida que elas forem aparecendo, mas já podemos citar a marcação que indica um parágrafo que não necessita, obrigatoriamente, de um .

Você sabia?

O navegador é um aplicativo que interpreta códigos em HTML, permitindo a interação do usuário com o conteúdo apresentado em uma página web. Esses aplicativos têm evoluído muito, acompanhando a evolução das tecnologias. Hoje existem diversos navegadores, desenvolvidos por diferentes empresas. Por exemplo:

Chrome da Google

Firefox da Mozilla

Internet Explorer da Microsoft

Safari da Apple

Existe um padrão estabelecido pela W3C e determinadas exigências de mercado, mas não existe uma lei que obrigue todos os desenvolvedores de navegadores a seguir tais padrões e exigências. E se o navegador não entende uma determinada marcação, ele simplesmente a ignora. Portanto, é importante realizar um cross-browser, ou seja, testar uma determinada página em diferentes navegadores.

Existem diversas soluções online que podem lhe ajudar a realizar esses testes. Veja alguns exemplos:

- Browsera. Disponível em: <https://turbo.net/browsers>
- Browserling. Disponível em: <https://www.browserling.com/>
- Browsershots. Disponível em: <http://browsershots.org/>
- IE NetRenderer. Disponível em: <http://netrenderer.com/index.php>
- Viewlike.us. Disponível em: <https://www.viewlike.us/>>

Além da marcação <html> existem outras, que podem ser consideradas as marcações mínimas de uma página. Preste atenção!

```
<!DOCTYPE html>
<html>
  <title>
    Exemplo
  </title>
  <head>

    <meta charset="UTF-8">
  </head>
  <body>
    Minha Página
  </body>
</html>
```

- define este documento como HTML5
- elemento raiz de uma página HTML
- especifica um título para o documento
- contém meta-informações sobre o documento
- define o conjunto de caracteres a ser usado
- local do conteúdo da página visível

Você sabia?

Apesar de existir um padrão, na hora em que você está digitando seu código HTML, pode acontecer de você trocar alguma letra maiúscula pela equivalente minúscula e vice-versa. Não esquente a cabeça com isso!!! As marcações HTML não são “case sensitive”, não diferenciam letras maiúsculas de minúsculas. Em outras palavras, <HTML> significa o mesmo que <html>.

2.4 Elementos Básicos de uma Página HTML

Vamos agora estudar os elementos básicos de uma página HTML. São várias marcações que aparecem com maior frequência.

2.4.1 Cabeçalhos

No contexto do HTML, você utiliza cabeçalhos para definir um destaque para um determinado texto e podem ser definidos 6 níveis de cabeçalhos. Segue a codificação básica:

<Hnúmero>...</Hnúmero>

Exemplo:

Código HTML

```
<!DOCTYPE html>
<html>
   <title>Título da página</title>
   <head>
      <meta charset="UTF-8">
   </head>
   <body>
      <h1> destaque </h1>
      <h2> destaque </h2>
      <h3> destaque </h3>
      <h4> destaque </h4>
      <h5> destaque </h5>
      <h6> destaque </h6>
   </body>
</html>
```

Resultado no navegador

2.4.2 Parágrafos

Agora eu vou explicar como são inseridas quebras de parágrafo no HTML. Veja, a seguir:

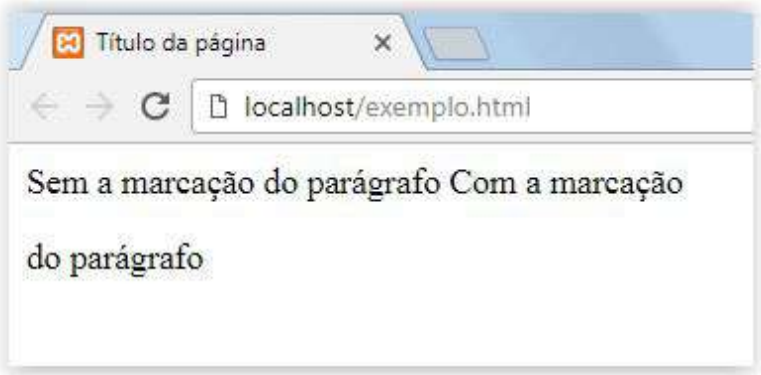
O texto de uma página web geralmente é dividido em parágrafos, como normalmente a gente faz fora da web. Entretanto, os navegadores não reconhecem as quebras de parágrafos que você coloca no código HTML.

Os navegadores inserem uma quebra de linha ou a marcação de um novo parágrafo apenas quando o texto chega no final da janela. Para inserir a quebra de um novo parágrafo manualmente, você precisa inserir a marcação , preferencialmente seguida da marcação . Observe o quadro ao lado.

Código HTML

```
<!DOCTYPE html>
<html>
  <title>Título da página</title>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    Sem a marcação do parágrafo
    Com a marcação <p> do parágrafo
  </body>
</html>
```

Resultado no navegador



2.4.3 Quebra de linha

Vamos refletir?

O espaço entre linhas não pode ser exagerado. Talvez você não queira colocar o espaço extra nas linhas, que aparece sempre que você usa a marcação <p>.

O que fazer?

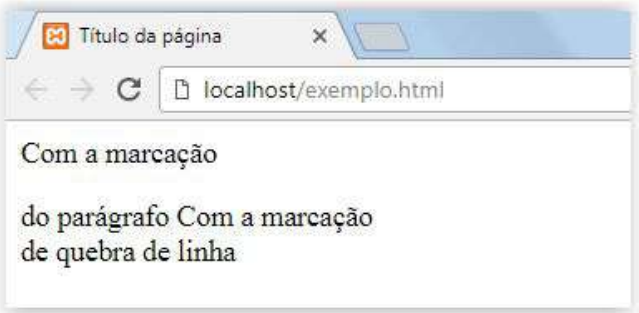
Neste caso, você precisa usar a marcação
, que quebra a linha, sem acrescentar um espaço extra entre as linhas.

Exemplo:

Código HTML

```
<!DOCTYPE html>
<html>
   <title>Título da página</title>
   <head>
      <meta charset="UTF-8">
   </head>
   <body>
      Com a marcação <p> do parágrafo
      Com a marcação <br> de quebra de linha
   </body>
</html>
```

Resultado no navegador



2.5 Formatando Textos

Tão importante quanto o texto que você apresenta na página web é a formatação e o destaque com que você apresenta o referido texto.

A linguagem HTML permite inserir diversas formatações em um texto qualquer. As principais marcações relacionadas à formatação são as seguintes:

Marcação	Função
<b>	Colocar o texto em negrito.
	Visualmente faz o mesmo que a marcação b, basicamente é apenas uma recomendação mais nova.
<i>	Coloca o texto em itálico.
	Visualmente faz o mesmo que a marcação i, basicamente é apenas uma recomendação mais nova.
<mark>	Coloca marcado com um uma cor de destaque.
<small>	Coloca o texto em destaque, diminuindo seu tamanho em relação ao tamanho padrão.
<del>	Coloca um traço sobre o texto para marcar que o referido trecho foi excluído do texto original.
<ins>	Coloca um traço sob o texto para marcar que o referido trecho foi incluído no texto original.
<sub>	Coloca o texto subescrito.
<sup>	Coloca o texto sobreescrito.

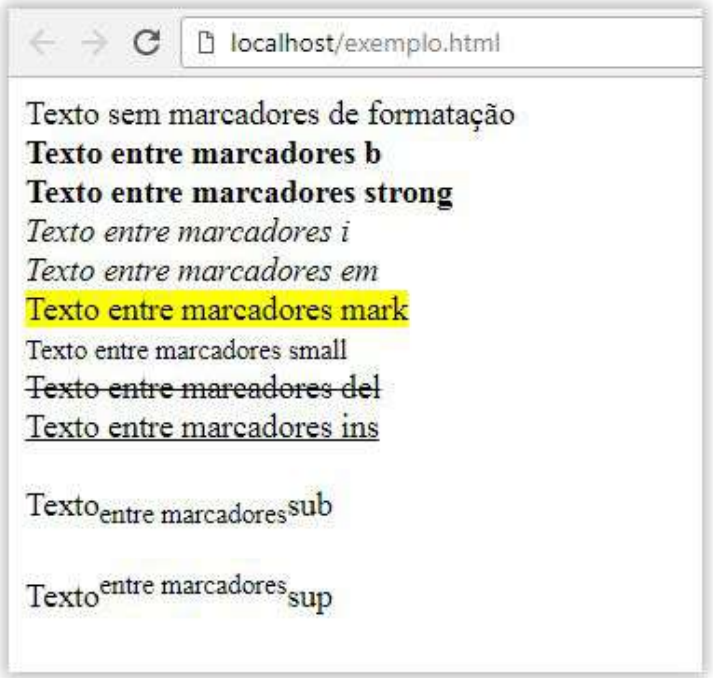
Tabela 3: Formatações básicas de texto
Fonte: elaborada pelo autor

Exemplo:

Código HTML

```
<!DOCTYPE html>
<html>
   <title>Título da página</title>
   <head>
      <meta charset="UTF-8">
   </head>
   <body>
      Texto sem marcadores de formatação<br>
      <b>Texto entre marcadores b</b><br>
      <strong>Texto entre marcadores strong</strong><br>
      <i>Texto entre marcadores i</i><br>
      <em>Texto entre marcadores em</em><br>
      <mark>Texto entre marcadores mark</mark><br>
      <small>Texto entre marcadores small</small><br>
      <del>Texto entre marcadores del</del><br>
      <ins>Texto entre marcadores ins</ins><br><br>
      Texto<sub>entre marcadores</sub>sub<br><br>
      Texto<sup>entre marcadores</sup>sup
   </body>
</html>
```

Resultado no navegador



2.6 Ligando Uma Página a Outra

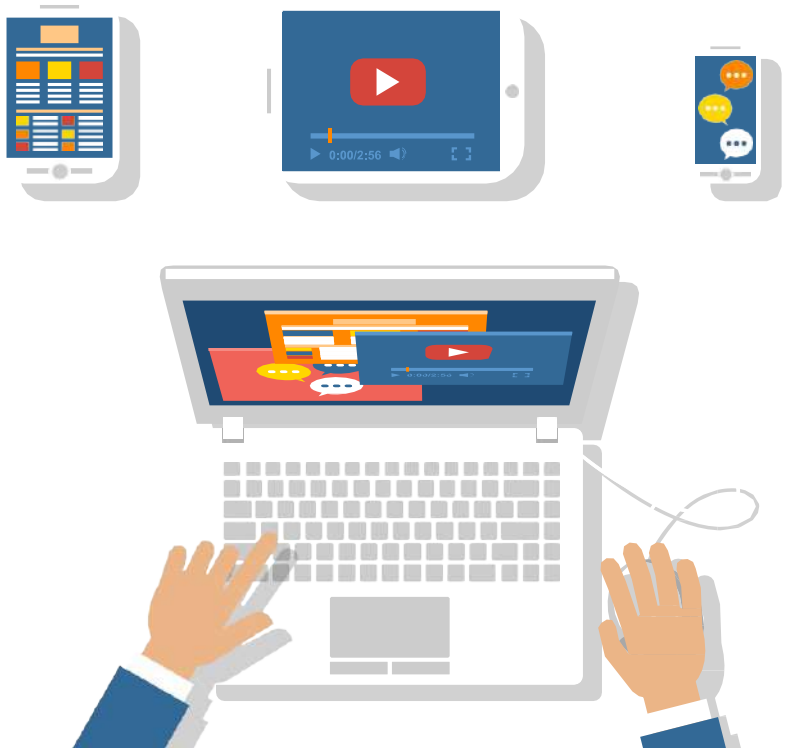
Imagine algo comigo! Você está na página do Youtube, vê um vídeo legal, talvez do seu Youtuber preferido e clica no link do vídeo. Então, a página com o vídeo que você selecionou aparece no navegador.

Essa é uma das grande vantagens do HTML: a possibilidade de você poder navegar pelas diferentes páginas, sem ter que seguir uma ordem pré-definida e sim o interesse do usuário.

Mas como fazer isso?

Usando a marcação de âncora.

A seguir, veja o passo a passo para criar um link para uma outra página.



Passo a passo para criar um link para uma outra página:

1. Utilizar a marcação <a>
2. Utilizar o atributo href para indicar qual página será “linkada”
3. Definir o texto ou imagem que servirá de link
4. Fechar com a marcação

Exemplo:

Página 1

```
<!DOCTYPE html>
<html>
  <title>Página 1</title>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <h1>Página 1</h1>
    Link para a
      <a href="pagina2.html">
      página 2
      </a>
  </body>
</html>
```

Página 2

```
<!DOCTYPE html>
<html>
  <title>Página 2</title>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <h1>Página 2</h1>
    Link para a
      <a href="pagina1.html">
      página 1
      </a>
  </body>
</html>
```

Resultado no navegador

Resultado no navegador

Você sabia?

O chamado Hypertext Transfer Protocol (ou na sigla HTTP, que em tradução livre do inglês significa Protocolo de Transferência de Hipertexto) é um protocolo de comunicação (na camada de aplicação segundo o Modelo OSI) utilizado para sistemas de informação de hipermídia distribuídos e colaborativos.

