LÓGICA DE PROGRAMAÇÃO E ALGORITMOS

UNITAU digital

VALESCA ALVES CORRÊA

Valesca Alves Corrêa

Lógica de Programação e Algoritmos

Taubaté 2022

UNITAU digital

Autor Valesca Alves Corrêa

Unitau-Reitoria
Rua Quatro de Março,432, Centro
Taubaté – São Paulo. CEP:12.020-270
Central de Atendimento:0800557255

Polo Taubaté – Sede
Rua Conselheiro Moreira de Barros, 203 - Centro
Taubaté – São Paulo. CEP:12.010-080
Telefones: Coordenação Geral: (12) 3621-1530
Secretaria: (12) 3622-6050

edUNITAU

EXPEDIENTE EDITORA

Sistema Integrado de Bibliotecas - SIBi/ UNITAU
Grupo Especial de Tratamento da Informação – GETI

C824l Corrêa, Valesca Alves
Lógica de programação e algoritmos [recurso eletrônico] / Valesca Alves Corrêa. – Dados eletrônicos. -- Taubaté : EdUnitau, 2022.

Formato: PDF
Requisitos do sistema: Adobe
Modo de acesso: world wide web

ISBN: 978-65-86914-52-8 (on-line)

1. Lógica de programação. 2. Matrizes. 3. Programação. 4. Sistemas de informação. 5. Função. I. Título.

CDD – 005.1

Ficha catalográfica elaborada pela Bibliotecária Ana Beatriz Ramos – CRB-8/6318

Índice para Catálogo sistemático

Lógica de programação – 005.1
Vetores – 515.63
Matrizes – 512.9434
Programação – 005.1
Sistemas de informação – 004.2
Função – 515.63

Sumário

Recursos de Imersão:

Explorando ideias

Eu indico

Pensando juntos

Pípulas de conhecimento

Podcast

QRCode

Lógica de Programação e Algoritmos

Unidade I
Lógica e Algoritmos para a Programação de Computadores

Apresentar uma visão geral sobre como os computadores dependem dos programas ou dos apps para resolver problemas.
Apresentar a relação entre Lógica de Programação, algoritmos e construção de programas de computador.
Apresentar a estrutura de um algoritmo, os tipos de dados e os tipos de operadores.
Apresentar a estrutura mínima de um programa em linguagem Cv.

Introdução

Fonte: Freepik

Nesta Unidade, daremos início ao estudo da Linguagem de Programação.

A programação de computadores teve início com a programação de linguagens de baixo nível, mais ligadas ao *hardware*. Com a evolução tecnológica, passamos a programar com as linguagens de alto nível, que permitem uma interação abstrata com a máquina, explorando os recursos de hardware sem precisar interagir com ele.

Para garantir a independência em relação às linguagens de programação, foi desenvolvida a lógica computacional no aprendizado de programação. Trata-se de um procedimento desenvolvido em lógica computacional e que é o mesmo para qualquer linguagem de programação.

Assim, estudaremos o desenvolvimento da lógica computacional e sua conversão para a linguagem de programação estruturada C. Para isso, faremos uma introdução a esses conceitos, abordando aspectos iniciais relacionados à lógica de programação, suas formas de representação, além de abordarmos a estrutura básica de um programa em linguagem de programação C.

Esperamos que, ao final desta Unidade, você seja capaz de compreender os principais conceitos da área da Linguagem de Programação, de modo que esses saberes ajudem a construir sua

compreensão sobre o papel do profissional de TI nas práticas de uso e de desenvolvimento desse tipo de linguagem.

Bons estudos.

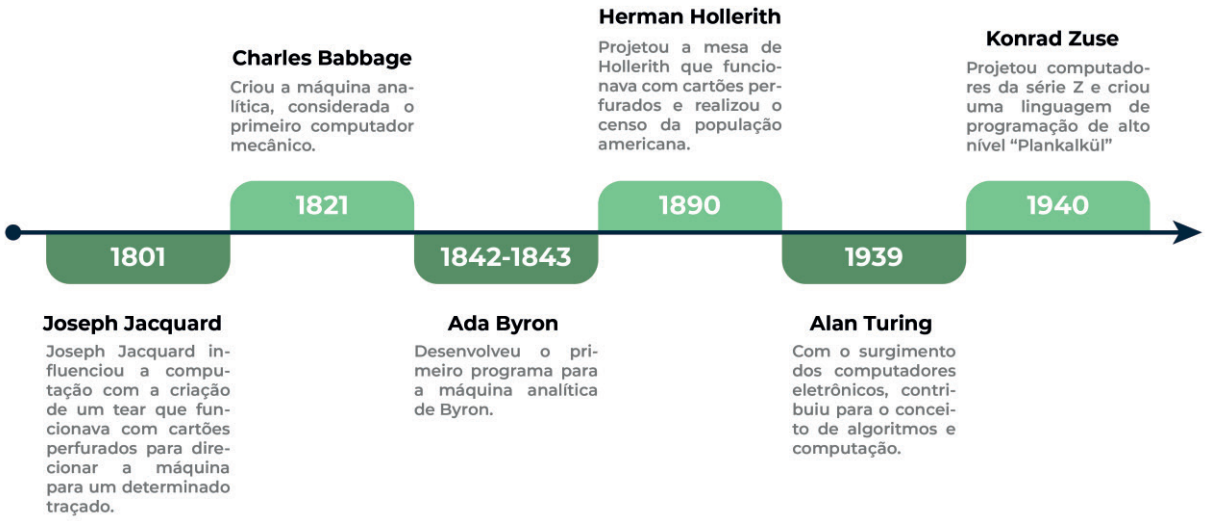
1.1 Programas de computador: o que são e como são construídos?

Os sistemas computadorizados fazem parte do dia a dia de inúmeras pessoas, considerando que desde a utilização de um caixa eletrônico até a realização de compras on-line, o uso de redes sociais, a realização de aulas e de reuniões remotas requerem o emprego de algum sistema computacional ou de algum *app*.

Fonte: Freepik

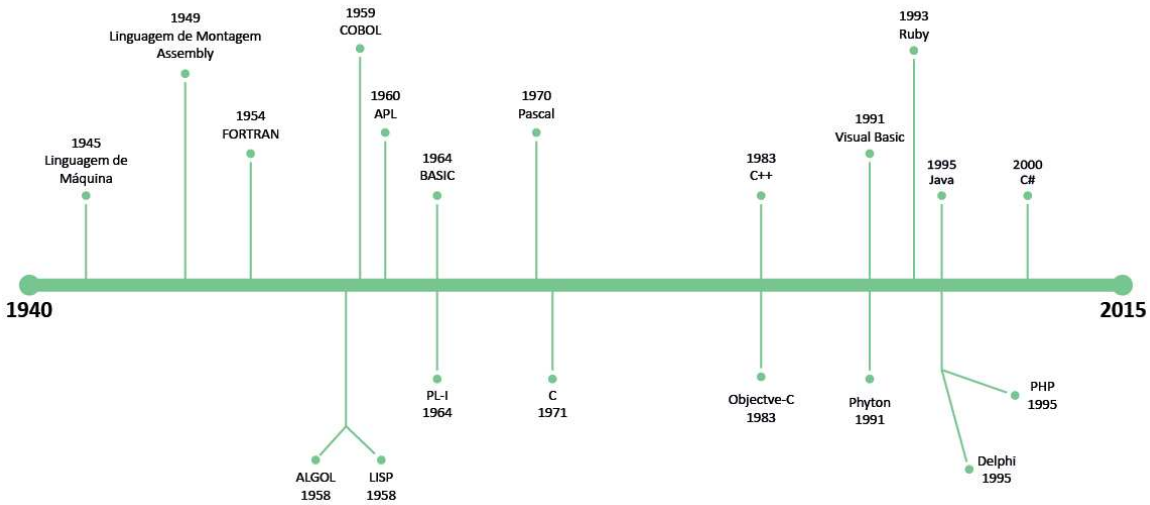
Cada um desses sistemas computadorizados conta com um programa computacional desenvolvido em uma linguagem de programação responsável por seu funcionamento.

Esses tipos de linguagem não são sempre os mesmos, e nem foram sempre os mesmos, pois ao longo do tempo alguns fatos históricos colaboraram com a criação das linguagens de programação como conhecemos. A seguir a síntese das contribuições para evolução dessas linguagens:



Fonte: Unitau Digital

A partir dessas contribuições, as linguagens de programação foram sendo desenvolvidas. É o que podemos observar na linha do tempo apresentada a seguir:



Fonte: Unitau Digital

Acompanhando a evolução do hardware e a necessidade de desenvolvimento de programação para outros dispositivos computacionais, outras linguagens foram surgindo.

Desde o desenvolvimento dos primeiros computadores, houve a necessidade de programar as atividades que as máquinas realizam. Essas atividades representavam tarefas complexas e limitadas, realizadas por componentes eletrônicos.

Fonte: http://www.asrepooya.com/

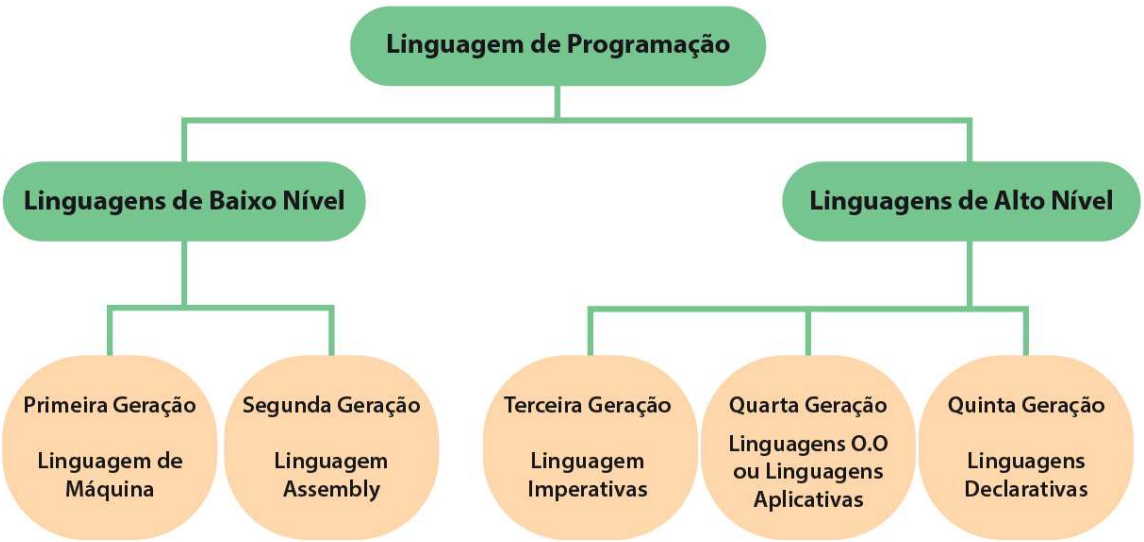
As linguagens de programação, então, foram desenvolvidas com o objetivo de facilitar a programação das tarefas que os computadores realizam.

As primeiras linguagens eram escritas em sequências binárias chamadas linguagens de máquina ou linguagem de baixo nível.

Esse tipo de linguagem é pouco intuitivo, com sintaxe complexa, sendo necessário conhecer o hardware do dispositivo para operá-la. A principal linguagem de baixo nível é o ***Assembly***.

As linguagens de alto nível, por sua vez, foram desenvolvidas para minimizar os conceitos relacionados com a máquina, visando facilitar a tarefa de instruir o computador e dessa forma colaborar com o aprendizado da máquina. São linguagens chamadas de imperativas, orientadas a objetos ou aplicativas e declarativas.

As linguagens de programação foram classificadas em gerações.



Fonte: Unitau Digital

As linguagens de programação imperativas já trabalham com o conceito de variáveis que são alteradas durante um programa, baseadas em instruções e comandos para definir as ações a serem tomadas na execução.

A partir de então, paradigmas foram alterando a forma de programar, onde as linguagens de programação orientada a objetos ou aplicativas focam a programação em classes que possuem características, assim como os objetos do mundo real.

As linguagens declarativas definem o que o programa deve realizar sem definir exatamente como realizar, desconhecendo-se o estado do programa.

Finalizando, os paradigmas de programação são outra forma de classificar as linguagens de programação em relação às suas funcionalidades.

Olá, aluno(a)! Indico a você assistir o filme "O Jogo da Imitação" com direção de Morten Tyldum. que se passa durante a segunda guerra mundial e nos conta um pouco da história de Alan Turing e sua contribuição científica para aquele período.

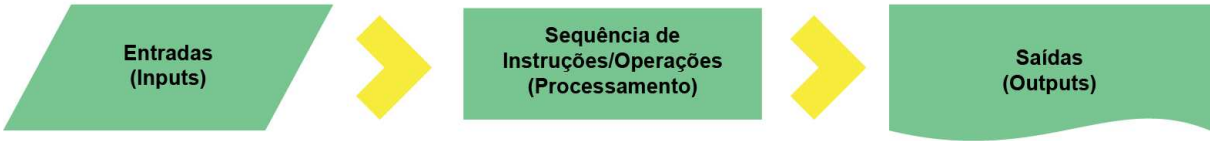
1.2 Algoritmos computacionais e manipulação de dados

Para a criação de programas e de sistemas computacionais é necessário determinar uma sequência lógica das ações que serão executadas, chamada de lógica de programação.

Independentemente da linguagem de programação que será trabalhada, é fundamental o desenvolvimento do raciocínio lógico antes da programação propriamente dita e para isso utilizam-se algoritmos computacionais.

Um algoritmo pode ser definido como uma sequência finita de passos para resolver um determinado problema.

O que são Algorítimos?

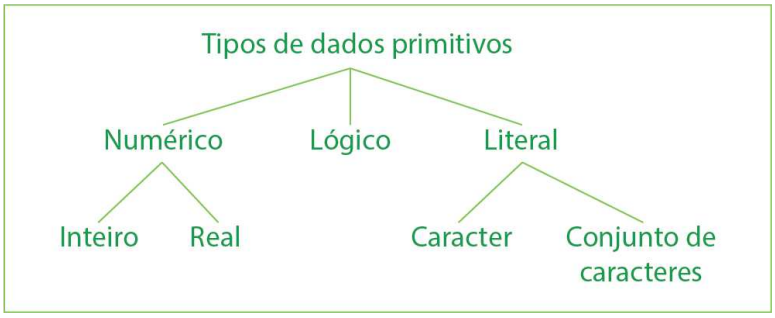


Fonte: Unitau Digital

Os algoritmos estão presentes no cotidiano das pessoas, como, por exemplo, na tarefa de consultar um saldo em um caixa eletrônico. Nesse caso, uma sequência de ações deve ser executada para que seja possível realizar a ação desejada. Em relação a esse mesmo exemplo, deve-se observar que algumas informações precisam ser fornecidas, como senha ou CPF, para que o sistema do caixa eletrônico identifique o cliente. Essas informações são as entradas ou *inputs*, assim como o saldo mostrado é uma informação de saída ou *output*. As ações internas do caixa eletrônico como localização do cliente e verificação do saldo são operações internas ou de processamento, realizadas a partir das informações de entrada fornecidas.