(Fonte: https://www.portaleducacao.com.br/conteudo/artigos/idiomas/o-protocolo-http/20408)

2.7 Inserindo Imagens

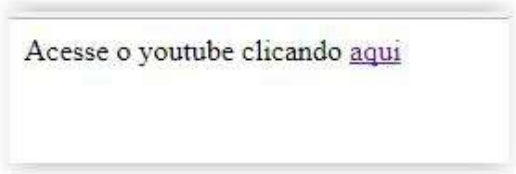
Mas... uma página não deveria ser feita apenas de textos, correto? E uma imagem pode dizer mais que 1000 palavras. Para inserir uma imagem em uma página HTML basta utilizar a marcação <img> e alguns atributos,a saber:

- src: define o nome do arquivo da imagem;
- alt: define o texto que será apresentado, no lugar da imagem, quando a imagem não pode ser exibida;
- width: define a largura da imagem (em pixels);
- height: define a altura da imagem (em pixels).

Usando uma imagem local

```
<!DOCTYPE html>
<html>
   <title>Página 2</title>
   <head>
      <meta charset="UTF-8">
   </head>
   <body>
      Acesse o youtube clicando
      <a href="https://www.youtube.com/">
         aqui
      </a>
   </body>
</html>
```

Resultado no navegador

Antes de clicar	Depois de clicar
	

2.8 Inserindo Vídeos

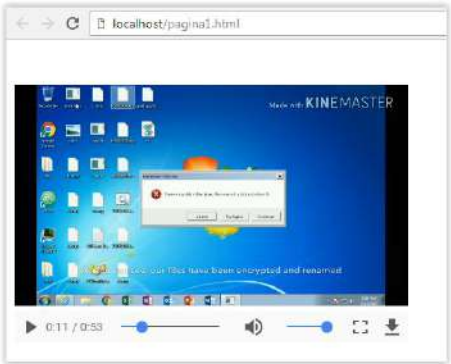
Por falar em Youtube, esse é um dos maiores sites de vídeos do mundo, onde todos podem se divertir, aprender e, muitas vezes, se chocar. E se você quiser inserir um vídeo na sua página? É fácil. Vamos aprender a fazer?

Com o HTML 5 a inserção de vídeos em uma página web ficou muito mais fácil. Você precisa utilizar basicamente duas marcações:

- <video>: que define que você irá inserir um vídeo;
- <source>: onde você especifica o formato do vídeo;
- <iframe>: use para incorporar um outro documento dentro do HTML.

Junto com essas marcações, você pode utilizar alguns atributos que podem ser muito úteis, a saber:

- width: você especifica a largura da janela do vídeo;
- height: você especifica a altura da janela do vídeo;
- controls: você permite que os controles de reprodução, pausa e volume sejam visualizados;
- autoplay: você define que o vídeo seja inicializado automaticamente;
- src: você especifica onde encontrar o arquivo do vídeo, seja numa pasta local ou na web;
- allowfullscreen: você especifica que o vídeo pode ser visto em tela cheia;
- type: você especifica o tipo de arquivo. Atualmente, os principais tipos de arquivos são os seguintes:
  - ➔ mp4: formato padrão de compactação de áudio e vídeo;
  - ➔ ogg: um formato orientado a stream, ou seja, não precisa fazer o download de todo o vídeo para começar a vê-lo;
  - ➔ webm: um formato de alta qualidade desenvolvido pela Google.

Usando um vídeo local	Usando o link para um vídeo na web
<!DOCTYPE html> <html> <title>Exemplo vídeo</title> <head> <meta charset="UTF-8"> </head> <body> <video width="400" height="300" allofullscream controls autoplay> <source src="Wanna_cry_virus_in_action.mp4"> </video> </body> </html>	<!DOCTYPE html> <html> <title>Exemplo vídeo</title> <head> <meta charset="UTF-8"> </head> <body> <iframe width="560" height="315" src="https://www.youtube.com/embed/kJQP7kiw5Fk" frameborder="0" allowfullscreen> </iframe> </body> </html>
Resultado no navegador	**Resultado no navegador**
	

2.9 Criando Tabelas

E se você quiser apresentar alguns dados em página de forma organizada e ordenada, divididos em colunas e linhas? Por exemplo, um site que apresenta os resultados do Brasileirão. Nesse caso você poderia apresentar os dados na forma de uma tabela do HTML.
Agora você vai aprender a criar tabelas.

Para criar uma tabela, você precisa usar pelo menos as seguintes marcações:

- <table>: define onde começa e termina uma tabela;
- <tr>: define uma nova linha da tabela;
- <th>: define uma célula especial, do tipo cabeçalho, com destaque;
- <td>: define uma célula de dados.

Além disso você pode utilizar os seguintes atributos:

- border: permite incluir as linhas de borda;
- align: permite definir o alinhamento ao centro (center), à esquerda (left) ou à direita (right);
- rowspan: permite mesclar duas linhas;
- colspan: permite mesclar duas colunas.

Código HTML

```
<!DOCTYPE html>
<html>
   <title>Exemplo vídeo</title>
   <head>
      <meta charset="UTF-8">
   </head>
   <body>
      <table border="1">
         <tr>
            <td rowspan="2">
                                                                  <img
src="https://c1.staticflickr.com/1/28/49684002_3890beba97.jpg"
                  width="200"
                  height="300">
            </td>
            <td align="left">Hermione Granger</td>
            <td align="right">Ronald Weasley</td>
         </tr>
         <tr>
            <td colspan="2" align="center">Harry Potter</td>
         </tr>
      </table>
   </body>
</html>
```

Resultado no navegador

Questões de autoaprendizagem

Vamos pensar e praticar...

Sei que vocês têm seus programas e séries favoritas, seus sites favoritos, as coisas com as quais você consegue passar horas vendo e se divertindo. Vamos contar isso para todo mundo?

Crie uma página reproduzindo a página a seguir. Seja criativo e pode mudar o conteúdo e os temas, mas não a estrutura. Tenho certeza de que você consegue, pois é uma pessoa muito inteligente. Atenção: as imagens são apenas ilustrativas.

Coisas que eu gosto	
Meu desenho favorito	
Meu game preferido	
Meu youtuber preferido	

Solução...

Para resolver essa questão, você deve primeiro pensar na estrutura da tabela e na forma que você quer apresentar. Depois, pesquisar algumas imagens que são interessantes para você. Por fim, deveria escrever um código semelhante a este:

```
<!DOCTYPE html>
<html>
    <title>
        Exemplo
    </title>
    <head>
        <meta charset="UTF-8">
    </head>
    <body>
        <table border>
            <tr>
                <th colspan="2">Coisas que eu gosto</th>
            </tr>
            <tr>
                <td>Meu desenho favorito</td>
                <td>
                    <img width="200"
                        height="200"
                            src="https://i.ytimg.com/vi/5_f5Nmt1pRE/maxresdefault.jpg">
                </td>
            </tr>
            <tr>
                <td>Meu game preferido</td>
                <td>
```

(continua...)

```
                <img width="200"
                    height="200"
            src="http://cdn.edgecast.steamstatic.com/steam/apps/730/header.jpg?t=1513742714"
            </td>
        </tr>
        <tr>
            <td>Meu youtuber preferido</td>
            <td>
                <img width="200"
                    height="200"
                    src="http://s2.glbimg.com/8BrYj3OetdqHBzbUammebQ2o4VA=/620x465/s.glbimg.
com/jo/g1/f/original/2016/06/27/whindersson1.png"
            </td>
        </tr>
        <tr>

        </tr>
    </table>
</body>
</html>
```

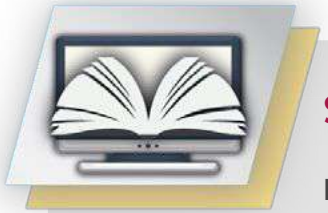
Vamos rever?

Nesta unidade você conheceu um breve histórico sobre o HTML e suas versões. Também estudou sobre alguns ambientes de desenvolvimento web, tais como o Adobe Dreamweaver e o Brackets. Agora você já sabe que os elementos básicos de uma página HTML são: cabeçalhos, parágrafos e quebra de linhas.

Você também aprendeu sobre formatações básicas de texto relacionando as marcações às suas devidas funções. Por fim, creio que você já consegue criar link entre páginas, inserir imagens e tabelas, tudo isso usando o HTML.

Agora você está preparado (a) para a próxima lição, em que aprenderá a formatar a sua página, usando uma folha de estilos.

Continue estudando...



Sites indicados

HTML5 Tutorial. Disponível em: <https://www.w3schools.com/html/> Acesso em: 14 jan. 2018.

W3C Brasil. Disponível em: <http://www.w3c.br/Home/WebHome> Acesso em: 14 jan.2018.

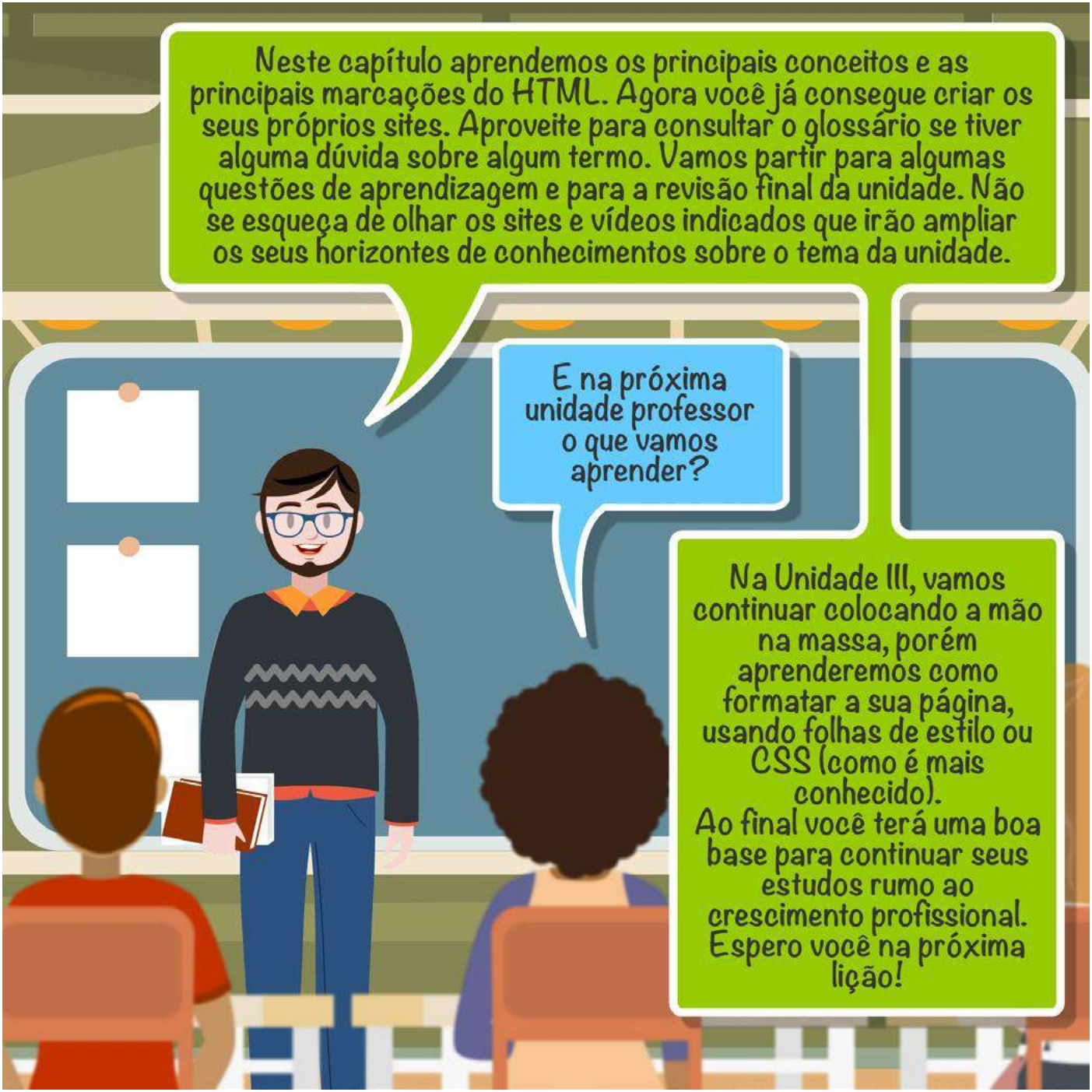
Glossário

- HTML
  HyperText Markup Language ou Linguagem de Marcação de HiperTexto é o construtor de blocos mais básico da web. Ele serve para descrever e definir o conteúdo de uma página web.