Os dados de entrada são necessários para a resolução de um problema e devem ser representados nos algoritmos de acordo com o tipo de informação que representam, para que ocorra reserva de memória e consequentemente manipulação desses dados.

Os dados numéricos podem ser divididos em inteiros (não tem casas decimais) e reais (tem representação inteira e decimal).



Os dados do tipo lógico podem representar apenas valores do tipo verdadeiro ou falso.

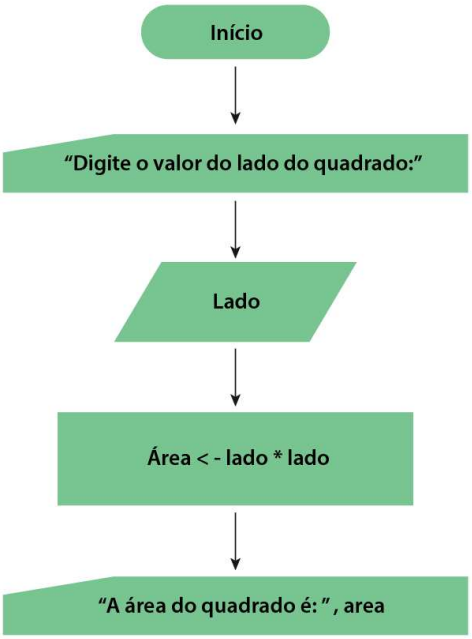
Os dados literais podem ser divididos em *caracter* (composto de um único digito) ou um conjunto de caracteres (composto por um conjunto de dígitos). São usados para representar caracteres alfabéticos, alfanuméricos ou caracteres especiais. O esquema aqui apresentado sintetiza os tipos de dados e suas representações.

Os algoritmos computacionais podem ser representados de forma escrita ou gráfica, mas sempre obedecendo uma lógica padronizada. As formas mais conhecidas de representação são descrição narrativa, fluxograma e pseudocódigo (linguagem estruturada ou portugol).

Na descrição narrativa, os algoritmos são expressos em uma linguagem natural, como exemplo, as receitas de bolo. Essa forma de representação pode gerar ambiguidades de interpretação e imprecisões, podendo acarretar inconsistências na transcrição para a linguagem de programação.

Os fluxogramas utilizam uma representação gráfica com símbolos padronizados para demonstrar a lógica computacional. Os principais símbolos são:

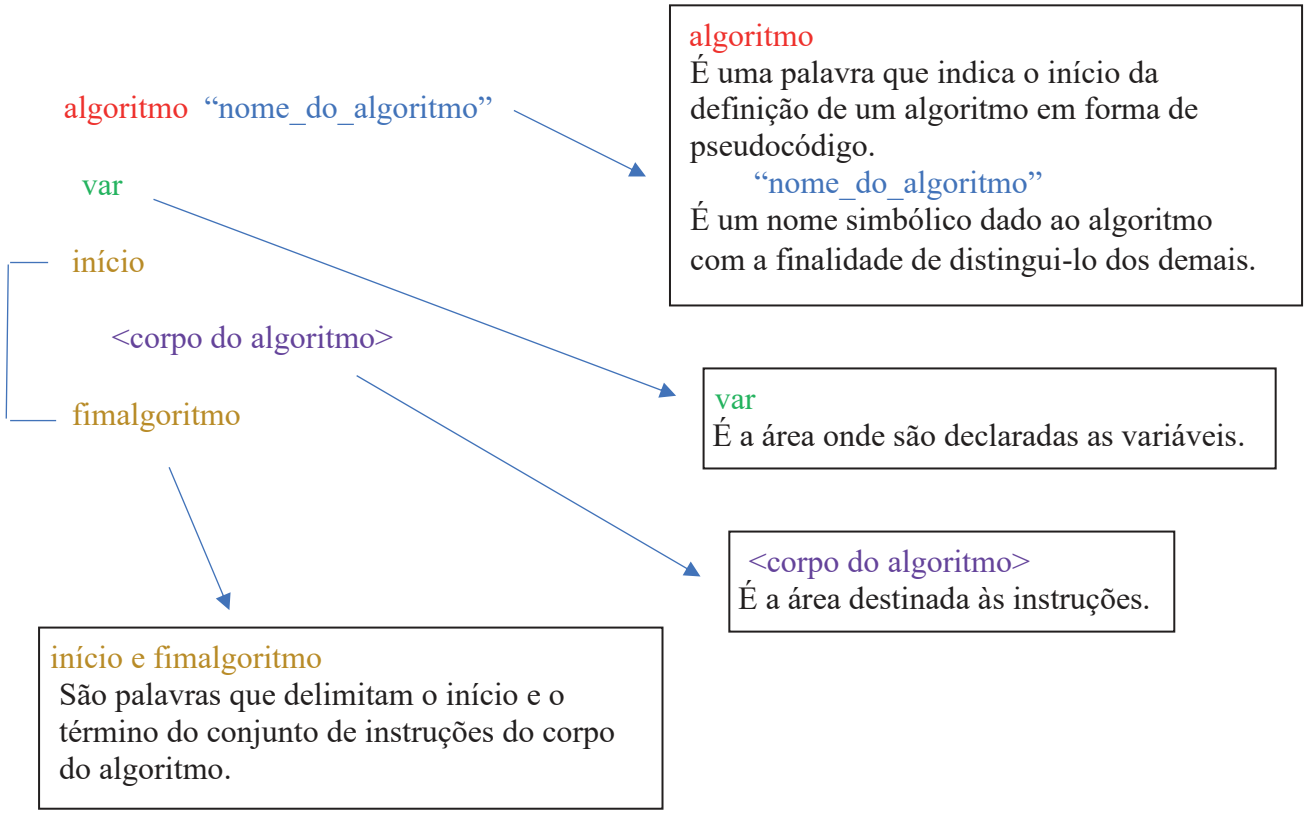
Figura	Significado
	Define o início e o fim do algoritmo
	Representa processamento e atribuições
	Representa a entrada de dados
	Representa a saída de dados
	Representa o processo seletivo ou condicional, possibilitando desvio do processamento
	Representa conector
	Identifica o sentido do fluxo de dados



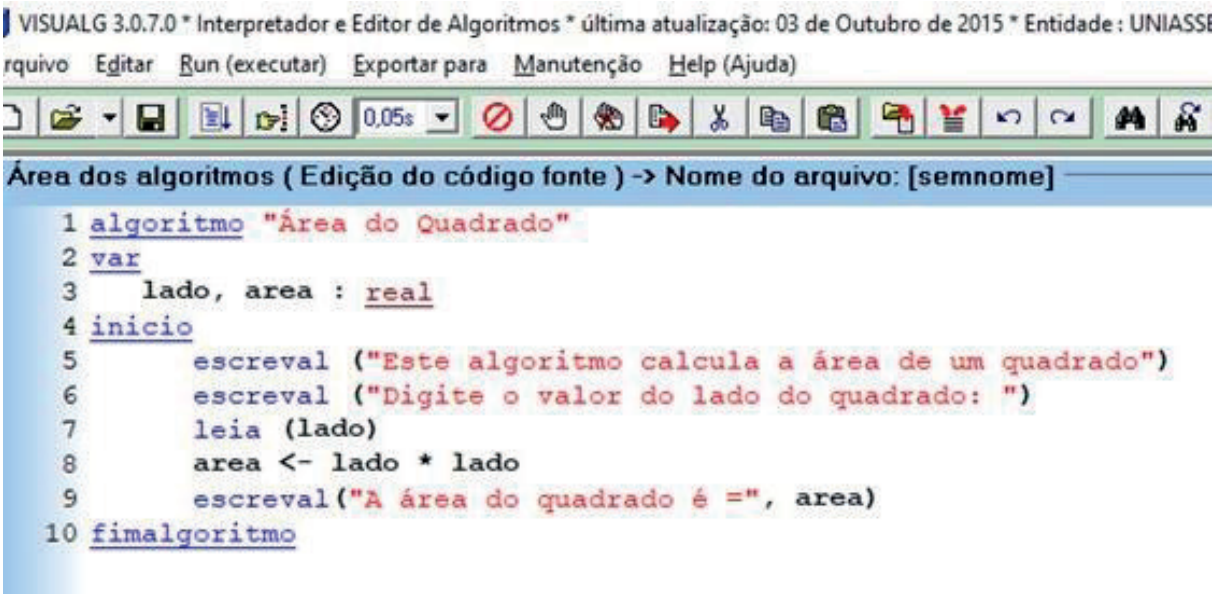
No fluxograma, é possível ver a resolução do cálculo da área de um quadrado.

O pseudocódigo é o tipo de representação que mais se aproxima das linguagens de programação. Por esse motivo, é a mais usada.

A forma geral da representação de um algoritmo na forma de pseudocódigo é a seguinte:



A resolução do cálculo da área de um quadrado na forma de pseudocódigo é a seguinte:



Vimos que o Visualg é um programa gratuito de edição, interpretação e execução de algoritmos, na representação de pseudocódico. É um programa de uso e distribuição livres, utilizado em diversas Iinstituições de Ensino no Brasil para o ensino de lógica de programação. O download desse ambiente poderá ser feito gratuitamente no link abaixo:

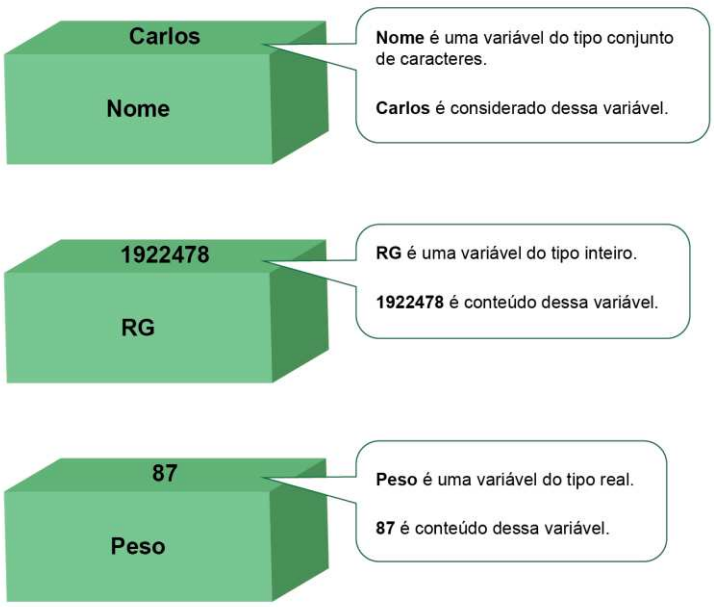
Link: download https://sourceforge.net/projects/visualg30/

VISUALG 3.0
O melhor interpretador de algoritmos

1.3 Primeiros passos na Linguagem de programação C

Os algoritmos utilizam os recursos de hardware mais básicos para executar os programas de computador.

A memória é responsável por armazenar e manipular dados. O recurso utilizado nos programas para escrever e ler dados da memória do computador é conhecido como variável, que é um espaço na memória que pode ser reservado e que pode ser nomeado.



Fonte: Unitau Digital

A memória pode ser vista como um conjunto de caixas com uma etiqueta de identificação (nome da variável), e cada caixa tem um conteúdo (informação). A imagem a seguir ilustra o conceito de memória.

Outro tipo de informação que pode ser representada são as constantes, ou seja, valores que podem ser associados a um nome, mas que não têm variação de conteúdo, por exemplo, o valor PI (3.14159265359…).

A resolução de problemas computacionais geralmente envolve algum tipo de expressão aritmética ou de lógica. Para essas expressões utilizam-se operadores que podem ser aritméticos, lógicos ou relacionais.

Os operadores aritméticos envolvem as operações mais simples, como soma, subtração, multiplicação, divisão e resto da divisão de inteiros, como sintetizado no quadro ao lado.

Símbolo	Significados
-	Subtração
+	Adição
*	Multiplicação
/	Divisão
%	Resto da divisão (módulo)

Deve-se observar que existe prioridade no uso dos operadores. Os parênteses em uma expressão têm maior precedência, seguidos dos operadores de multiplicação e de divisão e com menor precedência os operadores de adição e subtração.

Outra categoria de operadores envolve as expressões lógicas cujo valor resultante só pode ser um verdadeiro ou falso. Os operadores relacionados a essas expressões são os operadores lógicos e os operadores relacionais.

Operadores Lógicos

Operador	Função
and	lógico E
or	lógico OU
not	lógico de negação

1º Valor	Operador	2º Valor	Resultado
V	And	V	V
V	And	F	F
F	And	V	F
F	And	F	F
V	Or	V	V
V	Or	F	V
F	Or	V	V
F	Or	F	F
V	NOT		F
F	NOT		V

Nas expressões computacionais, também existe a prioridade entre os operadores aritméticos, lógicos e relacionais.

Prioridade	Operadores
1ª	Parênteses mais internos
2ª	Operadores aritméticos
3ª	Operadores relacionais
4ª	Operadores lógicos

Operadores Relacionais

Operação	Símbolo	Exemplo	Resultado
Igual	=	7 = 7	VERDADEIRO
Maior que	>	10 > 20	FALSO
Menor que	<	100 < 1000	VERDADEIRO
Menos ou igual a	<=	1.25 <= 2.50	VERDADEIRO
Maior ou igual a	>=	1234 >= 1234	VERDADEIRO
Diferente de	<>	10 <> 10	FALSO

Fonte: Unitau Digital

Para compreender melhor o desenvolvimento da lógica de programação, pode-se utilizar um ambiente de desenvolvimento integrado chamado IDE (IDE - Integrated Development Environment), para escrever e compilar os algoritmos e os programas computacionais como ferramenta de apoio.

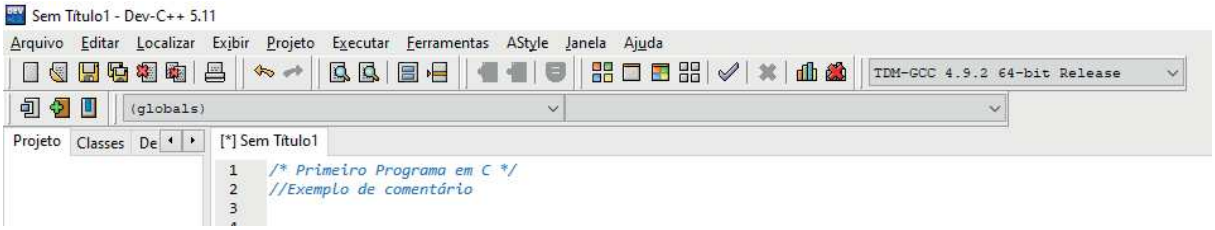
Para o desenvolvimento de algoritmos é recomendado o programa VisuALG, que permite a edição e a execução de pseudocódigos, sendo um software de livre uso e distribuição.

Também sendo de distribuição livre, o IDE Dev C++ é recomendado para edição e execução da programas em linguagem C.

Para ambos os ambientes, podem ser feitos o download na Internet de forma gratuita.

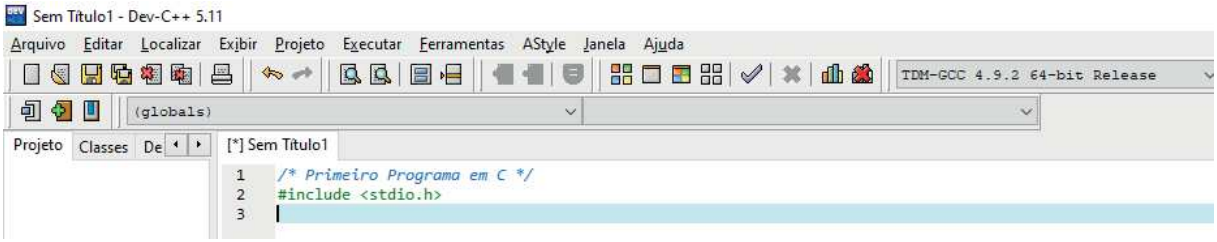
A linguagem de programação C é imperativa, procedural, de alto nível e compilada. Um programa em linguagem C apresenta regras lexicais e semânticas distintas que formam a estrutura do programa.

Toda linha de um programa em C é compilada e executada, exceto os comentários, representados por símbolos /* */ se passarem de uma linha ou // para uma única linha, como ilustra a imagem a seguir:



Em um programa em linguagem C, é necessário começar com um pré-processamento iniciado pelo símbolo # com o comando **include** que recebe como parâmetro um nome de arquivo normalmente com extensão "**.h**" que significa cabeçalho ("*header*"), os quais são arquivos de definições.

Para definir funções de entrada e de saída de informações é usado o arquivo "stdio.h" que pertence à biblioteca padrão C.



Um programa em C pode ter várias funções. Assim, é necessário definir uma função principal que indica onde o programa será executado. A função **main( )** faz essa indicação e é precedida da palavra **int** que define um tipo de informação.

A linguagem C é case-sensitive, ou seja, faz distinção entre letras maiúsculas e minúsculas; então, **main ( )** é diferente de **Main ( )** que é diferente de **MAIN ( ).**

Todas as instruções que comporão a função **main( )** estarão na área delimitada por chaves.

Conforme abordamos, o Dev-C++ é um ambiente de desenvolvimento integrado livre que utiliza os compiladores do projeto GNU para compilar e executar programas. Suporta as linguagens de programação C e C++, e possui toda a biblioteca ANSI C. A IDE é escrita em Delphi. O download desse ambiente poderá ser feito gratuitamente no link abaixo:

Link : https://sourceforge.net/projects/orwelldevcpp/

1.4 Síntese da Unidade

Nesta Unidade, vimos que os sistemas computacionais fazem parte da rotina diária das pessoas e são desenvolvidos por programas em alguma linguagem de programação.

Conversamos sobre fatos históricos que contribuíram para o desenvolvimento das linguagens de programação desde 1801, mas as primeiras linguagens foram difundidas na década de 40, sendo então classificadas em gerações.

Também vimos que um algoritmo pode ser definido como uma sequência finita de passos para resolver um determinado problema. Pode ser representado por descrição narrativa, que é pouco eficiente para problemas computacionais; fluxogramas, que utilizam símbolos geométricos para definir o fluxo de dados; e pseudocódigo, que mais se aproxima das linguagens de programação.

Além disso, você aprendeu que os tipos de dados primitivos que são manipulados nos algoritmos são numéricos, lógicos e literais; que as variáveis são endereços de memória referenciados por um nome e um tipo, enquanto as constantes apresentam um nome e um valor fixo.

Os tipos de operadores que podem ser usados nos programas computacionais são os aritméticos, lógicos e relacionais e possuem precedência que deve ser respeitada.

Por fim, vimos que, para uma melhor compreensão do desenvolvimento da lógica de programação, pode-se utilizar um ambiente de desenvolvimento integrado chamado IDE, tanto para o desenvolvimento de algoritmos como para as linguagens de programação, e que a estrutura mínima de um programa em linguagem C apresenta comentários, comandos de pré-processamento e função principal.

Esperamos que os saberes apresentados tenham ajudado você a compreender os conceitos introdutórios da Lógica de Programação, tema que aprofundaremos ainda mais nas próximas Unidades desta disciplina.

1.5 Para Saber Mais

Livros

Recomendo que você faça a leitura do Capítulo 1 do livro "Lógica de Programação Algorítmica", do Organizador Sérgio Guedes (2014) da editora Pearson Education do Brasil, para se aprofundar um pouco mais no conteúdo abordado.

Unidade II
Instruções de entrada e saída e estruturas de seleção (decisão)

Introdução

Fonte: freepik.com

Nesta Unidade, daremos início ao estudo das instruções de entrada e saída e das estruturas de seleção ou decisão. A partir das instruções de entrada e saída e suas diferentes sintaxes, vamos conhecer algoritmos e programas executados de forma sequencial, iniciando o aprendizado da resolução de problemas reais. Para garantir a continuidade do desenvolvimento de algoritmos, aprenderemos a trabalhar com estruturas de seleção ou decisão, que permitem o desvio do fluxo de execução a partir do resultado de uma condição. Assim, veremos duas dessas estruturas aplicadas nos algoritmos, suas diversas formas de representação e diferentes níveis de aplicação com condições vinculadas aos operadores relacionais e lógicos. Na sequência, veremos como essas estruturas são representadas na Linguagem de programação C. Esperamos que, ao final desta Unidade, você seja capaz de iniciar o trabalho de resolver algoritmos e programas com resolução sequencial e com estruturas de seleção ou decisão.

Bons estudos!!!

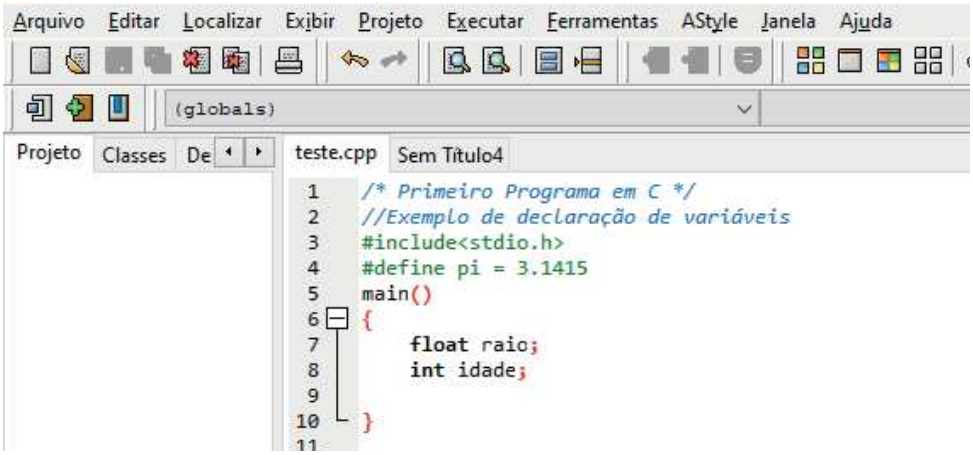
2.1 Tipos de dados e as instruções de entrada e saída em linguagem C

De forma similar à vista nos algoritmos, as variáveis também precisam ser declaradas em linguagem C. Entretanto, na estrutura de um programa em C, as variáveis podem ser declaradas ao longo do desenvolvimento do programa, porém antes de serem utilizadas.

Os tipos de dados podem ser vistos na tabela a seguir, sendo os tipos primitivos representados por **char** (caracter), **int** (inteiro) e **float** (real). Os tipos derivados com variação de tamanho serão estudados com o avanço da disciplina.

Tipo	Bits	Faixa
char	8	- 127 a 127
unsigned char	8	0 a 255
signed char	8	-127 a 127
int	16	-32.767 a 32.767
unsigned int	16	0 a 65.535
signed int	16	-32.767 a 32.767
short int	16	-32.767 a 32.767
unsigned short int	16	0 a 65.535
signed short int	16	-32.767 a 32.767
long int	32	-2.147.483.647 a 2.147.483.647
signed long int	32	-2.147.483.647 a 2.147.483.647
unsigned long int	32	0 a 4.294.967.295
float	32	Seis dígitos de precisão
double	64	Dez dígitos de precisão
long double	80	Dez dígitos de precisão

Dentro da estrutura de um programa em C, as declarações de variáveis ficam:



Observe que ao final das declarações do exemplo foram incluídos os terminadores ponto-e-vírgula (**;**). Então, o final de cada instrução em linguagem C deve ser finalizado com um ponto e vírgula.

Valores constantes, ou seja, que não alteram seu valor no programa, também podem ser declarados. Devem ser incluídos antes da função principal **main( ),** com a diretiva **define,** nesse caso, sem a necessidade do ponto e vírgula.

Os algoritmos e programas precisam de instruções que permitam a interação com os usuários e isso é feito através dos comandos de entrada e saída.

Os comandos de saída são mensagens que são reproduzidas na execução e favorecem a interação com o usuário e podem ser informativas, solicitantes ou mensagens que mostram uma resposta. Na tabela a seguir veremos os comandos de saída nos algoritmos e em Linguagem C.

	Algoritmo	Linguagem C
Instrução	escreva( ) ou escreval( )	printf( )
Exemplo de mensagem informativa	escreval("Algoritmo que calcula a área de um quadrado")	printf("\n Programa que calcula a área de um quadrado")
Exemplo de mensagem solicitante	escreval("Digite o valor do lado do quadrado")	printf("\n Digite o valor do lado do quadrado")
Exemplo de mensagem que emite resposta	escreval("O valor da área do quadrado = ", area)	printf("\n O valor da área do quadrado = %f", area)

Iniciando com a representação no algoritmo, pode-se utilizar a instrução **escreva**( ), que reproduzirá as mensagens em sequência, ou seja, uma após a outra, ou **escreval** ( ), que reproduzirá as mensagens uma em cada linha. Esteticamente, o **escreval** ( ) é mais recomendado.

Devemos observar que as mensagens estão englobadas por aspas duplas, portanto, o uso de outro tipo de aspas acarretará erro.

Nas mensagens do tipo informativa e solicitante, o conteúdo entre aspas será reproduzido na execução; já na mensagem que emite resposta, será reproduzida a mensagem entre aspas e a seguir o valor da uma variável. O exemplo da tabela faz referência à variável **area**, ou seja, nomes que se encontram fora da área limitada pelas aspas são variáveis.

Na representação em Linguagem C, temos a instrução **printf**( ). A funcionalidade dessa instrução é idêntica ao **escreval** ( ), apenas com diferenças de sintaxe.

A primeira variação trata da inclusão do caractere de controle **\n**, responsável por fazer o cursor avançar para a próxima linha na execução. Os principais caracteres de controle são **\"** exibe uma única aspa, **\'** exibe um único apóstrofo e \\ exibe uma única barra invertida.

Analogamente aos algoritmos, nas mensagens do tipo informativa e solicitante, o conteúdo entre aspas será reproduzido na execução.

Na mensagem que exibe resposta, temos além do caractere de controle **\n**, uma outra simbologia, o especificador de formato **%f**, que indica o tipo de informação que será exibida. A tabela a seguir mostra os possíveis especificadores de formato:

Especificador	Representa
%c	um único caracter
%o, %d, %x	um número inteiro em octal, decimal ou hexadecimal
%u	um número inteiro em base decimal sem sinal
%ld	um número inteiro longo em base decimal
%f, %lf	um número real de precisão simples ou dupla
%s	uma cadeia de caracteres (*string*)
%%	um único sinal de porcentagem

Dessa forma, na mensagem que exibe resposta no exemplo da tabela, onde está o especificador **%f**, significa que ali será exibido um valor **float**, que corresponde ao valor da variável **area**.

Os comandos de entrada de dados permitem que um usuário, durante a execução, forneça um valor em resposta a uma mensagem solicitante e armazene esse valor na memória. Na tabela a seguir, veremos para os algoritmos e para o programa em Linguagem C.

	Algoritmo - Pseudocódigo	Linguagem C
Instrução	leia( )	scanf( );
Exemplo de leitura de um valor que será armazenado na variável lado	leia(lado)	scanf("%f", &lado);

No algoritmo, através da instrução **leia**, quando o valor for digitado, será armazenado na variável denominada lado.

No programa em linguagem C, a funcionalidade da instrução **scanf** será a mesma da instrução **leia**, mas com algumas especificações na sintaxe. O primeiro parâmetro é o especificador **"%f"**, mostrando que ali será lido um valor do tipo **float.** O segundo parâmetro conta com o operador **&** antes do nome da variável, e informa o endereço de memória na qual a variável será alocada.

A seguir, um exemplo do uso dessas instruções com suas respectivas execuções no algoritmo e na Linguagem C, em que poderão ser observadas as aplicações dos comandos de entrada e saída. Devemos observar na sequência a expressão do cálculo da área. A leitura deve iniciar pela expressão, ou seja, o resultado da multiplicação de lado vezes lado será atribuído à

variável **area**. A simbologia representada no algoritmo para atribuição é uma seta invertida formada pelo sinal de menor e um traço (<-). Já na linguagem C, a simbologia será um sinal de igual.

Algoritmo - Pseudocódigo	Linguagem C

`algoritmo "área de um quadrado"`
 	
`var`
`  area, lado: real`
 	
`inicio`
`ESCREVAL ("Este algoritmo calcula a área de um quadrado")`
`ESCREVAL ("Digite o valor do lado")`
`LEIA (lado)`
`area <- lado*lado`
`ESCREVAL ("Área do quadrado = ", area)`
`fimalgoritmo`
	`//Programa que calcula a área de um quadrado`
	`#include<stdio.h>`
	`int main()`
	`{`
	`float area, lado;`
	`printf("\n Calcula a área de um quadrado");`
	`printf("\n Digite o valor do lado \n");`
	`scanf("%f", &lado);`
	`area = lado*lado;`
	`printf("\n A área = %f", area);`
	
	`}`

`Início da execução`
`Este algoritmo calcula a área de um quadrado`
`Digite o valor do lado`
`5`
`Área do quadrado =  25`
 	
`Fim da execução.`
	`Calcula a ßrea de um quadrado`
	`Digite o valor do lado`
	`5`
	
	`A ßrea = 25.000000`
	`--------------------------------`
	`Process exited after 3.725 seconds with return value 0`
	`Pressione qualquer tecla para continuar. . .`

Finalizando, estudamos um algoritmo e um programa em Linguagem C com estrutura sequencial, ou seja, executado linha a linha. Abordamos também as instruções de entrada e saída e suas formas de representação.