  Fonte: https://developer.mozilla.org/pt-BR/docs/Web/HTML

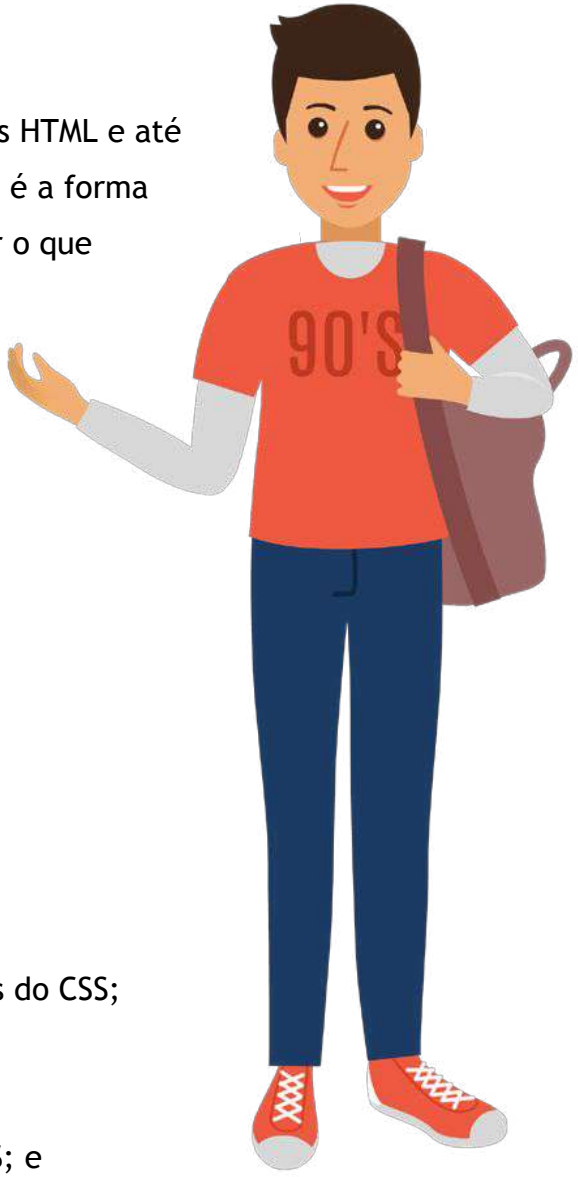
- HTTP
  O chamado Hypertext Transfer Protocol (ou na sigla HTTP, que em tradução livre do inglês significa Protocolo de Transferência de Hipertexto) é um protocolo de comunicação (na camada de aplicação segundo o Modelo OSI) utilizado para sistemas de informação de hipermídia distribuídos e colaborativos. Seu uso para a obtenção de recursos interligados levou ao estabelecimento da World Wide Web.

  Fonte: https://www.portaleducacao.com.br/conteudo/artigos/idiomas/o-protocolo- http/20408


Neste capítulo aprendemos os principais conceitos e as principais marcações do HTML. Agora você já consegue criar os seus próprios sites. Aproveite para consultar o glossário se tiver alguma dúvida sobre algum termo. Vamos partir para algumas questões de aprendizagem e para a revisão final da unidade. Não se esqueça de olhar os sites e vídeos indicados que irão ampliar os seus horizontes de conhecimentos sobre o tema da unidade.
E na próxima unidade professor o que vamos aprender?
Na Unidade III, vamos continuar colocando a mão na massa, porém aprenderemos como formatar a sua página, usando folhas de estilo ou CSS (como é mais conhecido).
Ao final você terá uma boa base para continuar seus estudos rumo ao crescimento profissional.
Espero você na próxima lição!

Unidade 3

INTRODUÇÃO AO CSS



Olá novamente!!!

Na Unidade II, eu expliquei para você sobre como criar códigos HTML e até um pouquinho sobre alguma formatação, entretanto, essa não é a forma mais prática e produtiva de uma página web. O melhor é usar o que chamamos de folha de estilos.

Por isso, nesta unidade, você verá como utilizar as folhas de estilo em cascata Style Sheets, ou simplesmente CSS. Assim, você poderá facilmente aplicar formatação a todas as páginas de um site qualquer, o que também facilita ajustes e eventuais manutenções, pois faz a correção em um único lugar que se repete em todos os sites que usam o mesmo arquivo CSS.

Ao final desta unidade, você poderá:

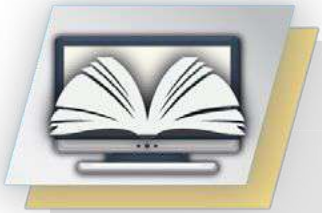
- Identificar os principais conceitos sobre folhas ;
- Identificar os principais editores;
- Usar classes e id's,
- Aplicar as diferentes maneiras de incluir as formatações do CSS;
- Identificar as diversas formas de cores;
- Alterar os padrões de fundo;
- Formatar textos e fontes usando as propriedades do CSS; e
- Fazer animações simples usando o CSS.

3.1 Conceito de CSS

Passamos agora a analisar em maiores detalhes a definição de CSS (Cascading Style Sheets).

As folhas de estilo especificam como os elementos de uma página HTML serão exibidos na tela, separando o conteúdo do formato. Ao utilizarmos o CSS temos os seguintes benefícios:

- Melhorar o controle da aparência de cada elemento HTML;
- Facilitar a manutenção, pois podemos fazer a alteração em um único arquivo CSS e utilizar a mesma formatação em todas as páginas de um portal;
- Permitir o desenvolvimento de páginas mais leves, facilitando o seu carregamento.



Saiba mais!

Bootstrap é o mais popular framework HTML, CSS e JS para desenvolvimento de projetos responsivo e focado para dispositivos móveis na web. Um site responsivo permite que o conteúdo se adapte ao tamanho da tela, mudando a aparência e disposição dos elementos.

Veja as configurações CSS em geral, classes CSS para elementos HTML fundamentais e um poderoso sistema de grid.

Fonte: http://getbootstrap.com.br/

3.2 Editores CSS

Agora que já falamos sobre a importância do CSS, vamos aprender os códigos e colocar em prática. Vamos formatar uma página web usando o CSS.

Por onde você deve começar? Afinal, qual aplicativo você precisa usar para criar um código CSS?

A rigor, basta ter um editor de texto simples (bloco de notas) para criar um código CSS (neste caso você deve salvar como um arquivo .css) e um navegador para visualizar o resultado do referido código. Existem dezenas de ambientes de desenvolvimento. Vou apresentar para você os principais.

Nome	Breve descrição	Onde encontrar?
Adobe Dreamweaver	Famoso programa para desenvolvimento de páginas web.	http://www.adobe.com/br/products/dreamweaver.html
Notepad++	Editor de código-fonte gratuito, regido pela Licença GPL[1] .	https://notepad-plus-plus.org/
Sublime Text	Semelhante ao Notepad++, porém com uma interface melhorada.	https://www.sublimetext.com/
Brackets	Um editor de códigos leve e limpo com ótimas funções de apoio.	http://brackets.io/
NetBeans	Ambiente de desenvolvimento integrado gratuito e de código aberto para desenvolvedores de software de várias linguagens.	https://netbeans.apache.org/download/index.html

Tabela 1: ambientes de desenvolvimento CSS
Fonte: elaborado pelo autor

[1] GNU General Public License (Licença Pública Geral GNU), GNU GPL ou simplesmente GPL.
Fonte: http://www.gnu.org/

Exemplo:

Peço que você teste todos os exemplos que mostraremos. Para isso, faça o seguinte:

1. Abra a primeira janela do aplicativo chamado “bloco de notas”;
2. Escreva o código HTML a seguir;
3. Salve o arquivo com o nome exemplo.html;
4. Abra uma segunda janela do “bloco de notas”;
5. Salve o arquivo com o nome teste.css;
6. Dê dois cliques em cima do arquivo exemplo.html
7. O navegador padrão do seu computador será aberto e você verá o resultado.

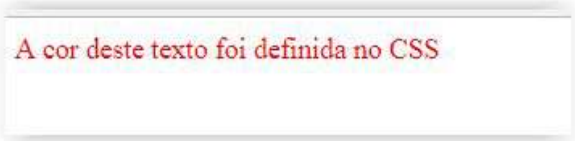
Código HTML

```
<!DOCTYPE html>
<html>
   <title>Exemplo CSS</title>
   <head>
      <meta charset="UTF-8">
      <link href="teste.css"
          type="text/css"
          rel="stylesheet">
   </head>
   <body>

A cor deste texto foi definida no CSS
   </body>
</html>
```

Código CSS

```
body{
   color:red;
}
```

Resultado na tela



3.3 Sintaxe Básica e Seletores

Você, uma pessoa inteligente, já entendeu o que precisa usar para trabalhar com o CSS. Agora, vamos conhecer a estrutura essencial desse tipo de código.

A sintaxe básica do CSS é composta basicamente de um seletor e um bloco de declarações, delimitado por chaves. O seletor indica o elemento do HTML que será formatado.

No bloco de declarações, temos o seguinte:

- uma ou mais propriedades, seguida de dois pontos;
- o valor de cada propriedade, seguida de ponto e vírgula.

No primeiro exemplo apresentamos o seguinte:

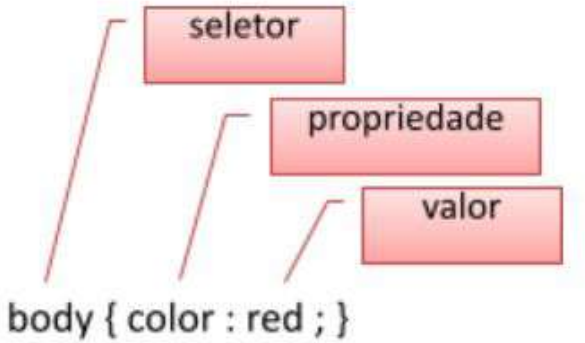


Imagem 1
Fonte: elaborado pelo autor

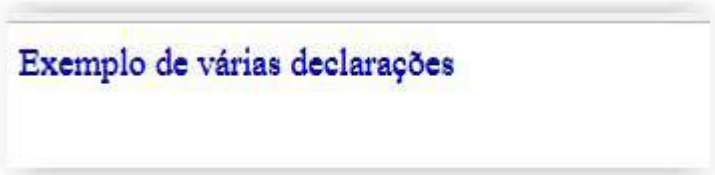
Em outras palavras, toda página HTML que referenciar esse código CSS apresentará em vermelho todos os textos que estiverem entre as marcações <body> e </body>.

Se você quiser fazer várias declarações para o mesmo seletor, ou seja, se quiser definir várias formatações para a mesma marcação do HTML, precisa separar cada declaração por um ponto e vírgula.

Código HTML	Código CSS

<!DOCTYPE html>
<html>
<title>Exemplo CSS</title>
<head>
<meta charset="UTF-8">
<link href="teste.css"
type="text/css"
rel="stylesheet">
</head>
<body>
Exemplo de várias declarações
</body>
</html>
	body{
	color:blue;
	text-shadow: 0px 1px black;
	}

Resultado na tela



3.4 Classes e ID's

Você conheceu a sintaxe básica do CSS. Agora, vamos aprender como usar algumas estruturas que permitem a aplicação de determinada formatação utilizando classes e id's.

Você sabia?

"Em HTML e CSS, há a possibilidade de aplicar estilos através de 'class' e 'id' e, em JavaScript é possível identificar algum elemento de uma página por sua classe, id ou tag. Mas qual a diferença entre 'class' e 'id'?

As classes são uma forma de identificar um grupo de elementos. Através delas, pode-se atribuir formatação a VÁRIOS elementos de uma vez.

As ids são uma forma de identificar um elemento e devem ser ÚNICAS para cada elemento. É como se fossem impressões digitais de nossos dedos ou RGs. Através delas, pode-se atribuir formatação a um elemento em especial".

Fonte: http://tableless.github.io/iniciantes/manual/css/class-id.html

3.4.1 Usando o id

Se você definir no CSS que os parágrafos serão na cor vermelha, todos serão vermelhos. E se você quiser que um parágrafo específico seja na cor azul?

Nesse caso, você poderia usar um id, que deve ser único, como o CPF de um brasileiro, que é único. Você define a formatação de um determinado id no CSS e utiliza o referido id com um atributo de uma marcação do HTML. Para criar um id, basta fazer o seguinte:

- **No CSS:** colocar o símbolo # seguido de um nome que será usado no HTML para identificar o id.
- **No HTML:** colocar o atributo id seguido do sinal = e o nome identificador entre aspas duplas.

Exemplo:

Código HTML

```
<!DOCTYPE html>
<html>
   <title>Exemplo CSS</title>
   <head>
      <meta charset="UTF-8">
      <link href="teste.css"
          type="text/css"
          rel="stylesheet">
   </head>
   <body>
      PERSONAGENS DE DRAGON BALL Z
      <p>Goku</p>
      <p id="identificador1">Vegeta</p>
      <p id="identificador2">Freeza</p>
   </body>
</html>
```

Código CSS

```
p{
   color:pink;
}
#identificador1{
   color:blue;
}
#identificador2{
   color:red;
}
```

Resultado na tela

Se usar o mesmo id para mais de um marcador, seu código não vai passar pelo validador. Segundo a W3C[2], a maioria dos documentos da Web são escritos usando linguagens de marcação, como o HTML, que são definidas por especificações técnicas, que geralmente incluem uma gramática formal (e vocabulário) legível por máquina. O ato de verificar um documento contra essas restrições é chamado de validação, e isso é o que o Validador de Marcação faz. A validação de documentos da Web é um passo importante que pode ajudar dramaticamente a melhorar e a garantir a sua qualidade, e pode economizar muito tempo e dinheiro.

3.4.2 Usando o class

Se você quiser agrupar características semelhantes, deve usar classes. Assim, você poderia aplicar a mesma formatação diferenciada em diferentes marcações. Pode parecer um pouco complicado entender, mas na prática é bem mais fácil.

Para usar uma classe, você define a formatação de uma determinada classe no CSS e a utiliza com um atributo de uma marcação do HTML. Para criá-la, basta fazer o seguinte:

- **No CSS:** colocar o símbolo . seguido de um nome que será usado no HTML para identificar a classe.
- **No HTML:** colocar o atributo class seguido do sinal = e o nome identificador entre aspas duplas.