Você sabia que o nome das variáveis tem regras de formação. Essas regras também são válidas para o nome de constantes, funções e métodos.

Não podemos utilizar os caracteres especiais: - ç - ~ - ´ - ' - " - % - & - (-) - { - } - [-] e os operadores aritméticos: - + - - - * - /

Também, não devemos começar um nome de uma variável com um número, ou seja, deve começar com uma letra ou com o caractere _ (underline), podendo os números serem incluídos na continuidade do nome.

Uma variável também deve ser composta de uma única palavra e na necessidade de juntar duas palavras usamos o caractere _ (underline).

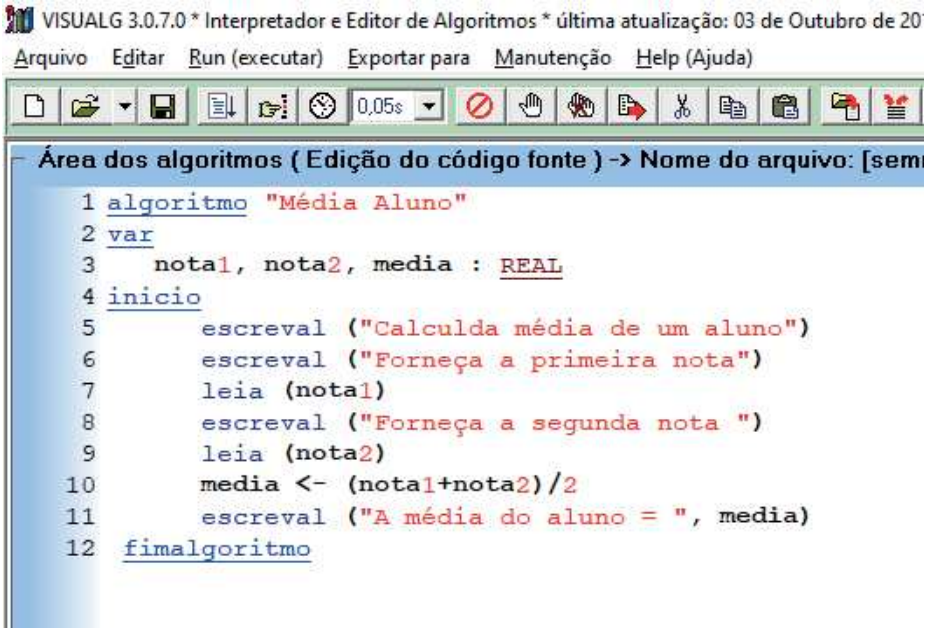
2.2 Representação de estruturas de seleção ou decisão em Algoritmos

Até o momento, vimos a resolução de algoritmos de forma sequencial, porém a maioria dos problemas computacionais exige outros recursos, como, por exemplo, a tomada de decisões.

As estruturas de seleção ou decisão permitem o desvio do fluxo de instruções dependendo do resultado de uma condição.

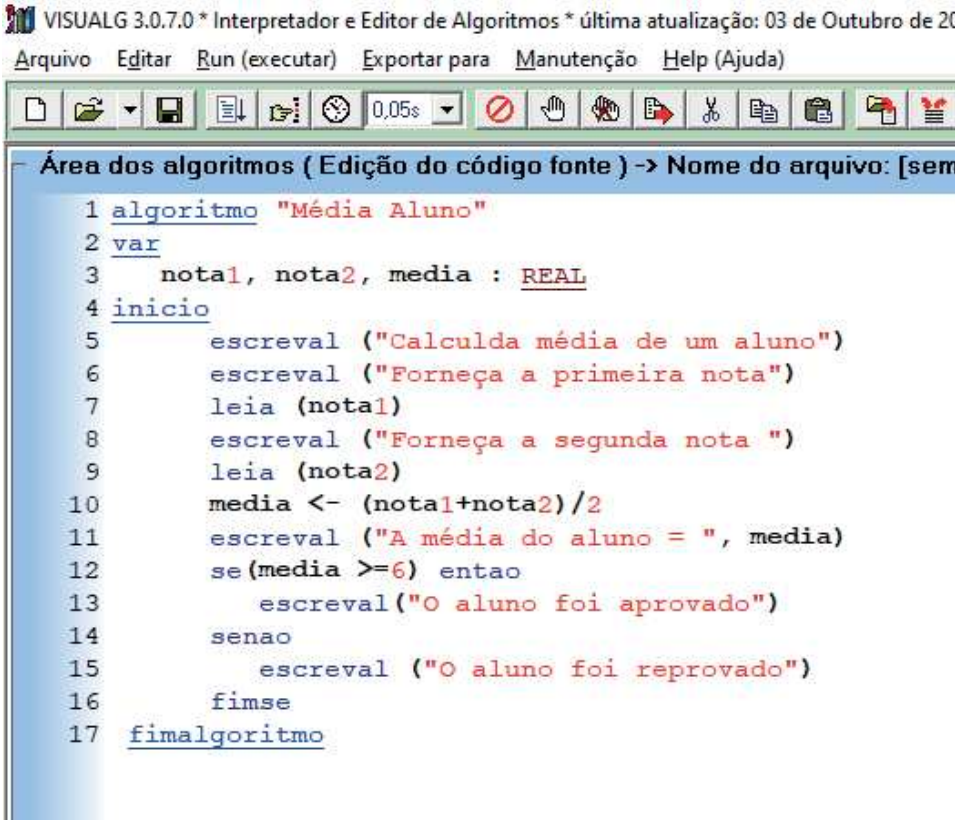
A estrutura **Se - Senao** permite essa representação nos algoritmos. Para exemplificar, vamos considerar um algoritmo que calcule a média de um aluno, a partir de duas notas fornecidas:

O

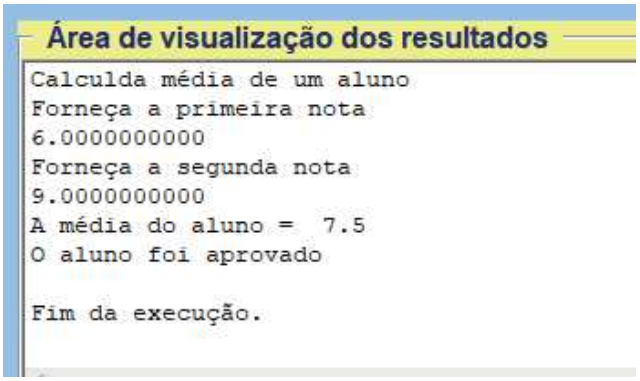


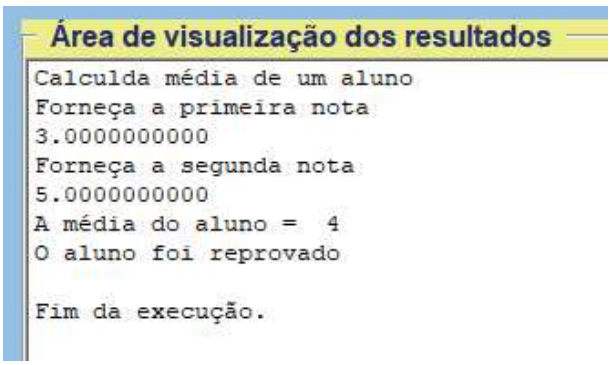
algoritmo

exemplificado é eficiente quanto ao cálculo da média, porém, para mostrar se o aluno foi aprovado ou reprovado, considerando uma média para aprovação 6,0, podemos usar a estrutura **Se – Senao**. O algoritmo a seguir mostra essa inclusão.



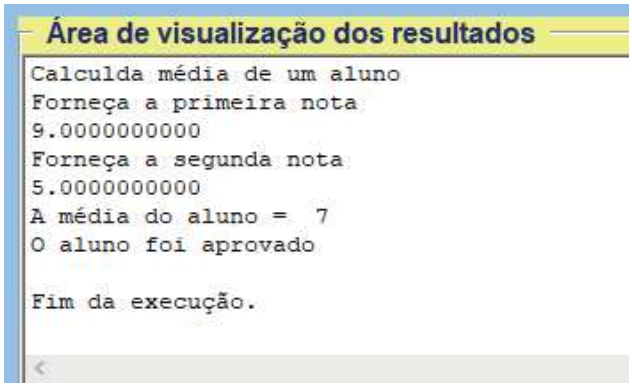
Devemos observar que as palavras **entao** e **senao** que compõem a estrutura **se** não são acentuadas. A seguir, como ficariam as execuções para as duas situações possíveis:

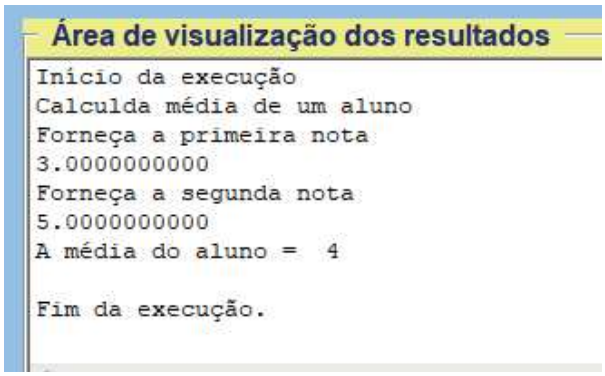




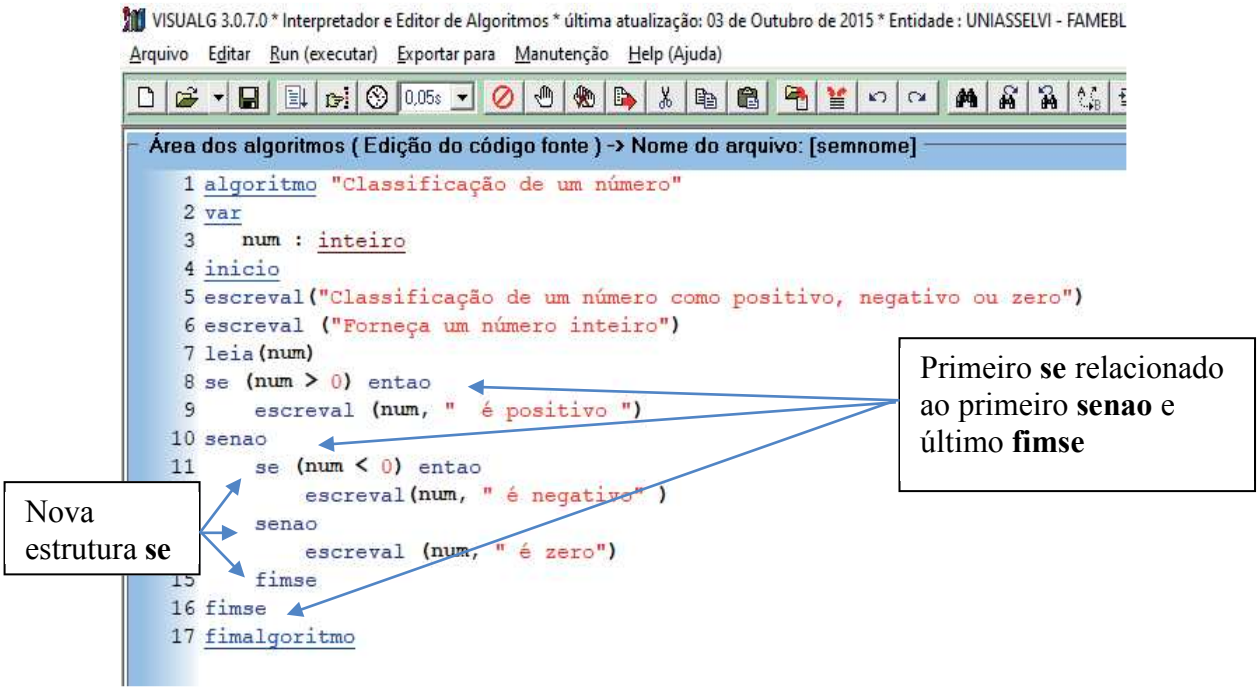
A estrutura **se** também pode ser usada na forma simplificada, ou seja, sem a possibilidade de desviar o fluxo para o resultado da condição falsa. Para exemplificar, vamos considerar a situação de mostrar apenas se o aluno for aprovado.

A seguir, como ficariam as execuções para as duas situações possíveis, com média maior ou igual a 6 e inferior a 6:





Observamos que quando a condição é falsa, a instrução **escreval** não é executada. Outra possibilidade do uso da estrutura **se** é quando temos mais de duas possibilidades de ocorrência; nesse caso, usamos estruturas aninhadas, ou seja, uma estrutura **se** dentro da outra. Para exemplificar, vamos considerar a situação de classificar um número inteiro como positivo, negativo ou zero (três possibilidades). O algoritmo a seguir apresenta a solução.