[2] World Wide Web Consortium (W3C) é uma comunidade internacional onde membros de organizações, funcionários em tempo integral e o público em geral trabalham em conjunto para desenvolver padrões da Web .
Fonte: https://www.w3.org/Consortium/

Exemplo:

Código HTML	Código CSS

<!DOCTYPE html>
<html>
<title>Exemplo CSS</title>
<head>
<meta charset="UTF-8">
<link href="teste.css"
type="text/css"
rel="stylesheet">
</head>
<body>
PERSONAGENS DE DRAGON BALL Z
 Goku
 Vegeta
 Freeza
</body>
</html>
	p{
	color:red;
	}
	.minhaclasse{
	color:blue;
	}

Resultado na tela

3.5 Maneiras de Incluir Estilos

Imagine a seguinte situação: você foi contratado(a) para desenvolver um grande portal. Parabéns!!! Então, você terá que criar e formatar 45 páginas HTML. Imagine ter que formatar cada página individualmente. E o problema se agrava quando estamos na fase de homologação, quando temos que testar e corrigir todas as páginas. Um trabalho maior do que a batalha entre Goku e Freeza, que durou 20 episódios (risos).

Nesse caso, seria melhor você entender as diferentes formas de inserir as formatações do CSS: linking, embedding e inline.

Você já entendeu como usar classes e id's. Parabéns!!! Você é uma pessoa inteligente. Agora vamos aprender que podemos inserir as formatações do CSS de diferentes formas.

Você pode formatar um documento HTML de acordo com as informações de uma folha de estilo (CSS), que podem ser inseridas de três formas diferentes, a saber:

- **Linking (externa ou lincada):** carrega um arquivo CSS, que pode ser usado em várias páginas, utilizando a marcação link;
- **Embedding (interna ou incorporada):** define um código CSS na própria página HTML, sem afetar outras páginas, utilizando a marcação style.
- **Inline (na linha):** define a formatação do CSS apenas em uma marcação específica, utilizando a propriedade style.

Você sabia?

Imagine a seguinte situação: você criou um site com dezenas de páginas HTML. Em cada página você colocou um link para um arquivo externo de CSS.

Entretanto, em uma única página você quer fazer uma alteração específica, ou seja, não quer mudar tudo, apenas um detalhe. Nesse caso, qual regra será seguida? A regra do arquivo externo ou a regra da página específica?

Em ordem de hierarquia, a formatação inline se sobrepõe à formatação incorporada, que se sobrepõe à formatação lincada.

CSS Externo	CSS Incorporado	CSS inline
<!DOCTYPE html> <html> <title>Exemplo CSS</title> <head> <meta charset="UTF-8"> <link href="teste.css" type="text/css" rel="stylesheet"> </head> <body> CSS externo </body> </html>	<!DOCTYPE html> <html> <title>Exemplo CSS</title> <head> <meta charset="UTF-8"> <link href="teste.css" type="text/css" rel="stylesheet"> <style> p{color:blue;} </style> </head> <body> CSS incorporado </body> </html>	<!DOCTYPE html> <html> <title>Exemplo CSS</title> <head> <meta charset="UTF-8"> <link href="teste.css" type="text/css" rel="stylesheet"> <style> p{color:blue;} </style> </head> <body> CSS inline </body> </html>
Resultado na tela		
		

3.6 Uso de Cores

Você já sabe como incluir o CSS. Vamos agora identificar como trabalhar com cores no CSS. Existem várias formas de especificar uma mesma cor. Podemos formatar as cores de um determinado elemento HTML, usando a propriedade color e valor da cor. Atualmente os navegadores suportam 140 cores diferentes. No CSS há diferentes formas de identificar uma cor:

- **pelo nome da cor:** existem 140 diferentes nomes pré-definidos de cores. Você pode encontrar todos os nomes padrão de cores no seguinte endereço: https://www.w3.org/TR/css-color-3/;
- **pelos valores em RGB:** todas as cores podem ser representadas pela combinação das cores vermelho (Red em inglês), verde (green em inglês) e azul (Blue em inglês). Para cada uma dessas cores, pode-se atribuir valores de 0 a 255;
- **valores em RGBA:** também representa as cores usando Red, Green, Blue e acrescentamos o Alfa, que indica a intensidade da transparência. O valor Alfa pode variar de 0 a 1, quando 0 é uma cor opaca e 1 é a total transparência;
- **valores em Hexadecimal:** os valores de 0 a 255 do RGB podem ser convertidos para o padrão hexadecimal. Assim, você coloca o símbolo # e os valores em hexadecimal representando os valores do padrão RGB;

- **valores em HSL:** podemos definir uma cor em termos da tonalidade (Hue), saturação (Saturation) e leveza (Lightness):
    - saturação é um valor percentual que representa a pureza ou intensidade de uma cor, variando de 0% a 100%.
    - a leveza é um valor percentual que representa a quantidade de luz percebida, variando de 0% a 100%.
- **valores em HSLA:** também representa as cores usando o padrão HSL e acrescentamos o Alfa, que indica a intensidade da transparência. O valor Alfa pode variar de 0 a 1, quando 0 é uma cor opaca e 1 seria a total transparência.

O Hue é uma roda de cores, em que cada ângulo indica uma cor pura diferente. Por exemplo, 0 é vermelho, 120 é verde e 240 é azul.

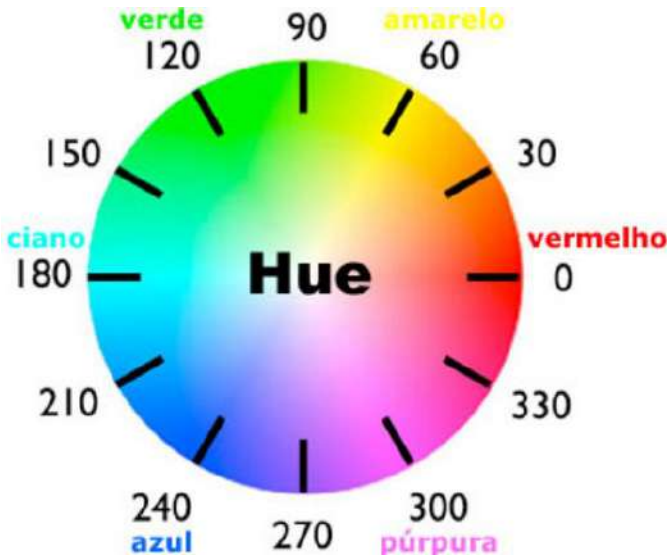


Imagem 2: HUE - roda de cores
Fonte:
https://www.maujor.com/tutorial/css3-modulo-para-cores.php

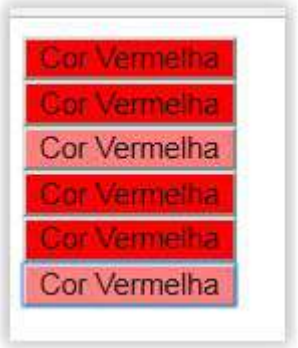
Código HTML

```
<!DOCTYPE html>
<html>
   <title>Exemplo CSS</title>
   <head>
      <meta charset="UTF-8">
      <link href="teste.css"
          type="text/css"
          rel="stylesheet">
   </head>
   <body>
      <button id="testeNome">Cor Vermelha</button><br>
      <button id="testeRGB">Cor Vermelha</button><br>
      <button id="testeRGBA">Cor Vermelha</button><br>
      <button id="testeHex">Cor Vermelha</button><br>
      <button id="testeHSL">Cor Vermelha</button><br>
      <button id="testeHSLA">Cor Vermelha</button><br>
   </body>
</html>
```

Código CSS

```
#testeNome{ background-color:red; }
#testeRGB{ background-color:rgb(255,0,0); }
#testeRGBA{ background-color:rgba(255,0,0,0.5); }
#testeHex{ background-color:#ff0000; }
#testeHSL{ background-color:hsl(0,100%,50%); }
#testeHSLA{ background-color:hsla(0,100%,50%,0.5); }
```

Resultado na tela



Saiba mais!

Eventualmente você vai querer converter uma determinada cor para padrões diferentes. O cálculo manual dessa conversão pode ser mais complicado. Entretanto, existem diversos sites que fazem a conversão online, um deles é o webcalc.

Disponível em: http://webcalc.com.br/Utilitarios/form/rgb_hex.

3.7 Alterando os Padrões de Fundo

Conforme vimos no exemplo acima, podemos definir diferentes padrões de fundo de um elemento do HTML, que pode ser uma cor específica ou uma imagem. Usando o CSS, você consegue formatar muito melhor os padrões de fundo, tanto cor quanto imagem de fundo.

3.7.1 Usando uma cor de fundo

Para definir a cor de fundo de um elemento do HTML, podemos utilizar a propriedade background-color com uma das especificações de cores vistas no item anterior.

Sintaxe básica:

seletor {background-color: cor;}

3.7.2 Usando uma imagem de fundo

E, se ao invés de uma cor, você quisesse colocar uma imagem de fundo? Nesse caso, você precisaria usar estas propriedades:

- **background-image**

Com esta propriedade, você pode especificar a imagem que será usada como fundo do elemento do HTML. Por definição, a imagem será repetida ao longo de todo o elemento.

Exemplo:

Código HTML

```
<!DOCTYPE html>
<html>
   <title>Exemplo CSS</title>
   <head>
      <meta charset="UTF-8">
      <link href="teste.css"
          type="text/css"
          rel="stylesheet">
   </head>
   <body>   </body>
</html>
```

Código CSS

```
body{
    background-image: url("imagens/tbbt.jpg");
}
```

Resultado na tela

Observe que você precisa indicar onde a imagem se encontra. Se estiver em uma pasta diferente de onde você salvou o código HTML, deverá indicar o caminho até a imagem.

Você também pode indicar o endereço do site onde a imagem pode ser localizada. Por exemplo:

```
body{
    background-image:
    url(“https://upload.wikimedia.org/wikipedia/lt/thumb/1/1b/
    BigBangTheoryTitleCard.png/250px-BigBangTheoryTitleCard.png”);
    }
```

- **background-repeat**

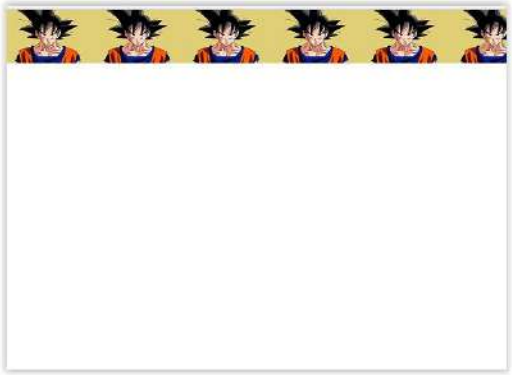
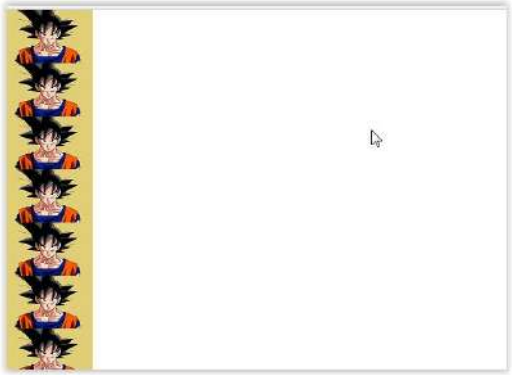
E se você quiser repetir a imagem apenas na vertical ou na horizontal?

Neste caso, você deve utilizar a propriedade background-repeat, junto com a propriedade background-image. Os principais valores permitidos para essa nova propriedade são:

- repeat-x: a imagem de fundo é repetida apenas horizontalmente;
- repeat-y: a imagem de fundo é repetida apenas verticalmente;
- no-repeat: a imagem de fundo não será repetida;
- space: a imagem de fundo é repetida o máximo possível sem recortar.

Código HTML	Código CSS

body{
background-image:
url("imagens/goku_menor.jpg");
background-repeat: repeat-x;
}
	body{
	background-image:
	url("imagens/goku_menor.jpg");
	background-repeat: repeat-y;

Resultado na tela	
	

- **background-attachment**

Usando esta propriedade, a imagem de fundo não rola com a página, fica fixa.

Código HTML

```
<!DOCTYPE html>
<html>
  <title>Exemplo CSS</title>
  <head>
    <meta charset="UTF-8">
    <link href="teste.css"
        type="text/css"
        rel="stylesheet">
  </head>
  <body>
    Todos os dias quando acordo<br>
    Não tenho mais<br>
    O tempo que passou<br>
    Mas tenho muito tempo<br>
    Temos todo o tempo do mundo<p>
    Todos os dias<br>
    Antes de dormir<br>
    Lembro e esqueço<br>
    Como foi o dia<br>
    Sempre em frente<br>
    Não temos tempo a perder<p>
    Nosso suor sagrado<br>
    É bem mais belo<br>
    Que esse sangue amargo<br>
    E tão sério<br>
    E selvagem! Selvagem!<br>
    Selvagem!<p>
```

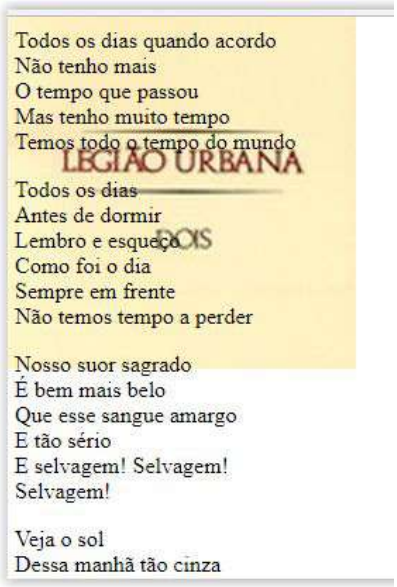
(continua...)

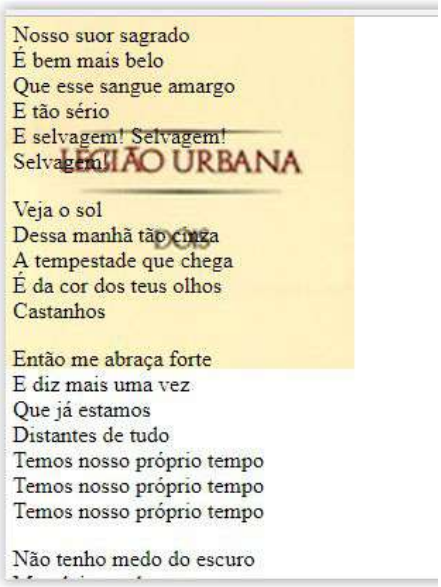
Código CSS

```
body{
   background-image:
        url("legiao.jpg");
   background-repeat: no-repeat;
   background-attachment: fixed;
}
```

Código HTML	Código CSS
Veja o sol Dessa manhã tão cinza A tempestade que chega É da cor dos teus olhos Castanhos<p> Então me abraça forte E diz mais uma vez Que já estamos Distantes de tudo Temos nosso próprio tempo Temos nosso próprio tempo Temos nosso próprio tempo<p> </body> </html>	

Resultados na tela (mostra que a imagem fixa, mesmo rolando o texto)





- **background-position**

Por padrão, uma imagem em segundo plano é colocada no canto superior esquerdo da tela. E se você não quiser a imagem nesse local?

A propriedade background-position define outra posição inicial de uma imagem de fundo, utilizando os valores: left top, left center, left bottom, right top, right center, right bottom, center top, center center e center bottom.

Código HTML

```
<!DOCTYPE html>
<html>
  <title>Exemplo CSS</title>
  <head>
    <meta charset="UTF-8">
    <link href="teste.css"
        type="text/css"
        rel="stylesheet">
  </head>
  <body>
    Veja o sol<br>
    Dessa manhã tão cinza<br>
    A tempestade que chega<br>
    É da cor dos teus olhos<br>
    Castanhos<p>
    Então me abraça forte<br>
    E diz mais uma vez<br>
    Que já estamos<br>
    Distantes de tudo<br>
    Temos nosso próprio tempo<br>
    Temos nosso próprio tempo<br>
    Temos nosso próprio tempo<p>
  </body>
</html>
```

Código CSS

```
body{
   background-image:
        url("imagens/legiao.jpg");
   background-repeat: no-repeat;
   background-attachment: fixed;
   background-position:center top;
}
html{
   background-color:gray;
   color:white;
}
```

Resultados na tela

Saiba mais!

"Em se tratando de planos de fundo, as CSS nos fornecem a propriedade background, recurso que traz diferentes opções para customização desse espaço nos elementos declarados em nossas páginas.

Com background podemos definir a cor de fundo, imagem, o posicionamento dela, se deve se repetir, entre outras opções, agregando um diferencial ainda maior para o visual das nossas aplicações". Saiba mais neste vídeo da DevMedia.

Fonte: https://www.devmedia.com.br/css-background/3831

3.8 Textos e Fontes

No item anterior, você compreendeu como formatar padrões de fundo. Agora vamos aprender a formatar textos e fontes usando as propriedades do CSS.

As propriedades que formatam o texto e as fontes são as seguintes:

- **color:** define a cor de um texto, utilizando os valores já vistos anteriormente;
- **text-align:** define o alinhamento do texto na horizontal, utilizando os seguintes valores:
    - o left: alinhado à esquerda;
    - right: alinhado à direita;
    - center: alinhamento centralizado;
    - justify: alinhamento justificado.
- **text-decoration:** define o uso de linhas de decoração em um texto, que pode ser:
    - o overline: acima do texto;
    - underline: abaixo do texto;
    - line-through: em cima do texto.
- **text-transform:** define o uso de letras em caixa alta, que pode conter os seguintes valores:
    - o uppercase: todas as letras em caixa alta;
    - lowercase: todas as letras em caixa baixa;
    - capitalize: apenas a primeira letra em caixa alta.

- **letter-spacing:** define o espaçamento (em pixels) entre os caracteres;
- **line-height:** define o espaçamento (em pixels) entre as linhas;
- **word-spacing:** define o espaçamento (em pixels) entre as palavras;
- **text-shadow:** insere uma sombra ao texto. Você deve especificar: a posição da sombra no sentido horizontal (em pixels);
    - a posição da sombra no sentido vertical (em pixels); e
    - a cor da sombra.
- **font-family:** define a família de fontes a ser utilizada. Se o navegador não suportar a primeira, ele tenta a próxima, e assim por diante.
- **font-style:** define se o texto será mostrado em itálico ou não, usando o valor italic;
- **font-size:** define o tamanho da fonte, que pode ser especificado em termos:
    - Absolutos: define a fonte com um tamanho específico (em pixels ou cm);
    - Relativos: define a fonte com um tamanho relativo, usando:
        - a unidade de medida em “em”, em que 1 em é o tamanho padrão;
        - xx-small: define um tamanho muito muito menor que o padrão;
        - x-small: define um tamanho muito menor que o padrão;
        - small: define um tamanho menor que o padrão;
        - large: define um tamanho maior que o padrão;
        - x-large: define um tamanho muito maior que o padrão;
        - xx-large: define um tamanho muito muito maior que o padrão;
        - smaller: define um tamanho menor que o padrão;
        - larger: define um tamanho maior que o padrão
        - %: define um valor percentual em relação ao tamanho padrão.

- **font-weight:** define o peso da fonte, ou seja, a intensidade do negrito. Pode utilizar um destes valores:
    - bold: define o negrito padrão;
    - bolder: define um negrito mais intenso;
    - lighter: define um destaque usando um negrito menos intenso que o padrão;
    - valores de 100 a 900.

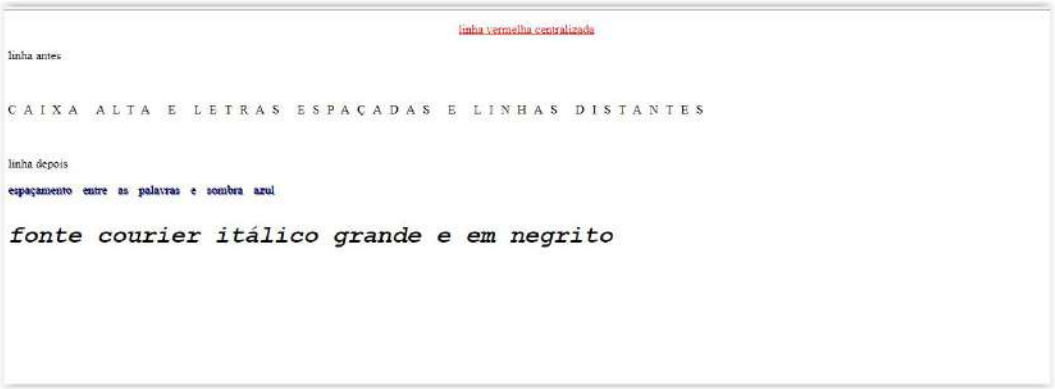
Código HTML

```
<!DOCTYPE html>
<html>
  <title>Exemplo CSS</title>
  <head>
    <meta charset="UTF-8">
    <link href="teste.css"
        type="text/css"
        rel="stylesheet">
  </head>
  <body>
    <p class="formato1">
        linha vermelha centralizada
    </p>
    linha antes
    <p class="formato2">
        caixa alta e letras espaçadas e
        linhas distantes
    </p>
    linha depois
    <p class="formato3">
        espaçamento entre as
        palavras e sombra azul
    </p>
    <p class="formato4">
        fonte courier itálico grande e
        em negrito
     </p>
  </body>
</html>
```

Código CSS

```
p.formato1{
   color:red;
   text-align:center;
   text-decoration:underline;
}
p.formato2{
   text-transform:uppercase;
   letter-spacing:10px;
   line-height:90px;
}
p.formato3{
   word-spacing: 10px;
   text-shadow:1px 1px blue;
}

p.formato4{
   font-family:courier,"Times New Roman";
   font-style:italic;
   font-size:xx-large;
   font-weight:900;
}
```

Resultados na tela



Saiba mais!

A MDN, uma rede de desenvolvedores da Mozilla, é uma plataforma de aprendizagem sobre as principais tecnologias da Web, que disponibiliza um tutorial CSS, do qual destacamos a página referente aos estilos de texto. Disponível em:

https://developer.mozilla.org/pt-PT/docs/Web/CSS/Como_come%C3%A7ar/Estilos_de_texto

3.9 Animação com CSS

Vamos refletir?

Já imaginou fazer uma animação bem legal, fazendo a transição de estilo CSS para outro? Por exemplo, fazendo a transição suave de cores de fundo de um elemento HTML qualquer. É muito simples. Vamos aprender?

Para a fazer uma animação, o mínimo que você precisa fazer é o seguinte:

- Definir regras dentro de um @keyframe: essas regras especificam os percentuais de mudança de estilo. Para que uma animação funcione, você deve vincular a animação a um elemento.
- Definir os critérios da animação usando:
    - animation-name: especifica um nome para a animação @keyframes;
    - animation-duration: define quantos segundos ou milissegundos uma animação leva para completar um ciclo;
    - animation-delay: especifica um atraso em segundos ou milissegundos para o início de uma animação;
    - animation-direction: define se uma animação deve tocar no sentido inverso ou em ciclos alternados. Use alternate ou reverse;
    - animation-iteration-count: especifica o número de vezes que uma animação deve ser reproduzida. Use um número específico de vezes ou infinite.

3.9.1 Transformações

Junto com as animações você também pode fazer transformações 2D, usando a propriedade transform e os seguintes valores:

- translate(): que move um elemento HTML da sua posição atual (de acordo com os parâmetros dados para o eixo X e o eixo Y). Exemplo: translate(10px,10px);
- rotate(): que gira um elemento HTML no sentido horário ou anti-horário de acordo com um determinado grau. Exemplo: rotate(10deg);
- scale(): que aumenta ou diminui a escala de um um elemento HTML. Exemplo: scale(1.5,1.5).

Código HTML	Código CSS

<!DOCTYPE html>
<html>
<title>Exemplo CSS</title>
<head>
<meta charset="UTF-8">
<link href="teste.css"
type="text/css"
rel="stylesheet">
</head>
<body>
<button></button>
</body>
</html>
	@keyframes aula {
	0% {transform: translate(0px,0px);}
	20% {transform: translate(200px,0px);}
	40% {transform: translate(400px,0px);}
	60% {transform: translate(600px,0px);}
	80% {transform: translate(800px,0px);}
	100% {transform: translate(1000px,0px);}
	}
	button {
	background-color: red;
	animation-name: aula;
	animation-duration: 2s;
	animation-iteration-count:infinite;
	animation-direction:alternate;
	width: 100px;
	height: 100px;
	}

Resultado na tela

Questões de autoaprendizagem

Vamos pensar e praticar...

Vamos voltar àquele exercício de HTML, no qual você criou uma tabela com seu desenho, jogos e Youtuber favoritos. Tenho certeza que você conseguiu fazer. Afinal, você é uma pessoa muito inteligente!!!

Agora, vamos formatar aquela página usando o CSS. Você deve alterar, no mínimo:

- cores de fundo;
- tamanho e negrito das fontes;
- expandir a largura da tabela para 100%;
- inserir animações diferentes para as 3 imagens.

Novamente, seja criativo(a), faça algo legal e mostre para seus/ suas amigos(as). Tenho certeza de que você consegue. Atenção: as imagens são apenas ilustrativas.

Coisas que eu gosto	
Meu desenho favorito	
Meu game preferido	
Meu youtuber preferido	

Coisas que eu gosto	
Meu desenho favorito	
Meu game preferido	
Meu youtuber preferido	

Solução...