Observe que após o primeiro **senao**, temos uma nova estrutura **se** e que **c**ada estrutura **se** exige seu próprio **fimse**. A seguir, os possíveis resultados:

```
Início da execução
Classificação de um número como positivo, negativo ou zero
Forneça um número inteiro
-3
 -3 é negativo

Fim da execução.
```

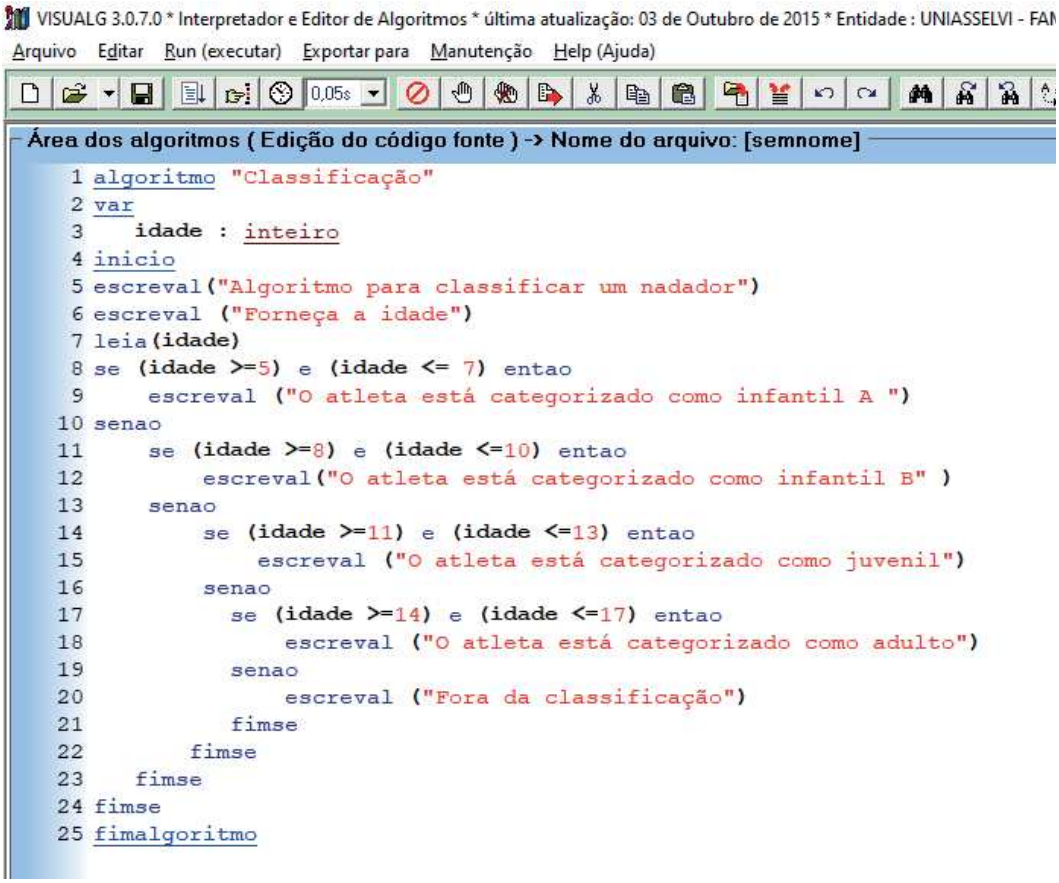
```
Início da execução
Classificação de um número como positivo, negativo ou zero
Forneça um número inteiro
2
 2 é positivo

Fim da execução.
```

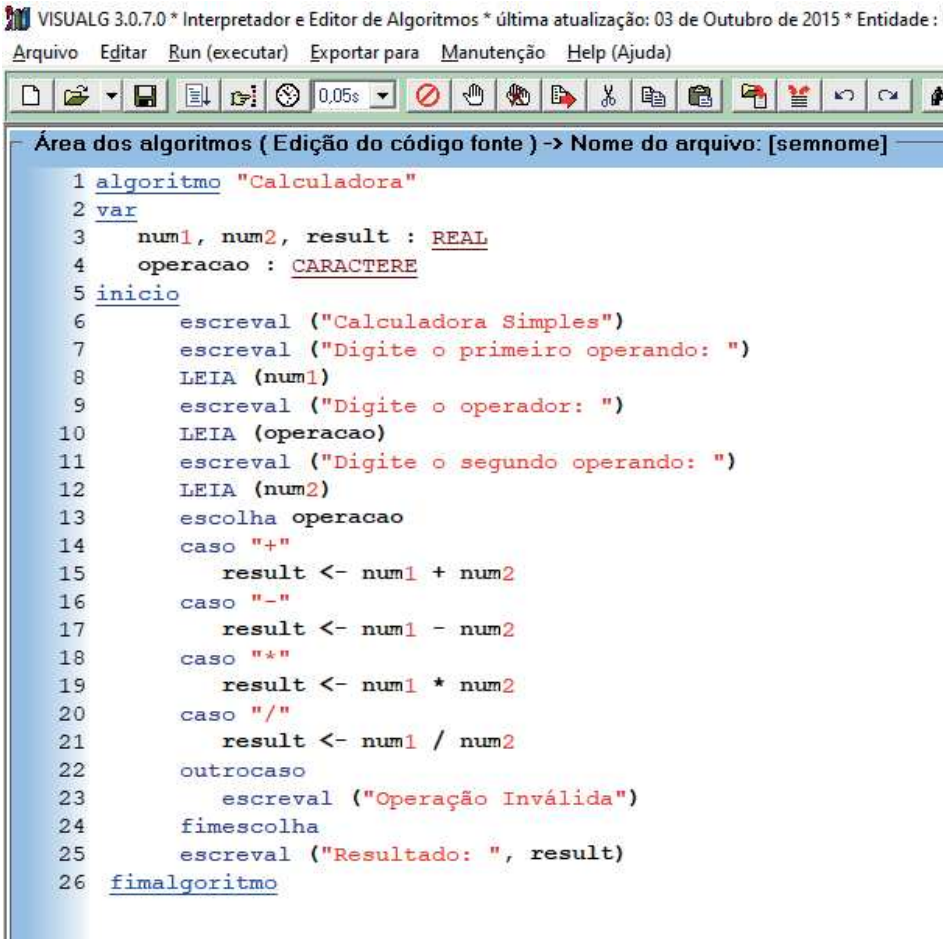
```
Início da execução
Classificação de um número como positivo, negativo ou zero
Forneça um número inteiro
0
 0 é zero

Fim da execução.
```

Também podemos utilizar os operadores lógicos junto com as estruturas de seleção nas situações que envolvem um intervalo fechado, por exemplo. Mostramos essa aplicação no algoritmo a seguir, que classifica um nadador em determinada categoria em relação à idade, mostrando a necessidade do uso do operador lógico "**e**" para criar um limite entre as idades.



Outra instrução de seleção é a instrução **escolha-caso**, mais indicada quando múltiplas opções devem ser representadas. Para exemplificar, vamos ver o algoritmo de uma calculadora para as quatro operações aritméticas.



Começaremos a análise desse algoritmo pela declaração das variáveis, as variáveis **num1** e **num2** vão representar os operandos e **result** será responsável pelo resultado da operação. Observe que foram declaradas na mesma linha separadas por vírgula, sendo isso possível, por serem do mesmo tipo. Também foi declarada a variável **operacao** do tipo caractere para representar o operador.

Após a solicitação e leitura dos valores, o comando **escolha** vai determinar qual **caso** será executado a partir do conteúdo da variável **operacao**.

A instrução **escolha** possui uma opção não obrigatória para tratar a estrutura se nenhuma opção for satisfeita, o **outrocaso**, que só será executado se nenhuma das alternativas for válida.

A linguagem de programação C, possui um conjunto de palavras reservadas, que são comandos de uso muito específico dentro da linguagem, não podendo ser usadas para nomear variáveis ou funções.

Essas palavras são: **asm, auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while**

Observe que as palavras reservadas da linguagem C são sempre escritas em letras minúsculas.

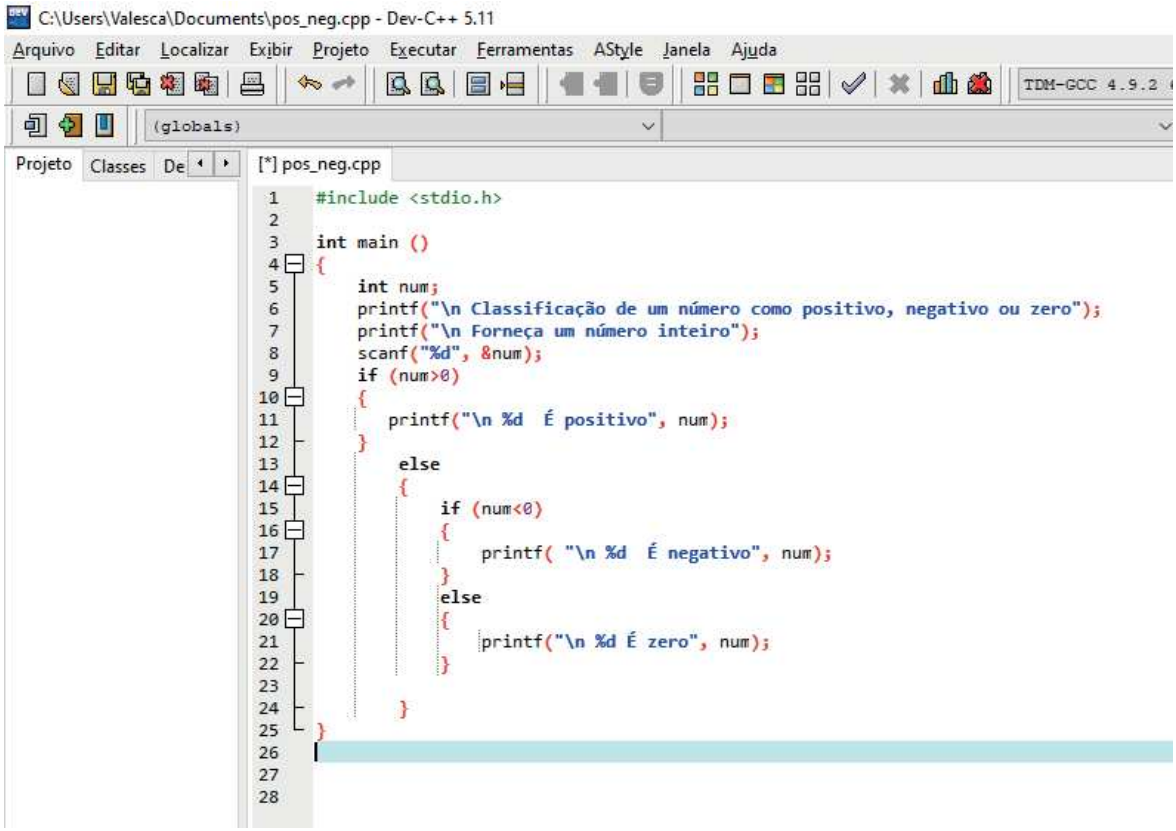
2.3 Representação de estruturas de seleção ou decisão em linguagem C

A representação das estruturas de seleção ou decisão em Linguagem C segue a mesma lógica vista para os algoritmos, ou seja, para cada instrução vista no Tópico anterior, temos uma instrução correspondente em Linguagem C.

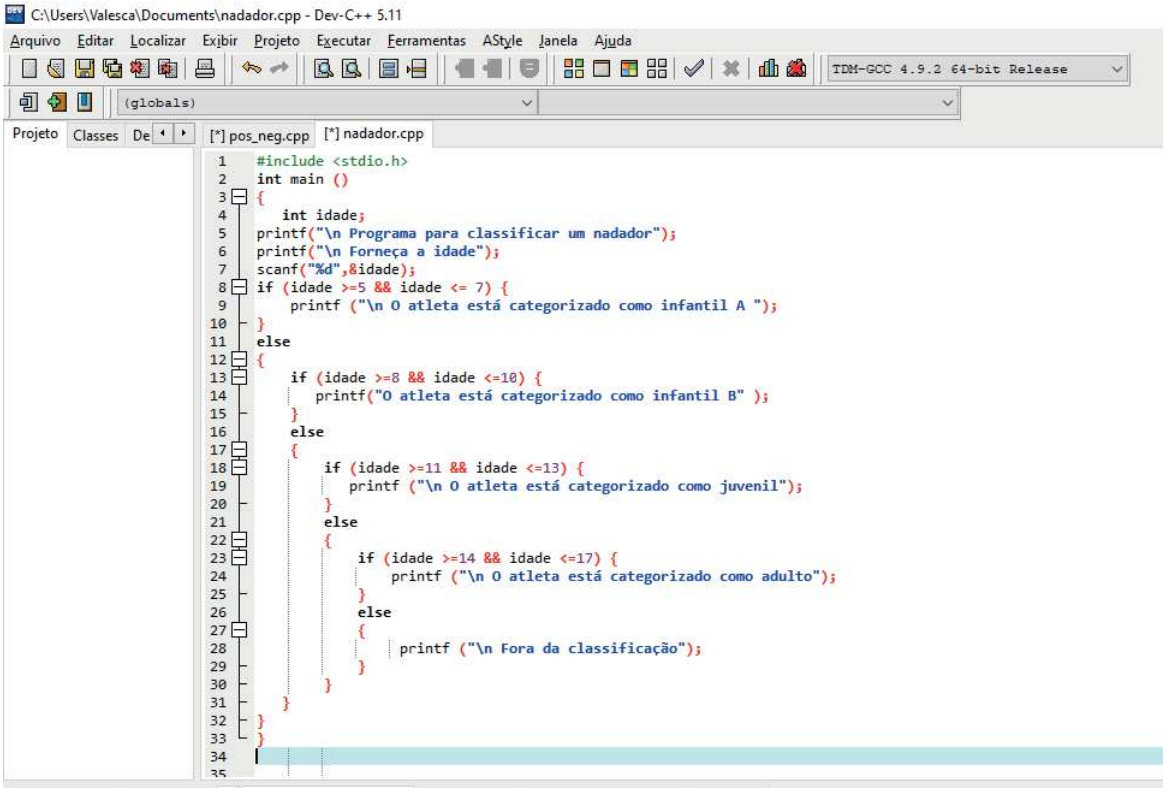
A tabela a seguir mostra a correspondência entre as instruções:

	Algoritmo - Pseudocódigo	Linguagem C
Instrução	Se – senao	if - else
	Escolha - caso	Switch() - case

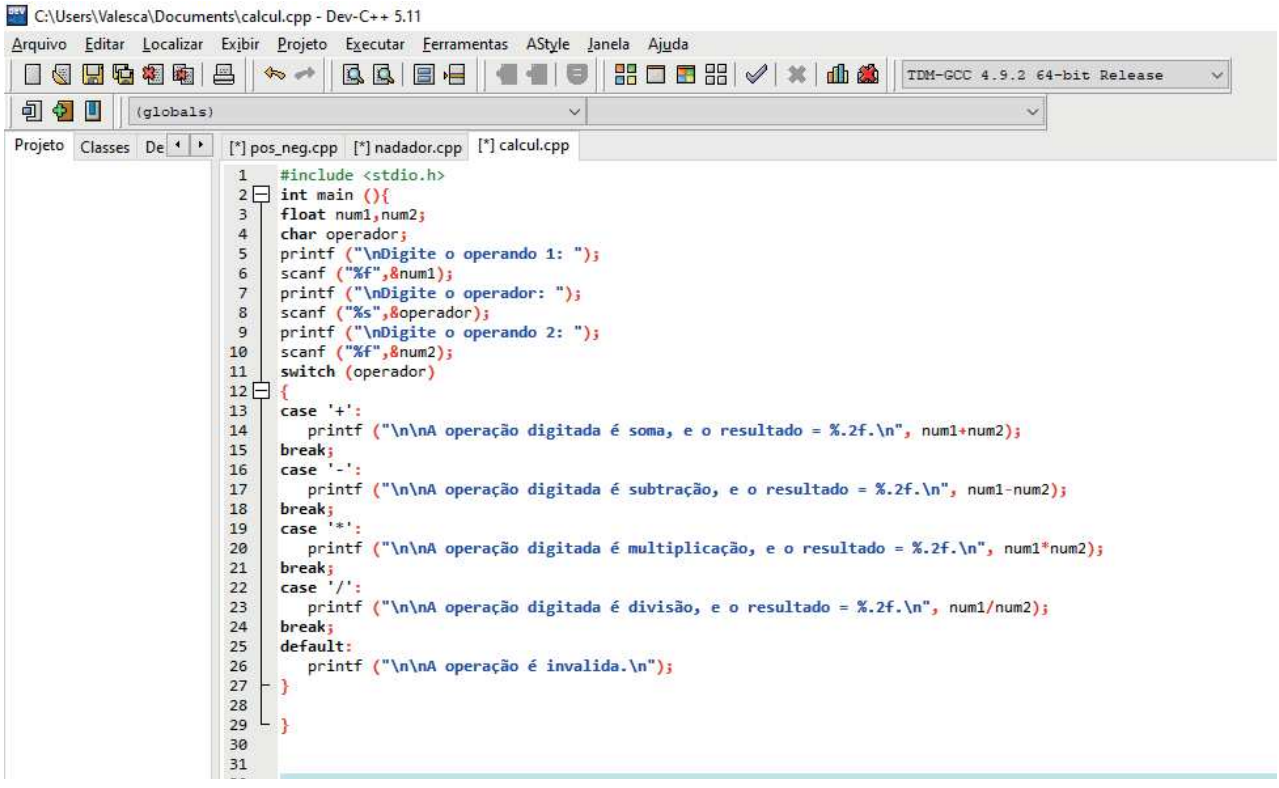
Para exemplificar, vamos considerar a mesma situação de classificar um número inteiro como positivo, negativo ou zero (três possibilidades). Como o programa a seguir apresenta a solução com **if´s** aninhados.