Para resolver essa questão, você deveria pensar no padrão de cores que iria usar, alterar as fontes e imaginar as animações de translação, mudança de escala e rotação. Por fim, deveria escrever um código semelhante a este:

Arquivo HTML

```
<html>
  <head>
    <title>Exemplo</title>
    <meta charset="UTF-8">
    <link rel="stylesheet"
        href="exemplo.css
        "
        type="text/css">
  </head>
  <body>
    <table border>
      <tr>
        <th colspan="2">Coisas que eu gosto</th>
      </tr>
      <tr>
        <td>Meu desenho favorito</td>
        <td>
          <img
            id="deslizaDesenho"
            width="200"
            height="200"
            src="https://i.ytimg.com/vi/5_f5Nmt1pRE/maxresdefault.jpg">
        </td>
      </tr>
```

```
        <tr>
          <td>Meu game preferido</td>
          <td>
            <img
                id=”encolheJogo”
                width=”200”
                height=”200”
            src=”http://cdn.edgecast.steamstatic.com/steam/apps/730/header.jpg?t=1513742714”
            </td>
        </tr>
        <tr>
          <td>Meu youtuber preferido</td>
          <td>
            <img
                id=”giraYoutuber”
                width=”200”
                height=”200”
                src=”http://s2.glbimg.com/8BrYj3OetdqHBzbUammebQ2o4VA=/620x465/s.glbimg.com/jo/g1/f/original/2016/06/27/whindersson1.png”
            </td>
        </tr>
        <tr>
        </tr>
      </table>
   </body>
</html>
```

Arquivo CSS

```
table{
    width:100%;
}
th{
    background-color:rgb(12,97,18)
    ;  color:white;
    text-align:center;
    text-transform:uppercase
    ;  font-size:xx-large;
}
td{
    background-color:rgb(187,247,191)
    ;  text-align:center;
    text-shadow: 1px 1px
    green;  font-weight: 600;
    font-size:30px;
}
@keyframes animaYoutuber {
    8% {transform: rotate(30deg);}
    17% {transform: rotate(60deg);}
    25% {transform: rotate(90deg);}
    33% {transform: rotate(60deg);}
    42% {transform: rotate(30deg);}
    50% {transform: rotate(0deg);}
```

```
    58% {transform: rotate(-30deg);}
    66% {transform: rotate(-60deg);}
    75% {transform: rotate(-90deg);}
    83% {transform: rotate(-60deg);}
    91% {transform: rotate(-30deg);}
    100% {transform: rotate(0deg);}
}
@keyframes animaJogo {
    13% {transform: scale(0.9,0.9);}
    25% {transform: scale(0.7,0.7);}
    38% {transform: scale(0.5,0.5);}
    50% {transform: scale(0.3,0.3);}
    63% {transform: scale(0.5,0.5);}
    75% {transform: scale(0.7,0.7);}
    88% {transform: scale(0.9,0.9);}
    100% {transform: scale(1.0,1.0);}
}
@keyframes animaDesenho {
    13% {transform: translate(30px,0px);}
    25% {transform: translate(60px,0px);}
    38% {transform: translate(30px,0px);}
    50% {transform: translate(0px,0px);}
    63% {transform: translate(-30px,0px);}
    75% {transform: translate(-60px,0px);}
    88% {transform:
            translate(-3
            0px,0px);}
    100% {transform: translate(0px,0px);}
}
```

```
#giraYoutuber{
    animation-name:
    animaYoutuber;
    animation-duration: 3s;
    animation-iteration-count:infinite;
}
#encolheJogo{
    animation-name:
    animaJogo;
    animation-duration: 3s;
    animation-iteration-count:infinite;
}
#deslizaDesenho{
    animation-name:
    animaDesenho;
    animation-duration: 3s;
    animation-iteration-count:infinite;
}
```

Vamos rever?

Bom, chegamos ao final de mais uma unidade! Espero que você tenha gostado. Você, uma pessoa muito inteligente e divertida, conseguiu compreender os principais conceitos sobre folhas de estilo, conheceu os principais editores, aprendeu a usar classes e id's, descobriu diversas maneiras de incluir as formatações do CSS, aprendeu as diversas formas de uso de cores, conheceu as técnicas para alterar os padrões de fundo, entendeu como aplicar formatações a textos e fontes, e aprendeu a fazer animações simples usando o CSS.

Sites indicados

- CSS Tutorial. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/CSS Acesso: 15 jan. 2018.

- CSS. Disponível em: https://www.w3schools.com/css/default.asp Acesso: 15 jan. 2018.

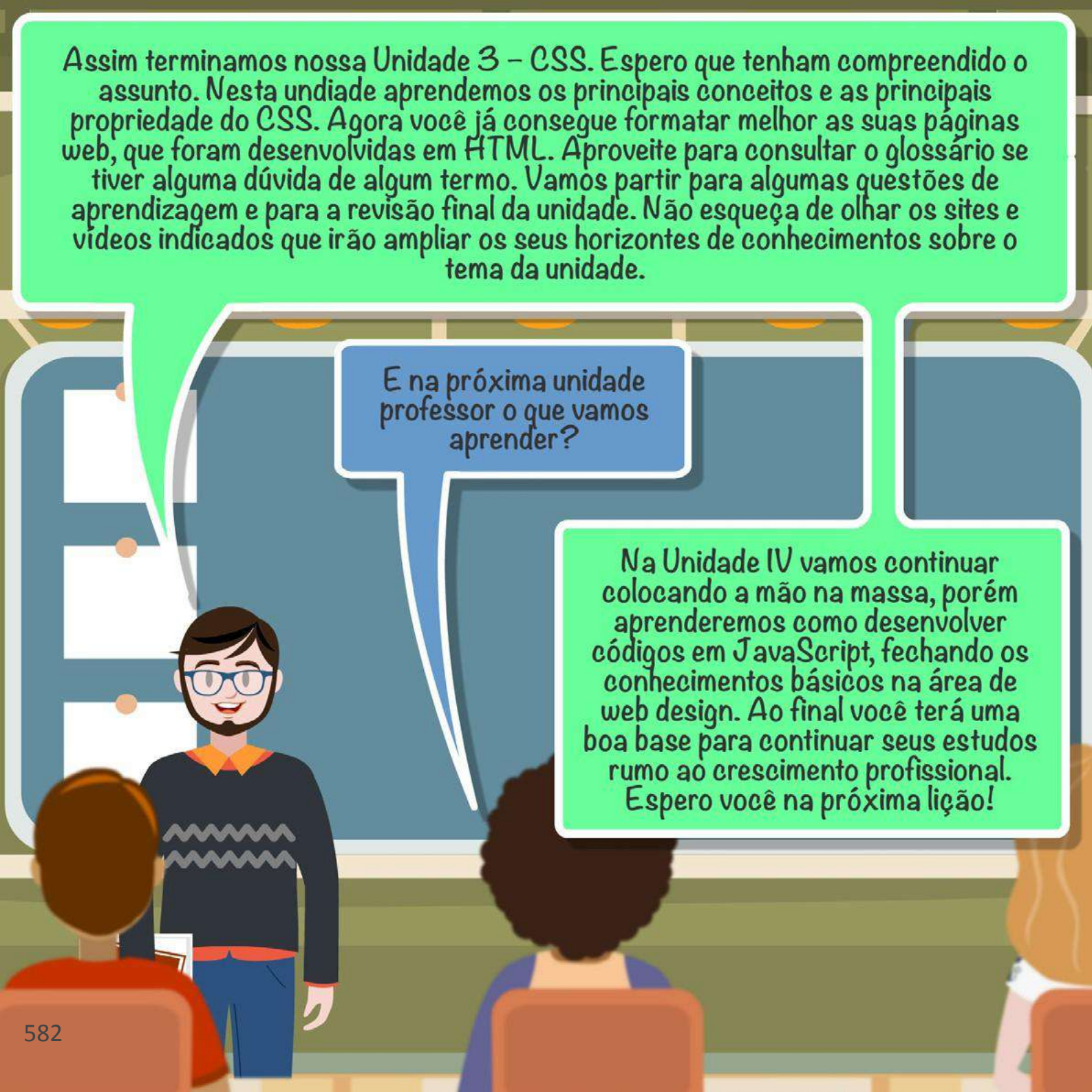
Glossário

- Background
  É um substantivo masculino que significa fundo ou segundo plano.
  Fonte: https://www.dicio.com.br/background/

- CSS
  CSS (Cascading Style Sheets ou em português: folhas de estilos em cascata) é uma linguagem de estilo usada para descrever a apresentação de um documento escrito em HTML ou em XML (incluindo várias linguagens em XML como SVG ou XHTML). O CSS descreve como elementos são mostrados na tela, no papel, no discurso ou em outras mídias.
  Fonte: https://developer.mozilla.org/pt-BR/docs/Web/CSS

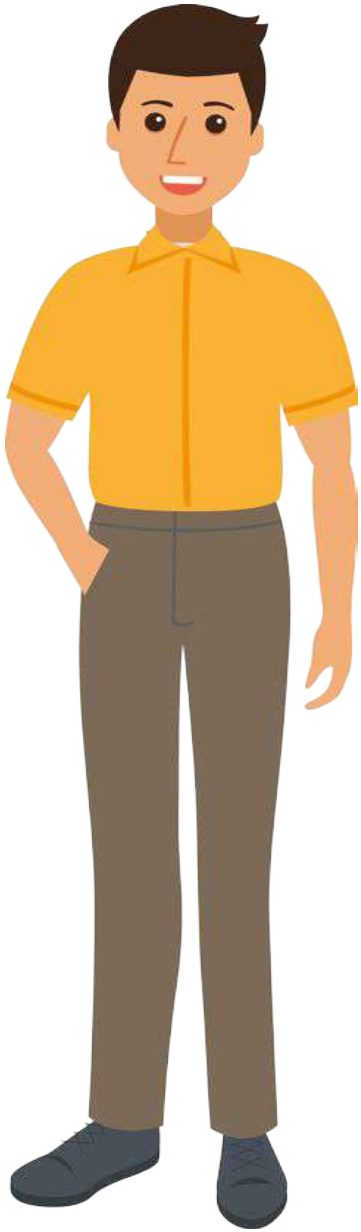
- Padding
  É uma propriedades CSS usada para gerar espaço em torno do conteúdo de um elemento, dentro de qualquer limite definido.
  Fonte: https://www.w3schools.com/css/css_padding.asp

- Template
  É um modelo a ser seguido, com uma estrutura predefinida que facilita o desenvolvimento e a criação do conteúdo a partir de algo construído a priori. Ao instalar o Joomla! em sua versão mais básica, alguns modelos já são trazidos e podem ser usados e modificados livremente, mas dependendo do que se deseja construir ou modificar, existem outros modelos que melhor se adaptam à nossa necessidade.
  Fonte: https://www.portaleducacao.com.br/conteudo/artigos/educacao/o-que-e-template/39828


Assim terminamos nossa Unidade 3 – CSS. Espero que tenham compreendido o assunto. Nesta undiade aprendemos os principais conceitos e as principais propriedade do CSS. Agora você já consegue formatar melhor as suas páginas web, que foram desenvolvidas em HTML. Aproveite para consultar o glossário se tiver alguma dúvida de algum termo. Vamos partir para algumas questões de aprendizagem e para a revisão final da unidade. Não esqueça de olhar os sites e videos indicados que irão ampliar os seus horizontes de conhecimentos sobre o tema da unidade.
E na próxima unidade professor o que vamos aprender?
Na Unidade IV vamos continuar colocando a mão na massa, porém aprenderemos como desenvolver códigos em JavaScript, fechando os conhecimentos básicos na área de web design. Ao final você terá uma boa base para continuar seus estudos rumo ao crescimento profissional. Espero você na próxima lição!

Unidade 4 JavaScript

Olá, que bom encontrar você novamente!!! Vamos avançar mais um pouco nos fundamentos de web design? Na Unidade 3, você descobriu como formatar uma página HTML usando as propriedades do CSS. Agora, você aprenderá a utilizar o JavaScript, uma linguagem extremamente versátil e essencial para quem trabalha com desenvolvimento web.



Por exemplo, você já preencheu um formulário na web e quando deixou um campo em branco, recebeu um aviso para preencher? Então, isso é feito com o JavaScript, que é uma linguagem de programação codificada no interior de página HTML ou em um arquivo separado.

Ao final desta unidade, você conhecerá os principais conceitos sobre o JavaScript, conhecerá os principais editores, aprenderá as diversas maneiras de incluir a linguagem JavaScript em sua página web, conhecerá várias maneiras de mostrar uma informação na tela usando essa linguagem, saberá quais são os elementos básicos de programação, aprenderá a criar funções e eventos com o JavaScript.

Você sabia?

Muita gente acha que JavaScript é uma espécie de versão da Linguagem Java, mas não é. A linguagem de programação JavaScript, desenvolvida pela Netscape, não faz parte da plataforma Java. O JavaScript não cria applets ou aplicações independentes. Na sua forma mais comum, o JavaScript fica embutido nos documentos HTML e pode fornecer níveis de interatividade para páginas Web que não são acessíveis com um HTML simples.

Diferenças-chave entre o Java e o JavaScript:

- Java é uma linguagem de programação Orientada a Objeto, ao passo que Java Script é uma linguagem de scripts Orientada a Objeto.
- Java cria aplicações executadas em uma máquina virtual ou em um browser, ao passo que o código JavaScript é executado apenas em um browser.
- O código Java precisa ser compilado, ao passo que os códigos JavaScript estão totalmente em texto.
- Eles requerem plug-ins diferentes.

Fonte: https://www.java.com/pt_BR/download/faq/java_javascript.xml

4.1 Editores

Desenvolvedores profissionais utilizam Ambientes de Desenvolvimento Integrado (em inglês, Integrated Development Environment - IDE), que devem oferecer um conjunto de ferramentas que facilitam o trabalho do desenvolvedor, incluindo um bom editor de texto, sintaxes destacadas, funcionalidades de auto completar e teclas de atalho.

Sabemos que existem diversos IDE's, então por onde você deve começar? Afinal, qual aplicativo você precisa usar para criar um código em JavaScript? Veremos os principais ambientes logo a seguir.

A rigor, basta ter um editor de texto simples, como o bloco de notas, para criar um código CSS (neste caso, você deve salvar como um arquivo .css) e um navegador para visualizar o resultado do referido código. Existem dezenas de ambiente de desenvolvimento. Vou apresentar para você os principais.

Você precisa de algum aplicativo para criar um código em JavaScript?

Semelhante ao que dissemos na unidade sobre HTML, para criar um código em JavaScipt basta ter um editor de texto simples (bloco de notas), salvar como um arquivo .js e um navegador para visualizar o resultado do referido código. Entretanto, existem dezenas de ambientes de desenvolvimento, os mesmos do HTML, vamos relembrar na tabela a seguir.

Nome	Breve descrição	Onde encontrar?
Adobe Dreamweaver	Famoso programa para desenvolvimento de páginas web.	http://www.adobe.com/br/products/dreamweaver.html
Notepad++	Editor de código-fonte gratuito, regido pela Licença GPL[1] .	https://notepad-plus-plus.org/
Sublime Text	Semelhante ao Notepad++, porém com uma interface melhorada.	https://www.sublimetext.com/
Brackets	Um editor de códigos leve e limpo com ótimas funções de apoio.	http://brackets.io/
NetBeans	Ambiente de desenvolvimento integrado gratuito e de código aberto para desenvolvedores de software de várias linguagens.	https://netbeans.org/downloads/

Tabela 1: ambientes de desenvolvimento CSS
Fonte: elaborado pelo autor

[1] GNU General Public License (Licença Pública Geral GNU), GNU GPL ou simplesmente GPL.
Fonte: http://www.gnu.org/

Exemplo:

Peço que você teste todos os exemplos que mostraremos. Para isso, faça o seguinte:

1. Abra o aplicativo chamado “bloco de notas”;
2. Escreva o código a seguir;
3. Salve o arquivo com o nome exemplo.html;
4. Dê dois cliques em cima do arquivo exemplo.html;
5. O navegador padrão do seu computador será aberto e você verá o resultado.

Você sabia?

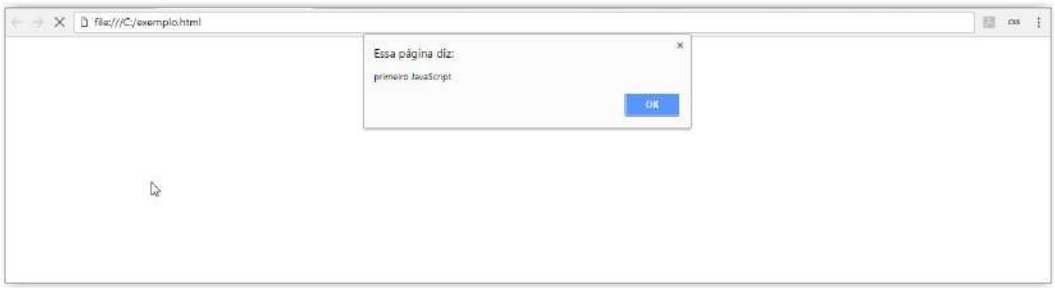
O código em JavaScript pode ser escrito tanto no interior de um código HTML, quanto em um arquivo externo. Veremos como fazer isso daqui a pouco!

O código em JavaScript pode ser colocado dentro do código HTML, entre as seções <body> ou <head>, utilizando a marcação <script>, usando o parâmetro type, com o valor text/javascript.

Código HTML com JavaScript

```
<html>
  <body>
    <script  type="text/javascript">
     alert("primeiro JavaScript");
    </script>
  </body>
</html>
```

Resultado no navegador



4.2 O JavaScript em um Arquivo Externo

Vamos refletir?

Lembra que a gente usou o elemento <link> do HTML para referenciar um arquivo externo do CSS. Então, com o JavaScript é bem semelhante.

Assim a gente pode utilizar o mesmo código em JavaScript em várias páginas HTML, de maneira rápida, fácil e eficiente, facilitando também a manutenção do referido código.

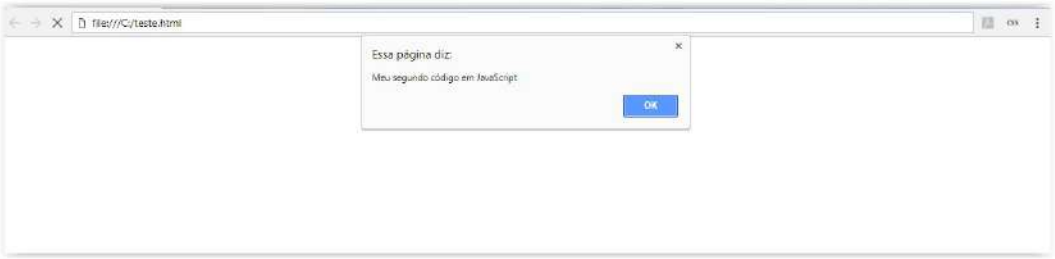
Escreva seu código JavaScript, salve em um arquivo com a extensão .js com todas as instruções do JavaScript. No código HTML, chame o referido arquivo .js da seguinte forma:

- Use a marcação script, a propriedade type e o valor text/javascript.
- Na mesma marcação script, use também a propriedade src, e como valor da mesma, use o nome do arquivo junto com a extensão.

Exemplo:

Código HTML	Código JavaScript Externo
<!DOCTYPE html> <html> <title>Título da Página</title> <head> <script charset="ISO-8859-1" type="text/javascript" src="exemplo.js"> </script> </head> <body> </body> </html>	alert("Meu segundo código em JavaScript");

Resultado no navegador



Você sabia?

O atributo do charset com o valor “ISO-8859-1” é usado para indicar que formatação utilizará a codificação de caracteres do alfabeto latino.

4.3 Como Mostrar Informações na Tela?

No JavaScript, assim como em outras linguagens, temos uma coisa chamada função, que é basicamente um bloco de código que realiza uma tarefa específica. Esse bloco facilita a manutenção do código, mas só é executado quando é chamado.

Em outro tópico, falaremos como criar as suas próprias funções. Neste momento utilizaremos funções pré-existentes, mais especificamente funções que mostram informações na tela do navegador.

4.3.1 Usando a função alert()

Você já deve ter observado que utilizamos essa função no exemplo anterior para abrir uma janela e mostrar uma mensagem de alerta. Adicionalmente, você pode concatenar um texto com um valor numérico usando o símbolo +.

Código HTML	Código JavaScript Externo
<html> <title>Título da Página</title> <head> <script charset="ISO-8859-1" type="text/javascript" src="exemplo.js"> </script> </head> <body> </body> </html>	alert("Numero = "+5);

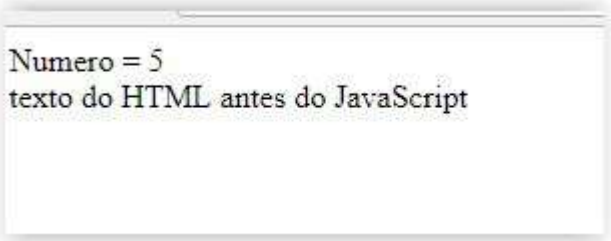
Resultado no navegador

4.3.2 Usando a document.write()

Funciona de forma semelhante ao alert, mas em vez de apresentar o texto em uma janela, apresenta na própria página web.

Código HTML	Código JavaScript Externo
<!DOCTYPE html> <html> <title>Título da Página</title> <head> <script charset="ISO-8859-1" type="text/javascript" src="exemplo.js"> </script> </head> <body> texto do HTML antes do JavaScript </body> </html>	document.write("Numero = "+5);

Resultado no navegador



4.3.3 Usando a função getElementById() com o innerHTML

Você lembra que a gente estudou o uso do id unidade III, quando falamos sobre o CSS? Foi dito que id é uma forma de identificar um elemento do HTML. Dissemos também que o id deve ser único para cada elemento.

E se você quisesse mostrar alguma informação em um elemento específico do HTML usando esse id? Nesse caso, você poderia utilizar o método getElementById() do objeto document, junto com a propriedade innerHTML. Essa propriedade possibilita a troca do conteúdo de um determinado elemento, sem a necessidade de recarregar página web.

Vamos apresentar um exemplo a seguir e se você quiser ver mais detalhes acesse estes sites:

- document.getElementById(). Disponível em:
  https://developer.mozilla.org/pt-BR/docs/Web/API/Document/getElementById
  Acesso: 19 jan.2018.

- Element.innerHTML. Disponível em:
  https://developer.mozilla.org/pt-BR/docs/Web/API/Element/innerHTML
  Acesso: 19 jan.2018.

A sintaxe básica é a seguinte:

```
document.getElementById(id).innerHTML = valor
```

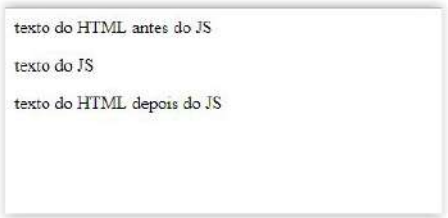
Onde:

- **getElementById():** é a função que retorna, por meio do id, a referência de um determinado elemento;
- **id:** é o identificador único que você utilizou em um elemento do HTML;
- **innerHTML:** é a propriedade que substitui o elemento do HTML;
- **valor:** é o conteúdo que será utilizado para substituir o conteúdo do elemento identificado pelo id.

Código HTML com JavaScript

```
<!DOCTYPE html>
<html>
  <title>Título da Página</title>
  <body>
    texto do HTML antes do JS
    <p id="testeJS"> </p>
    <script
       charset="ISO-8859-1"
       type="text/javascript">
      document.getElementById("testeJS").innerHTML = "texto do JS";
    </script>
    texto do HTML depois do JS
  </body>
</html>
```

Resultado no navegador



Você sabia?

Programação Orientada a Objetos é um paradigma de programação que usa abstração para criar modelos baseados no mundo real. E o método é uma ação do objeto, como ligar, desligar, frear se estivermos representando um veículo, por exemplo. É uma subrotina ou função associada a uma classe.

Fonte: https://developer.mozilla.org/pt-BR/docs/conflicting/Learn/JavaScript/Objects

4.4 Elementos Básicos de Programação em JavaScript

Agora a gente vai fazer uma rápida passagem por alguns conceitos fundamentais, que são utilizados em todas as linguagens de programação. Aproveite para fazer uma revisão ou aprender o básico.

4.4.1 Variáveis

São endereços de memória associados a um determinado nome (também chamado de identificadores), que guardam algum dado em tempo de execução. Você pode atribuir qualquer nome a uma variável, desde que siga as regras abaixo.

- Não utilize símbolos especiais. Por exemplo, ç ou acentos;
- O nome da variável deve começar sempre com uma letra. Por exemplo, usuario1;
- Recomenda-se que os nomes de variáveis iniciem sempre com letras minúsculas.

Observações:

- O JavaScript é case sensitive, ou seja, considera letras maiúsculas e minúsculas diferentes;
- Quando você criar uma variável, poderá utilizar ou a palavra var antes do nome da variável. Se usar, a referida variável será considerada uma variável local e não global.

Variável com letra minúscula

```
<!DOCTYPE html>
<html>
  <title>Título da Página</title>
  <body>
    <script
        charset="ISO-8859-1"
        type="text/javascript">
        var palavra = "Goku";
        var PALAVRA = "Jiren";
        alert(palavra);
    </script>
  </body>
</html>
```

Variável com letra maiúscula

```
<!DOCTYPE html>
<html>
  <title>Título da Página</title>
  <body>
    <script
        charset="ISO-8859-1"
        type="text/javascript">
        var palavra = "Goku";
        var PALAVRA = "Jiren";
        alert(PALAVRA);
    </script>
  </body>
</html>
```

Resultado no navegador

Você sabia?

A Rede Nacional de Ensino e Pesquisa (RNP) provê a integração global e a colaboração apoiada em Tecnologias de Informação e Comunicação para a geração do conhecimento e a excelência da educação e da pesquisa.

Fonte: https://www.rnp.br/sobre

A RNP disponibilizou uma série de videoaulas, entre as quais destaco a aula sobre "Tipos de Dados, Variáveis e Comandos de Entrada e Saída". Caso você tenha ainda alguma dúvida sobre estes assuntos básicos, recomendo que você assista a essa videoaula.

Disponível em: http://videoaula.rnp.br/v.php?f=/ufjf/ciencias_exatas/dcc119/aula1/dcc119_aula1.xml
Acesso: 19 jan. 2018.

4.4.2 Operadores Aritméticos

São aqueles que permitem realizar algumas operações matemáticas básicas. São eles:

Símbolo	Operação Aritmética	Exemplo	Valor armazenado em x (após as instruções ao lado)
+	Soma	var x = 3 + 2;	5
-	Subtração	var x = 3 - 2;	1
*	Multiplicação	var x = 3 * 2;	6
/	Divisão	var x = 3 / 2;	1.5
%	Resto da divisão	var x = 10 % 6;	4
++	Incremento	var x = 4; x++;	5
--	Descremento	var x = 4; x--;	3

Tabela 1: operadores matemáticos
Fonte: elaborado pelo autor

4.4.3 Uso do if e de operadores relacionais

Digamos que você entre com uma senha no Facebook. Se a senha estiver correta, abre a página do perfil. Caso contrário, mostra uma mensagem de erro. Como resolver isso em programação? Poderia usar um if.

A estrutura do if é utilizada para simular tomadas de decisão. Se um teste lógico for verdadeiro, executamos um bloco de instruções, caso contrário, executamos outro bloco de instruções, mas nunca os dois ao mesmo tempo.

Os testes são realizados utilizando operadores relacionais, a saber:

Símbolo	Descrição	Exemplo	Resposta do teste

==	igual	var x = 3;
		var y = 4;
		if(x == y)
			falso

!=	diferente	var x = 3;
		var y = 4;
		if(x != y)
			verdadeiro

>	maior	var x = 3;
		var y = 4;
		if(x > y)
			falso

Tabela 2: operadores relacionais
Fonte: elaborado pelo autor

Estrutura básica:

```
if (teste)
{
        instruções
else
{
instruções
}
```

Código HTML com JavaScript

```
<html>
  <head>
    <title>Goku</title>
  </head>
  <body>
    <p id="testeJS"></p>
    <script
      charset="ISO-8859-1"
      type="text/javascript">
      var idadeGoku = 40;
      if(idadeGoku == 44)
        document.getElementById("testeJS").innerHTML =
            "Goku tem 44 anos";
      else
        document.getElementById("testeJS").innerHTML =
            "mude o valor da variável idadeGoku para 44";
    </script>

  </body>
</html>
```

Resultado no navegador

4.4.4 Uso de laços de repetição

Digamos que você queira repetir seu nome 10 vezes na página do navegador. O que faria? Que código utilizaria? Nem pense em usar document.write() 10 vezes (risos).