Na Linguagem C, não temos termos correspondentes para as palavras **entao** e **fimse** dos algoritmos, ao invés disso, são usadas chaves para delimitar os comandos correspondentes a condição verdadeira e falsa. A lógica de execução permanece a mesma. No próximo exemplo, mostramos o programa da classificação do nadador, dando ênfase à utilização do operador lógico **AND** simbolizado por **&&** no programa. Outra observação quanto à sintaxe desse operador se refere a que as condições devem estar englobadas por um único conjunto de parênteses.



Veremos no próximo exemplo a estrutura switch – case.



Podemos observar a mesma lógica utilizada no algoritmo aplicada no programa em Linguagem C. A sintaxe do comando **switch**, englobada por chaves, é feita onde cada **case** é encerrado por um **break**. Para a situação de nenhum **case** ser satisfeito, usa-se a opção **default:** seguido do comando a ser executado.

Uma variação na formatação de saída foi usada nesse programa:

```
13  case '+':
14     printf ("\n\nA operação digitada é soma, e o resultado = %.2f.\n", num1+num2);
15  break;
```

Inicialmente foi usado **%.2f** que significa que naquele local da frase que será reproduzida terá um valor do tipo **float** com duas casas decimais. A seguir, temos a operação **num1** + **num2** sendo realizada no local destinado às variáveis, fato permitido, pois o resultado da operação será um valor **float**.

Finalizando, podemos observar que a estrutura **switch – case** permite uma única entrada, enquanto a estrutura **if** não tem limitantes quanto ao número de entradas.

2.4 Síntese da Unidade

Nesta Unidade, vimos como as instruções de entrada e saída são representadas nos algoritmos e nos programas em Linguagem de programação C. Em consequência dessas representações, pudemos desenvolver algoritmos e programas com resolução sequencial, porém, que já representam soluções de problemas reais.

Também vimos que podemos alterar o fluxo de execução de um algoritmo e de um programa com as estruturas de seleção ou decisão. A partir desse ponto, evoluímos de problemas mais simples com poucas verificações até problemas com operadores lógicos e relacionais para trabalhar com diversas condições.

Além disso, você aprendeu que tanto para os algoritmos como para os programas temos duas estruturas que podem ser usadas, usando a mesma lógica computacional, diferenciando-se pela sintaxe.

Por fim, vimos que, para uma melhor compreensão do desenvolvimento da lógica de programação, precisamos aprender novos recursos para ampliar o campo de aplicação.

Esperamos que os saberes apresentados tenham ajudado você a compreender os diversos recursos de seleção ou decisão da Lógica de Programação, que continuarão a ser aplicados nas próximas Unidades desta disciplina.

2.5 Para Saber Mais

Recomendo que você faça a leitura do Capítulo 2 do livro “Lógica de Programação Algorítmica”, do Organizador Sérgio Guedes (2014) da editora Pearson Education do Brasil, para se aprofundar um pouco mais no conteúdo abordado.

Unidade III
Sistemas de Informação: Tecnologia e Sistema

Os objetivos de aprendizagem da Unidade III envolvem os três aspectos importantes referentes à tecnologia da informação: os sistemas de informação e suas tecnologias, os conceitos de sistemas empregados em âmbitos organizacionais e, por fim, falaremos dos tipos de sistemas de informação comumente empregados nas organizações.

INTRODUÇÃO

Esta Unidade está subdividida em três subtópicos.

No primeiro, a reflexão gira em torno do conceito de tecnologia da informação, apontando-a como um poderoso suporte para a tomada de decisões estratégicas em âmbito organizacional. Também estudaremos as áreas que compõem os departamentos de TI nas organizações, bem como os ganhos e os benefícios que podem ser alcançados com os usos de diferentes sistemas da informação.

No segundo, vamos aprofundar conceitos ligados aos sistemas da informação empregados em âmbito organizacional, destacando a classificação de sistemas abertos e fechados, bem como os níveis organizacionais que podem ser beneficiados pelo emprego de diferentes sistemas de informação.

Para finalizar, o tópico 3.3 apresenta uma linha evolutiva do uso dos sistemas de informação nas organizações, considerando um período que tem início na década de 1960 e chega até os dias atuais. Estudaremos, ainda, os sistemas transacionais, os gerenciais e os estratégicos, imprescindíveis para a boa gestão das informações e dos processos organizacionais.

Esperamos que, ao final deste percurso, você seja capaz de compreender o papel dos sistemas de informação nas práticas de boa gestão organizacional, de modo a refletir sobre a importância dessa área e dos profissionais dessa área para a produtividade e o ganho de qualidade nos processos gerenciais, operacionais, estratégicos e táticos de cada organização.

Bons estudos!

3.1 Sistemas de Informação

Como temos estudado, a evolução das tecnologias da informação, ao longo do tempo, modificou nossas atividades em praticamente todas as áreas de conhecimento. Vivemos na Era da Informação, também chamada Era Digital ou Era Tecnológica, na qual a tecnologia está amplamente disponível e acessível para utilização por grande parte da população em suas tarefas cotidianas.

Com o amplo acesso à tecnologia, o que inclui computadores, redes e outras tecnologias, o mundo se tornou mais conectado e com menos barreiras físicas, permitindo a ampliação das fronteiras de conhecimento. Além disso, os computadores se tornaram mais acessíveis, compactos e potentes.

A Tecnologia de Informação (TI), definida como todas as atividades e soluções desempenhadas por recursos computacionais, surge com essa evolução, passando a contribuir com indivíduos e organizações, tendo a responsabilidade de tratar e transformar a informação considerada, atualmente, como um ativo de extremo valor. Essa evolução é representada na imagem a seguir.

Fonte: Unitau Digital

Com a evolução dessas tecnologias, as organizações passaram a utilizar recursos tecnológicos diversos para otimizar suas atividades. Hoje, as organizações têm investido

massivamente em Tecnologias de Informação, uma vez que precisam obter, armazenar, proteger, tratar e gerenciar suas próprias informações.

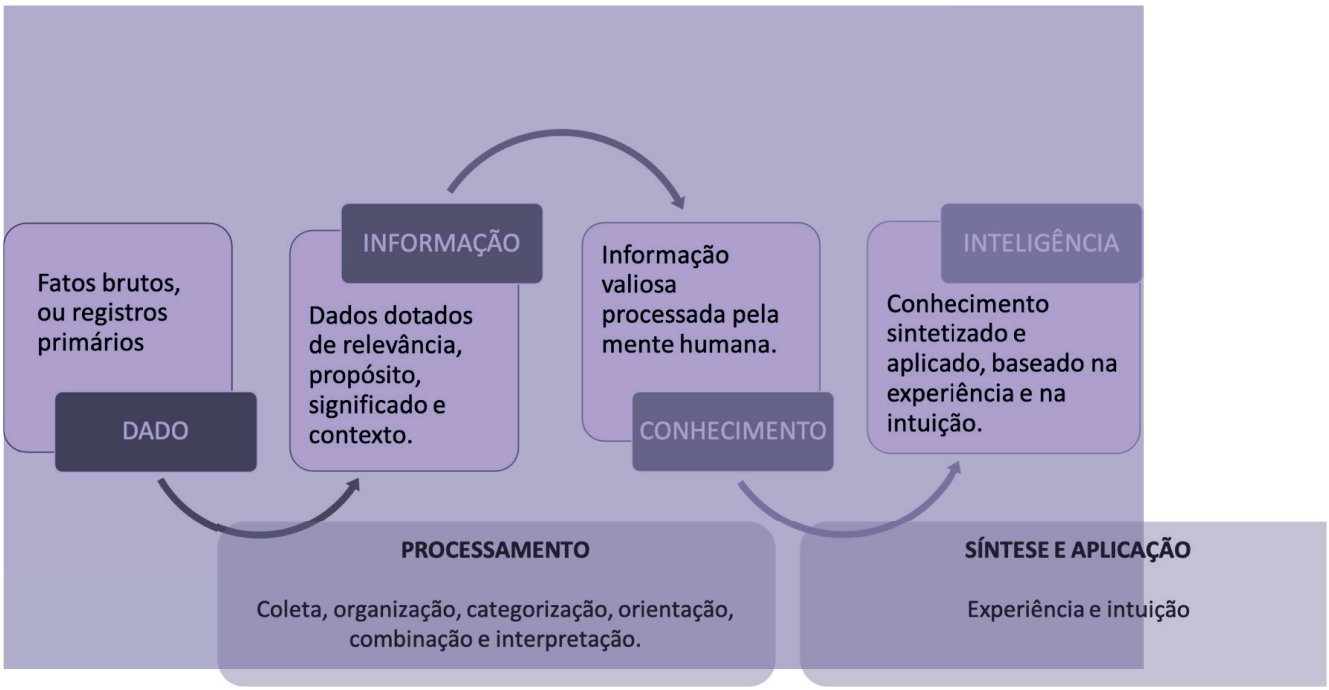
Para obter os resultados esperados, é preciso que recursos de *hardware* e de *software* sejam compatíveis com as atividades realizadas. Para garantir o bom uso desses recursos, foi sendo instituído o setor, ou o Departamento, de TI nas organizações. As áreas que compõem esses setores são as seguintes:

- **Governança de TI:** considerada uma área mais gerencial, trata de processos, políticas, normas e estratégias relacionadas à TI;
- **TI Operacional:** trata da rotina da organização, como implementação e suporte de sistemas, controle de segurança, entre outras ações;
- **Infraestrutura de TI:** área ligada a equipamentos, como servidores e estruturas de rede, por exemplo.

Outras áreas ou subáreas de Tecnologia da Informação podem ser definidas, dependendo da estrutura organizacional e do modelo de negócios. Os profissionais de Tecnologia da Informação podem atuar tanto em tarefas de inovação, criando novos dispositivos e aplicações, como em setores de planejamento, gerenciamento, desenvolvimento, manutenção, segurança, atualização e suporte de sistemas computacionais.

Sendo a informação um dos principais ativos de uma organização, ganham destaque os dados que, após serem manipulados, comparados, ordenados e interpretados, são transformados em informações relevantes para a tomada de decisões fundamentada e subsidiada em diversos âmbitos organizacionais.

O processo de tratamento dos dados por sistemas de informação considera um caminho que parte do dado e, depois de tratado, é considerado um saber, um conhecimento. O dado “bruto” não é suficiente para funcionar como subsídio para a tomada de decisões organizacionais. Na imagem a seguir, vemos o processo que vai da captação do dado até sua transformação em conhecimento.



Fonte: Unitau Digital

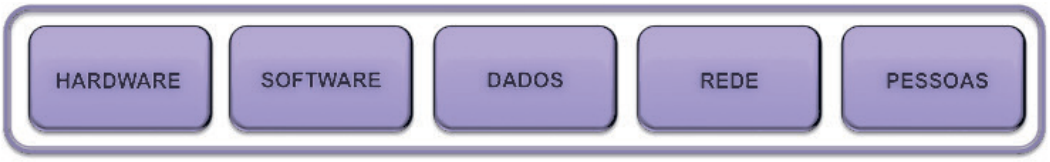
Assim, o papel dos sistemas de informação são cruciais para que os dados de uma organização sejam tratados, processados, de modo que sirvam para orientar processos de gerenciamento.

Para o gerenciamento eficaz de uma empresa, alguns desenvolvimentos foram fundamentais, causando impacto na utilização de TI, tais como:

a) a *internet* e os novos modelos de negócio;
b) a Globalização;
c) a reengenharia dos processos empresariais (conjunto de ações para redesenhar os processos de negócio, buscando produtividade, eficiência e qualidade necessárias para se adequar às novas tecnologias e tornar a empresa competitiva);
d) utilização da TI, em busca de vantagem competitiva.

Esses fatores levaram ao conceito de empresa digital, ou seja, aquela que visa, ao máximo, ao uso da Tecnologia de Informação em todos seus processos, uma vez que seus escritórios não precisam estar fisicamente na linha de produção. Os processos, como reposição de estoque e relacionamento com clientes e fornecedores, são realizados de forma remota.

Os Sistemas de Informação (SI) visam compreender e analisar o uso das Tecnologias de Informação nos processos gerenciais e administrativos de uma organização, sendo compostos pelos seguintes elementos: *hardware*, *software*, dados, rede e pessoas, tal como sintetizado na imagem a seguir.



Fonte: Unitau Digital

Ao utilizar os Sistemas de Informação, os benefícios de uma organização incluem:

a) Ganhos econômicos (redução de pessoal, rapidez na resolução dos problemas, melhor produtividade, redução de estoques e outros);
b) Ganhos de produtividade (mais qualidade e flexibilidade nos processos e mais eficiência e rapidez na tomada de decisões).

Para o gerenciamento efetivo das informações, nas organizações, é fundamental ter o conhecimento sobre os sistemas de informação e sobre as tecnologias envolvidas. A relação entre esses "espaços" e esses "atores" que fazem parte dos Sistemas é representada na imagem a seguir.