Nesse caso o melhor seria usar um laço de repetição.

Laços de repetição são estruturas utilizadas para repetir uma ou mais instruções. As duas principais estruturas de repetição são while e for.

A estrutura básica desses laços são:

Usando While

ValorInicialDoContador
while(TesteDeParada){
instruções
AlteraçãoDoValorDoContador
}

Usando For

for(ValorInicialDoContador; TesteDeParada; AlteraçãoDoValorDoContador){
instruções
}

Exemplos:

Usando While

```
<html>
  <head>
    <title>Jogo</title>
  </head>
  <body>
    <script charset="ISO-8859-1"
          type="text/javascript">
      var jogo = "Counter-Strike";
      var contador = 1;
      while(contador <= 10){
        document.write(
        contador+
        ") "+
         jogo+
         "<br>");
        contador++;
      }
    </script>
  </body>
</html>
```

Variável com letra maiúscula

```
<html>
  <head>
    <title>Jogo</title>
  </head>
  <body>
    <script charset="ISO-8859-1"
          type="text/javascript">
      var jogo = "Counter-Strike";
      for(var contador = 1;
            contador <= 10;
            contador++){
        document.write(
        contador+
        ") "+
         jogo+
         "<br>");
      }
    </script>
  </body>
</html>
```

Resultado no navegador

1) Counter-Strike
2) Counter-Strike
3) Counter-Strike
4) Counter-Strike
5) Counter-Strike
6) Counter-Strike
7) Counter-Strike
8) Counter-Strike
9) Counter-Strike
10) Counter-Strike

1) Counter-Strike
2) Counter-Strike
3) Counter-Strike
4) Counter-Strike
5) Counter-Strike
6) Counter-Strike
7) Counter-Strike
8) Counter-Strike
9) Counter-Strike
10) Counter-Strike

Você sabia?

Para saber mais sobre laços de repetição assista aos vídeos da Khan Academy.

- Introdução aos laços de repetição While. Disponível em: https://pt.khanacademy.org/computing/computer-programming/programming/looping/pt/intro-to-while-loops
- Laços For! Um novo tipo de Estrutura de Repetição. Disponível em: https://pt.khanacademy.org/computing/computer-programming/programming/looping/pt/for-loops-a-new-kind-of-loop

Assista! Vale a pena.

4.5 Criando as suas Funções em JavaScript

Como dissemos anteriormente, uma função é basicamente um bloco de código que realiza uma tarefa específica. Para que esse bloco seja executado, a função precisa ser chamada.

Syntaxe básica:

```
function nomeDaFunção (parâmetros){
instruções
 return dado;
}
```

Onde:

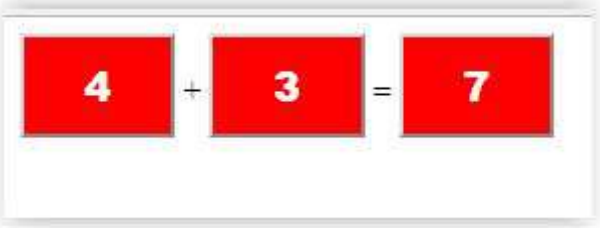
- **function** é a palavra-chave que define uma função;
- **nomeDaFunção** é nome que você dá para a função, que será usado quando você for chamá-la;
- **parâmetro** é a informação que a função precisa saber para ser executada. Por exemplo, a função que calcula a raiz quadrada precisa saber de qual número será calculada a raiz quadrada.
- **return** é opcional e serve para retornar alguma informação para quem chama a função.

Exemplo:

Código HTML com JavaScript e com CSS

```
<html>
  <head>
    <title>Funções</title>
    <style type="text/css">
      .botao {
        font-weight: 900;
        background-color:red;
        color: white;
        padding: 10px 30px;
        text-align: center;
        font-size: 20px;
      }
    </style>
  </head>
  <body>
    <button class="botao" id="valor1" width="50" height="50"></button> +
    <button class="botao" id="valor2" width="50" height="50"></button> =
    <button class="botao" id="resultado" width="50" height="50"></button>
    <script charset="ISO-8859-1"
        type="text/javascript">
      function soma(){
        document.getElementById("valor1").innerHTML = "4";
        document.getElementById("valor2").innerHTML = "3";
        document.getElementById("resultado").innerHTML = "7";
      }
      soma();
    </script>
  </body>
</html>
```

Resultado no navegador



4.6 Eventos em JavaScript

Normalmente um programa é executado sequencialmente. E se você quisesse que todas as vezes que passasse o mouse sobre uma figura, aparecesse uma mensagem? Como você faria? Use um evento. Existem dezenas de eventos, apresentamos os eventos mais utilizados, a saber:

Evento	Descrição
Onchange	Indica o que fazer quando um elemento do HTML mudar. Por exemplo, quando preencher o campo ou selecionar um item de um formulário.
Onclick	Indica o que fazer quando clicar sobre um elemento do HTML.
Onsubmit	Indica o que fazer quando clicar em botão submit de um formulário.
onmouseover	Indica o que fazer quando o mouse passar sobre um determinado elemento do HTML.
onload	Indica o que fazer quando o navegador termina de carregar uma página.
onunload	Indica o que fazer quando o navegador abandonar uma página HTML.

Tabela 3: eventos em JavaScript
Fonte: elaborado pelo autor

Exemplo:

Código HTML com JavaScript e com CSS

```
<html>
  <body>
    <h1>Clique sobre a imagem</h1>
    <img id="dbz"
      onclick="mudaImagem();"
      src="https://img00.deviantart.net/3cd7/i/2016/222/a/6/goku_by_bthrgking-dadbnvv.jpg"
      width="200"
      height="400">
    <script  type="text/javascript">
      function mudaImagem() {
        var image = document.getElementById('dbz');
        if (image.src.match("saiyan")) {
                                                                    image.src =
"https://img00.deviantart.net/3cd7/i/2016/222/a/6/goku_by_bthrgking-dadbnvv.jpg";
        } else {
                                                                    image.src =
"https://img00.deviantart.net/c429/i/2016/222/f/5/super_saiyan_3_goku_by_bthrgking-dadbo7l.jpg";
        }
      }
    </script>
  </body>
</html>
```

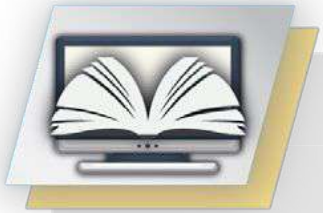
Resultado no navegador (imagem inicial)	Resultado no navegador (após o clique)
	

Você sabia?

Existem muitos eventos diferentes e listar e explicar todos eles não seria muito produtivo. Então, à medida da sua necessidade, você pode pesquisar qual evento é mais adequado para resolver um problema específico.

Para ajudar na sua pesquisa, indico um site onde consta uma lista mais completa de eventos em JavaScript.

- Desenvolvimento de Eventos. Disponível em: https://tdn.totvs.com/display/public/fluig/Desenvolvimento+de+Eventos Acesso: 19 jan. 2018.
- Referência do evento. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/Events Acesso: 19 jan. 2018.



Saiba mais!

Viu como eventos são uma ferramenta poderosa para o desenvolvedor web? Você conseguiu entender o conceito de evento? Caso reste alguma dúvida, proponho que você assista a mais uma videoaula da RNP.

No vídeo abaixo, é apresentado novamente o conceito de evento, os principais tipos e diz como associar trechos de código aos eventos e outros detalhes importantes.

Eventos. Disponível em:
http://videoaula.rnp.br/v.php?f=/cederj/sistemas_comp/ead05002/Aula_017/Aula_017.xml
Acesso: 19 jan. 2018.

Vamos rever?

Nesta lição a gente viu muita coisa sobre JavaScript. Vimos os principais editores para desenvolver seus projetos em JavaScript.

Aprendemos que podemos inserir o JavaScript dentro do próprio HTML ou podemos armazená-lo em um arquivo externo. Conhecemos algumas formas de mostrar uma informação na página do navegador. Fizemos uma rápida revisão sobre os principais elementos de programação. Aprendemos como criar funções e como criar eventos no JavaScript.

Sites indicados

JavaScript Tutorial. Disponível em: <https://www.w3schools.com/js/default.asp>. Acesso em: 19 jan. 2018.

Qual é a diferença entre o JavaScript e o Java?. Disponível em: <https://www.java.com/pt_BR/download/faq/java_javascript.xml>. Acesso em: 19 jan. 2018.

Introdução ao JavaScript Orientado a Objeto. Disponível em: <https://developer.mozilla.org/pt-BR/docs/conflicting/Learn/JavaScript/Objects>. Acesso em: 19 jan. 2018.

O que é uma variável?. Disponível em: <https://www.youtube.com/watch?v=-ZMCNZXmzZk>. Acesso em: 19 jan. 2018.

Introdução aos laços de repetição While. Disponível em: <https://pt.khanacademy.org/computing/computer-programming/programming/looping/pt/intro-to-while-loops>. Acesso em: 19 jan. 2018.

Laços For! Um novo tipo de Estrutura de Repetição. Disponível em: <https://pt.khanacademy.org/computing/computer-programming/programming/looping/pt/for-loops-a-new-kind-of-loop>. Acesso em: 19 jan. 2018.

HTML DOM Events. Disponível em: <https://www.w3schools.com/jsref/dom_obj_event.asp>. Acesso em: 19 jan. 2018.


Neste capítulo, aprendemos os conceitos e teorias utilizados no desenvolvimento de códigos em JavaScript. Consegue perceber a flexibilidade desta linguagem? Vamos partir para algumas questões de aprendizagem e para a revisão final da unidade. Não se esqueça de olhar os sites e vídeos indicados. Eles que irão ampliar os seus horizontes de conhecimentos sobre o tema da unidade.
E na próxima unidade, professor, o que vamos aprender?
Agora chegamos ao final da disciplina. Mas não pare seus estudos. Continue se aprimorando em JavaScript, pois essa linguagem tem um grande potencial de mercado.

Glossário

- DOM
  O Modelo de Objeto de Documento (DOM) é uma interface de programação para documentos HTML, XML e SVG. Ele fornece uma representação estruturada do documento como uma árvore. O DOM define métodos que permitem acesso à árvore, para que os mesmos métodos possam alterar a estrutura, estilo e conteúdo do documento. O DOM fornece uma representação do documento como um grupo estruturado de nós e objetos, possuindo várias propriedades e métodos. Os nós também podem ter manipuladores de eventos que lhe são inerentes e, uma vez que um evento é acionado, os manipuladores de eventos são executados. Essencialmente, ele conecta páginas web a scripts ou linguagens de programação.
  Fonte: https://developer.mozilla.org/pt-BR/docs/Web/API/Document_Object_Model

- JavaScript
  JavaScript® (às vezes abreviado para JS) é uma linguagem leve, interpretada e baseada em objetos com funções de primeira classe, mais conhecida como a linguagem de script para páginas Web, mas usada também em vários outros ambientes sem browser como node.js, Apache CouchDB e Adobe Acrobat. É uma linguagem de script multiparadigma, baseada em protótipo que é dinâmico e suporta estilos de programação orientado a objetos, imperativo e funcional. Saiba mais sobre JavaScript.
  Fonte: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript

- Programação Orientada a Objetos
  Programação Orientada a Objetos é um paradigma de programação que usa abstração para criar modelos baseados no mundo real. POO usa várias técnicas vindas de paradigmas previamente estabelecidos, incluindo modularidade, polimorfismo e encapsulamento.
  Fonte: https://developer.mozilla.org/pt-BR/docs/conflicting/Learn/JavaScript/Objects

Referências

BROWN, Tim. Design Thinking. Uma Metodologia Poderosa para Decretar o Fim das Velhas Ideias. Alta Books, 2017.

FLANAGAN, David. JavaScript - O Guia Definitivo. Bookman, 2012.

FREEMAN, Eric. FREEMAN, Elisabeth. Use a Cabeça! HTML com CSS e XHTML. Alta Books, 2008.

MELO, Adriana. ABELHEIRA, Ricardo. Design Thinking and Thinking Design - Metodologia, Ferramentas e Uma Reflexão Sobre o Tema. Novatec, 2015.

MORRISON, Michael. Use a Cabeça! Javascript. Alta Books, 2008.

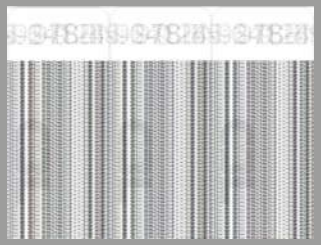
SHARP, Helen. ROGERS, Yvonne. PREECE, Jennifer. Design de Interação - Além da Interação Homem-computador, 3ª Ed., 2013.

SILVA, Maurício Samy. Fundamentos de Html5 e Css3. Novatec, 2015.

WATRALL, Ethan. Use a Cabeça! Web Design. Alta Books, 2009.

Currículo do autor

Salvador Alves de Melo Júnior. Bacharel em Engenharia Mecânica pela Universidade Federal de Uberlândia (UFU), Mestre na área de automação e robótica pela Universidade de Brasília (UnB) e Especialista em melhoria de processo de software pela Universidade Federal de Lavras (UFLA). É Professor universitário, com mais de 18 anos de experiência de sala de aula nos cursos ligados à área de Ciência da Computação.



INSTITUTO FEDERAL
Brasília

Secretaria de
Educação Profissional
e Tecnológica

MINISTÉRIO DA
EDUCAÇÃO