Fonte: Unitau Digital

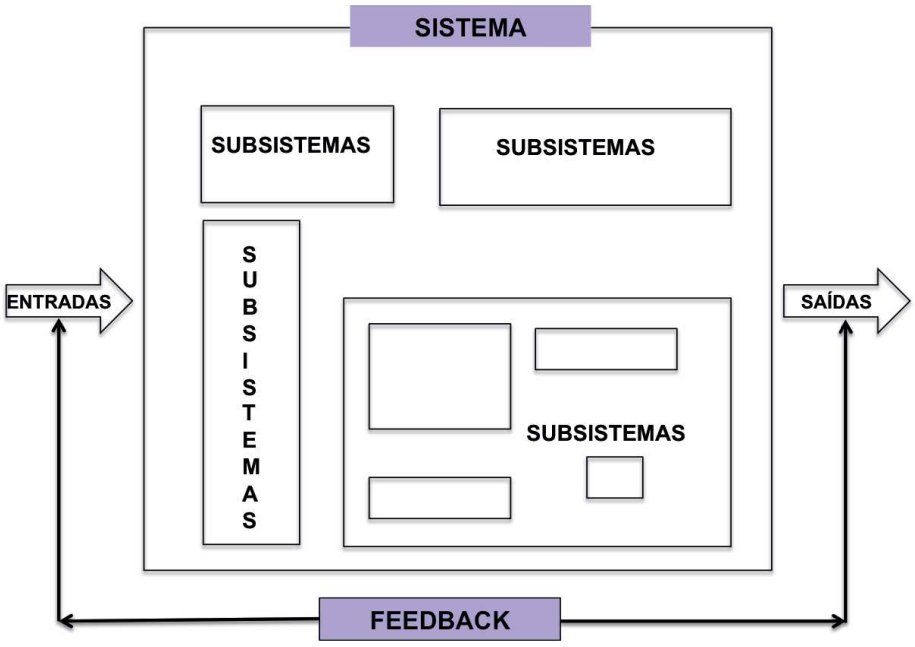
Você sabe o que significa globalização?

"[...] é um conjunto aparentemente bastante heterogêneo de fenômenos que ocorreram ou ganharam impulso a partir do final dos anos 80, como a expansão das empresas transnacionais, a internacionalização do capital financeiro, a descentralização dos processos produtivos, a revolução da informática e das telecomunicações, o fim do socialismo de Estado na ex-URSS e no Leste Europeu, o enfraquecimento dos Estados nacionais, o crescimento da influência cultural norte-americana etc., que desenharam uma espécie de 'sociedade mundial', ou seja, uma sociedade na qual os principais processos e acontecimentos históricos ocorrem e se desdobram em escala global".

(ALVAREZ, M. C. Cidadania e direitos num mundo globalizado. São Paulo: Perspectiva, 1999, p. 97).

3.2 Conceitos de Sistemas

Pode-se definir um sistema como um conjunto de componentes e subsistemas que formam um todo. Os componentes e subsistemas podem interagir para a obtenção de objetivos comuns e facilitar as operações de controle. A relação desses componentes e subsistemas é representado na figura a seguir.

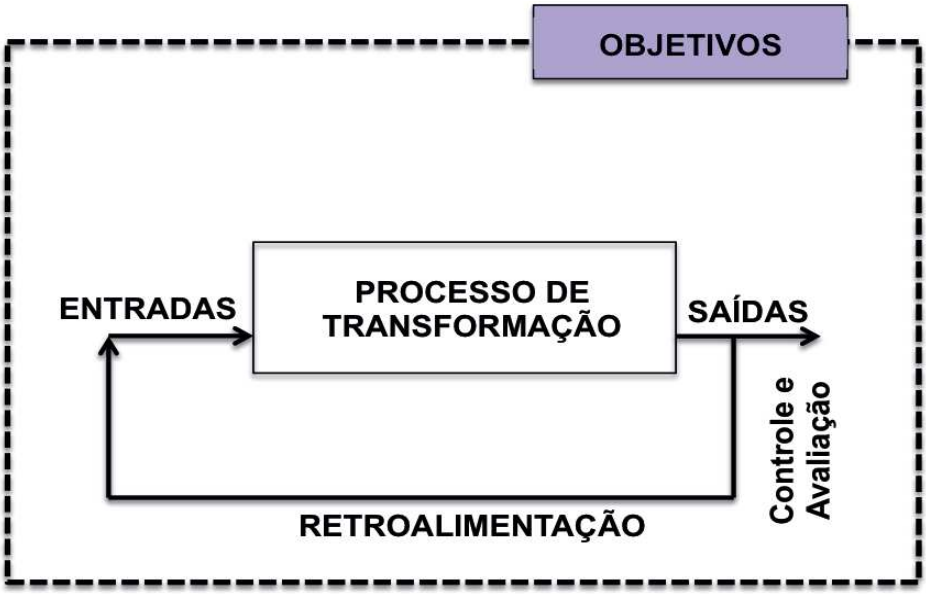


Um sistema é composto pelos seguintes componentes:

- Objetivos: a necessidade de existir, ou o motivo pelo qual foi criado;
- Entradas: as informações do sistema;

- Processos de transformação: a transformação de uma entrada em um produto ou saída desejada;
- Saídas: os resultados ou as respostas aos objetivos do sistema;
- Controles e avaliações: verificação das saídas, para averiguar se estão coerentes com os objetivos criados;
- Retroalimentação: reintrodução das saídas, sob a forma de informação.

Esses componentes estão esquematizados na figura a seguir:



Um sistema pode ser classificado como aberto, caso ocorra interação com o meio exterior. Nesse contexto, uma organização pode ser considerada um sistema aberto, pois, para ser eficiente e seguir seus próprios objetivos, precisa se comunicar e se flexibilizar para se adaptar aos diversos elementos com os quais terá de interagir.

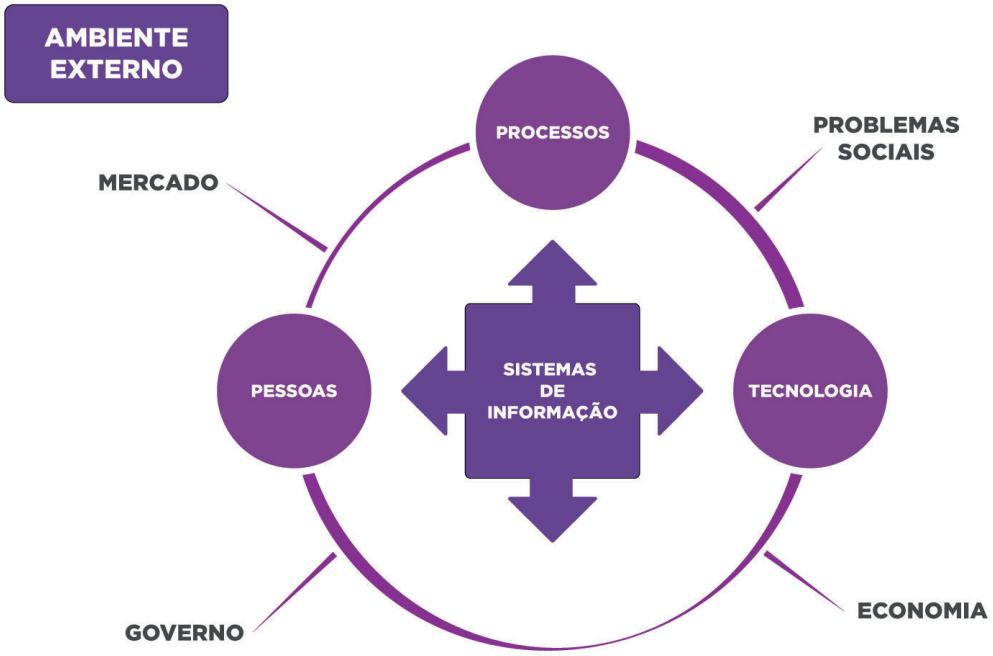
Os sistemas consideram fronteiras para delimitar e organizar seus subsistemas e componentes. As fronteiras de uma organização não se limitam apenas ao meio externo à organização, elas devem também ser consideradas entre os diversos subsistemas e componentes do próprio sistema, favorecendo a coordenação de esforços e a comunicação, evitando, assim, as disfunções.

Um Sistema Informatizado visa à redução da duplicidade de informações, evitando redundâncias e erros, minimizando os esforços das equipes e permitindo o bom funcionamento do sistema como um todo. Quando uma empresa for implantar um Sistema de Informação, essa ação levará a mudanças e ao desenvolvimento de novas competências, por meio das quais as operações

de profissionais devem ser facilitadas, criando, inclusive, cenários de integração da tecnologia com o ambiente de cada profissional.

Se a implantação for de uma organização cujas atividades são resultantes de um histórico de atividades, ela passará por uma transição e também precisará de uma integração. A partir dessa integração, cada profissional atuará manipulando sua informação, com a orientação da equipe de Sistemas de Informação. Então, os dados e a informação, assim como os próprios Sistemas de Informação, serão rearranjados, evitando perda de tempo e de recursos computacionais e financeiros.

A dimensão de uma organização, assim como seu próprio histórico, está relacionada à complexidade de dados e informações, por exemplo, ao volume de serviço que pode ser derivado do número de fornecedores, do número de clientes, do volume de vendas, da complexidade dos produtos, entre outros dados e informações. Um Sistema de Informação de uma organização pertence a um sistema aberto, cuja função é a integração com os componentes internos e externos, visando ao melhor tratamento da informação, com a meta de obter os resultados mais eficazes. Na imagem a seguir, temos um esquema dos elementos constitutivos de sistemas de informação em organizações.



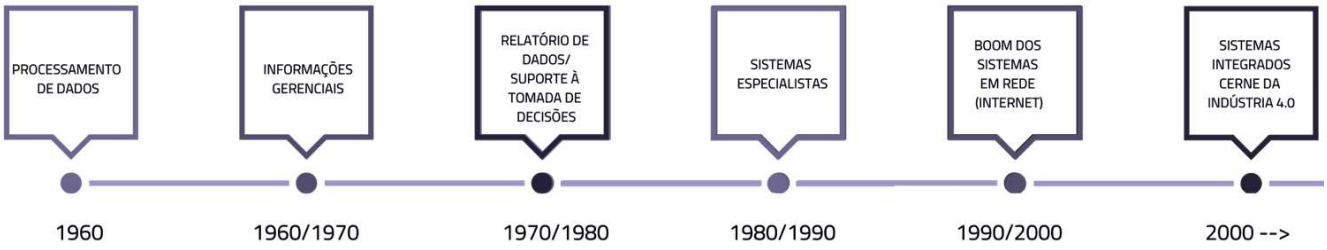
Fonte: Unitau Digital

Conforme vimos, um sistema é composto por objetivos, entradas, processos de transformação, saídas, controles, avaliações e retroalimentação.

Você consegue identificar esses elementos em algum sistema de seu uso cotidiano?

3.3 Sistemas de Informação e Organizações

Os Sistemas de Informação também passaram por uma evolução, durante a qual cada período estava associado a uma ênfase tecnológica para as organizações. A linha do tempo a seguir sintetiza os marcos dessa evolução:

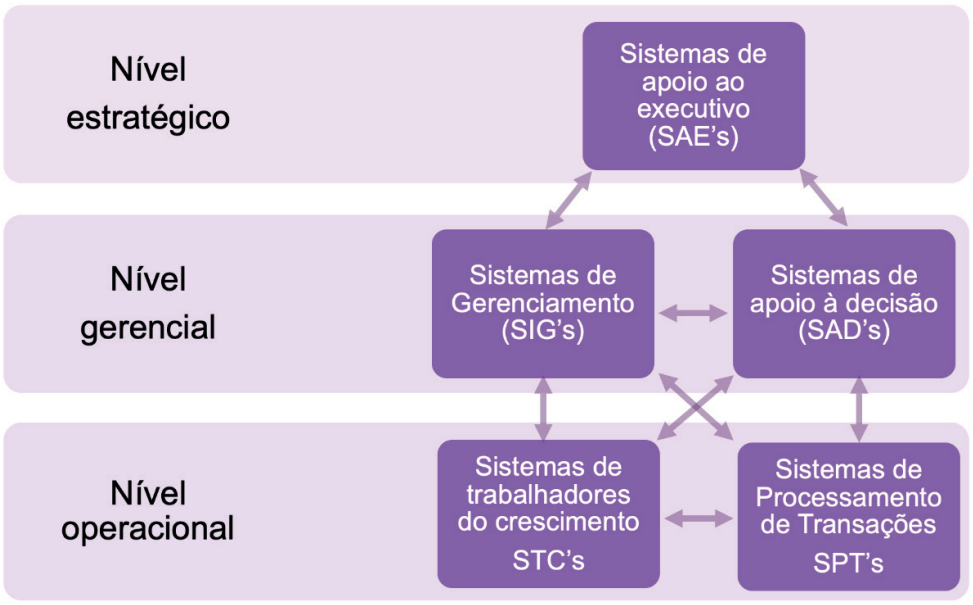


Os Sistemas de Informação podem ser classificados a partir da observação de sua própria estrutura organizacional, ou por níveis organizacionais, áreas funcionais principais, tipos de suporte que proporcionam e quanto à arquitetura da informação, trabalhando sempre com os componentes principais: hardware, *software*, dados, procedimentos e pessoas.

Considerando a dimensão hierárquica de uma organização, para cada nível de gestão organizacional estão relacionados alguns Sistemas de Informação, também relacionados com o tipo de problema ou o nível de tomada de decisão. É o que podemos observar na imagem a seguir, caracterizada como Pirâmide dos Sistemas.

Fonte: Unitau Digital

Os níveis de suporte também são comumente divididos em Nível Operacional (Sistemas Transacionais), Nível Gerencial (Sistemas de Informações Gerenciais) e Nível Estratégico (Sistemas de Apoio à Decisão e Sistemas de Apoio ao Executivo), como sintetizado na figura a seguir:



Fonte: Unitau Digital

As empresas precisam estar preparadas para resolver os problemas internos e externos. Para isso, devem buscar soluções nos sistemas de informação. Nesse contexto, as organizações se remodelaram e investiram em recursos associados à informação.

A tomada de decisões não é um processo linear e pré-estruturado, pois muitos são os elementos que entram em jogo quando é preciso tomar decisões nas organizações. Por isso, o emprego de sistemas de informação é tão importante para uma gestão eficiente e otimizada.

Hoje, temos sistemas que potencializam a capacidade organizacional de adquirir, guardar e disseminar conhecimentos, recorrendo a técnicas de extração de informação e de dados (*data mining*) e a programas inteligentes, como forma de extrair o conhecimento dos dados e a informação disponível na organização. A tecnologia levou, então, a guardar e a disseminar conhecimentos de formas mais eficazes e eficientes, atuando com a criação de modelos ou de mapas mentais, usando filtros de informação, o que aumenta o valor, a eficácia e a eficiência da organização. Essas relações estão esquematizadas na imagem a seguir:

Fonte: Unitau Digital

Você sabe identificar os ERP´s mais bem-conceituados do mercado? Conhece o Quadrante Mágico do Gardner Group? Que tal pesquisar sobre esse conceito?

3.4 Síntese da Unidade

Nesta unidade, você aprendeu que a Tecnologia de Informação (TI) é definida como todas as atividades e soluções desempenhadas por recursos computacionais. Conversamos sobre o fato de que as organizações, via de regra, têm investido cada vez mais em Tecnologias de Informação, o que levou ao surgimento do Departamento de TI, que pode ser subdividido em áreas, como: Governança de TI, TI Operacional e Infraestrutura de TI.

Enfatizamos que a informação é um dos principais ativos de uma organização. Assim, os dados organizacionais ganham destaque, pois, após serem manipulados, comparados, ordenados e interpretados, são transformados em informações relevantes e conhecimentos cruciais para as tomadas de decisão organizacionais.

Além disso, vimos que os Sistemas de Informação (SI) visam à compreensão e à análise do uso das Tecnologias de Informação nos processos gerenciais e administrativos de uma organização, e definimos sistema como um conjunto de componentes e subsistemas que formam um todo, podendo ser classificado, em TI, como aberto, caso ocorra interação com o meio exterior, ou fechado, caso dialogue apenas com o universo interno à organização.

Um Sistema de Informação visa reduzir as restrições no suporte do fluxo de dados, evitando que informações sejam duplicadas pelos subsistemas, gerando redundâncias e erros, minimizando, dessa forma, os esforços das equipes e permitindo o bom funcionamento do sistema como um todo.

Também conversamos sobre o fato de que os Sistemas de Informação podem ser classificados a partir da observação da estrutura organizacional ou por níveis organizacionais, áreas funcionais principais e tipos de suporte que proporcionam, e quanto à arquitetura da informação, trabalhando sempre com os componentes principais: *hardware*, *software*, dados, procedimentos e pessoas.

Considerando a dimensão hierárquica de uma organização, para cada nível de gestão organizacional, vimos que estão relacionados alguns Sistemas de Informação, também relacionados ao tipo de problema ou ao nível de tomada de decisão. Os níveis de suporte também são comumente divididos em Nível Operacional (Sistemas Transacionais), Nível Gerencial, ou tático (Sistemas de Informações Gerenciais) e Nível estratégico (Sistemas de Apoio à Decisão e Sistemas de Apoio Executivo).

Esperamos que, ao concluir esta Unidade, você tenha compreendido a relevância dos Sistemas de Informação em âmbito organizacional.

Não deixe de acessar as indicações de material para complementar seus estudos.

3.5 Para saber mais:

Livros: Biblioteca Pearson / SIBI UNITAU

SILVA, Kátia Cilene Neles da. **Sistemas de informações gerenciais**. Porto Alegre: SAGAH, 2019.

Vídeo

Sistema de Informações Gerenciais (SIG): Conceito, Fluxograma, Importância e Benefícios: Disponível em: https.//www.youtube.com/watch?v=xBJlHnpzDoo. Acesso em mar., 2021.

Unidade IV
Funções, sub-rotinas e recursividade

Nesta Unidade, você estudará os conceitos da programação estruturada, a definição das sub-rotinas e as definições das funções. Poderá, também, observar a construção de funções e as sub-rotinas em linguagem, além de discutir sobre os conceitos da recursividade e de como são definidos os algoritmos recursivos e a implementação de algoritmos recursivos em linguagem C.

Introdução

Nesta Unidade, daremos início ao estudo das sub-rotinas, definidas como funções e procedimentos em programação estruturada. A partir de uma abordagem sobre modularização, estudaremos o conceito de programação *top-down*, necessário para a construção do conhecimento sobre funções e procedimentos.

Veremos como representar as funções e os procedimentos com e sem passagem de valores nos algoritmos e na linguagem C, além de funções com passagem por referência na linguagem C. Ainda pertencendo a esse escopo, veremos a definição de variáveis locais e globais utilizadas quando representamos funções e procedimentos.

Na sequência, abordaremos as representações de funções recursivas nos algoritmos e na linguagem C, funções essas que são chamadas dentro delas mesmas, utilizadas em problemas com características recursivas, problemas com estruturas autocontidas.

Esperamos que, ao final desta Unidade, você seja capaz de iniciar o trabalho de resolver algoritmos e programas utilizando diversos tipos de funções e procedimentos.

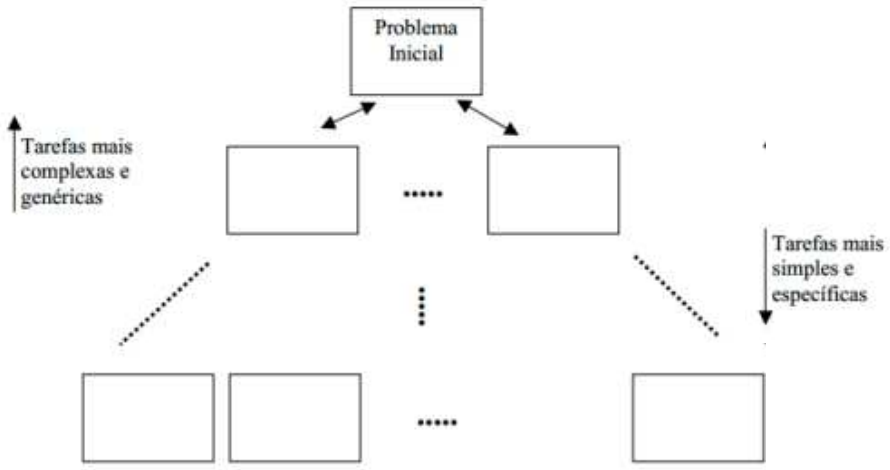
Bons estudos!

4.1 Programação top-*down* e modularização

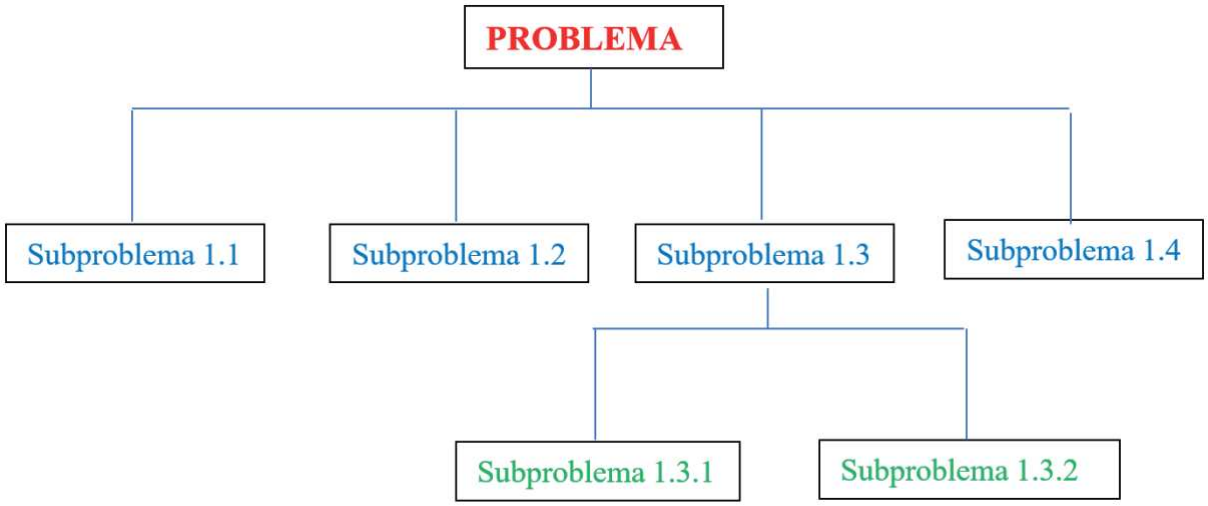
Um algoritmo, ou programa, tem como função resolver um problema específico a partir da execução de um número finito de instruções. A programação estruturada é a técnica que visa decompor um módulo de programa. Até o momento, trabalhamos com o princípio da programação estruturada para pensar o desenvolvimento em estruturas sequenciais, de seleção e de repetição. Entretanto, para ampliar nosso campo de aplicações, veremos como trabalhar com módulos. Um dos métodos usados nesse tipo de desenvolvimento é a programação ***top-down***.

Essa metodologia permite decompor um problema em partes menores, bem definidas e com mecanismos de interação claros e bem delimitados, em que o desenvolvimento do algoritmo, ou programa, é feito implementando-se cada uma das partes.

Em determinada fase do desenvolvimento, principalmente para programas maiores, faz-se necessário que ocorra um processo de decomposição em problemas menores, ou seja, com mais facilidade de desenvolvimento. Essa técnica pode ser aplicada diversas vezes, partindo-se do problema inicial até que se obtenham tarefas mais fáceis e menores, que podem ser implementadas como sub-rotinas. A figura a seguir exemplifica a decomposição.



A programação ***top-down*** é uma das técnicas de modularização cujo fluxo parte do geral para o específico.



Uma programação com o método ***top-down*** é mais usada no início da modelagem do problema, servindo de parâmetro para definir um escopo para o desenvolvimento, apoiando a estimativa de custo do projeto.

Podemos considerar a técnica ***top-down*** como um refinamento do problema, e que, em um nível de abstração, destaca as características mais importantes do problema e subdivide o sistema inicial em problemas menores. Quanto mais se subdivide o problema, mais simples se torna sua implementação.

Na prática, conforme o refinamento ocorre, pode ocorrer a necessidade da revisão dos níveis superiores.

Cada módulo pode ser desenvolvido de forma separada e independente, podendo, em algum momento, ser integrado ao problema principal.

Nas linguagens de alto nível, podemos fazer a modularização com a criação de funções e procedimentos

Para finalizar este tópico, importa frisar que essa técnica permite que os programas sejam mais fáceis de escrever e de ler com códigos mais curtos e com maior facilidade de alteração durante o desenvolvimento e durante manutenções futuras.

4.2 Representação de funções em algoritmos e linguagem C

Como foi visto no tópico anterior, uma modularização pode ser feita por funções e por procedimentos.

Os procedimentos são rotinas, ou trechos de código, capazes de executar uma tarefa. Nos algoritmos, também são chamados de subprograma, sub-rotina ou módulo. Eles são chamados dentro do corpo do programa principal, da mesma forma que os comandos, e, ao término de sua execução, o fluxo da execução continua a partir do ponto em que foi chamado.

A seguir, veremos um exemplo de procedimento no pseudocódigo.

```
1  algoritmo "Exemplo Procedimento"
2  var
3    n,m, res :inteiro
4    procedimento soma
5    var aux: inteiro
6    inicio
7      aux <- n + m
8      res <- aux
9    fimprocedimento
10 inicio
11   escreval("Este algoritmo mostra um exemplo de procedimento")
12 n <- 5
13 m <- -10
14 soma
15 escreva(res)
16 fimalgoritmo
17
```

O algoritmo é iniciado com a declaração das variáveis **n**, **m**, e **res** (chamadas de variáveis globais), e, a seguir, o procedimento **soma** é definido, iniciando com a palavra **procedimento,** seguido por um nome, que será referenciado no programa principal. Devemos observar que o formato de um procedimento é muito semelhante ao programa principal.

Foi declarada a variável **aux** do tipo inteiro no procedimento. Essa variável é chamada variável local, pois só é reconhecida pelo procedimento.

O procedimento também tem um **início** e o **fimprocedimento**.

No programa principal, a chamada do procedimento é feita pela instrução **soma** (linha 14), e então o fluxo de execução é desviado para o procedimento (linha 4), em que as instruções serão executadas, retornando para o programa principal (linha 15).

Devemos reforçar que as variáveis globais puderam ser usadas no procedimento e no algoritmo principal, porém as variáveis locais puderam ser usadas apenas no pseudocódigo.

Podemos realizar também um procedimento com parâmetros, como mostra o exemplo a seguir:

```
1  algoritmo "Exemplo Procedimento"
2  var
3    n,m, res :inteiro
4    procedimento soma (x,y : inteiro)
5    var
6    inicio
7       res <- x + y
8    fimprocedimento
9  inicio
10   escreval("Este algoritmo mostra um exemplo de procedimento")
11   Escreval ("Com parâmetros")
12 n <- 5
13 m <- -10
14 soma (m, n)
15 escreva(res)
16 fimalgoritmo
17
```

O pseudocódigo se inicia com a declaração das variáveis globais e, em seguida, o procedimento **soma**, agora com a passagem dos parâmetros **x** e **y** do tipo inteiro. Quando o procedimento **soma** é invocado, leva os conteúdos das variáveis **m** e **n** como parâmetros. Uma cópia desses valores é atribuída às variáveis **x** e **y**, que usa esses valores no procedimento.

Essa passagem de parâmetros do exemplo acima se chama passagem por valor, e a quantidade dos parâmetros, sua sequência e respectivos tipos não podem mudar, ou seja, devem estar de acordo com o que foi especificado na sua declaração.

Além dos procedimentos, teremos também as funções, cuja diferença em relação aos procedimentos é que funções retornam um valor para o pseudocódigo principal.

A declaração de funções é feita de modo análogo aos procedimentos. No exemplo a seguir, são declaradas as variáveis globais **n** e **m** do tipo inteiro. A declaração da função **soma** feita na sequência tem os parâmetros **x** e **y** definidos como inteiros. Além disso, devemos observar que a própria função **soma** também tem a declaração do seu tipo correspondente ao que retornará no pseudocódigo principal.

```
1  algoritmo "Exemplo Função"
2  var
3    n,m :inteiro
4    funcao soma (x,y: inteiro ): inteiro
5    var
6      res : inteiro
7    inicio
8        res <- x + y
9      retorne res
10   fimfuncao
11 inicio
12   escreval("Este algoritmo mostra um exemplo de função")
13 n <- 5
14 m <- -10
15 escreva("Resposta: ",soma(n, m))
16 fimalgoritmo
17
```

A variável **res** declarada como inteira é local, tendo valor somente na função. Devemos observar a instrução **retorne**, responsável por levar o valor da variável para o pseudocódigo principal.

No pseudocódigo principal, a função **soma** é invocada com os parâmetros **n** e **m**.

Na linguagem C, um procedimento é visto como uma função que não retorna valor.

Para exemplificar, veremos o uso do procedimento em um programa em linguagem C, que cria um procedimento chamado **pmedia**. O procedimento deve ser declarado logo após a diretiva **include**. Devemos observar que a chamada do procedimento é feita na função **main** utilizando os parênteses após o nome do procedimento.

Você sabia que o escopo de uma variável se refere ao local em que ela pode ser usada? Uma variável definida dentro de uma função só pode ser usada dentro desta função (não pode ser usada fora do escopo da função). Portanto, podemos definir variáveis com o mesmo nome em funções diferentes sem nenhum problema. Entretanto, as variáveis que são definidas dentro de uma função são locais a essa função, ou seja, elas só existem enquanto a função está sendo executada (elas passam a existir quando ocorre a entrada da função e são destruídas ao sair).

```c
#include<stdio.h>
float pmedia();
int main()
{
  pmedia();
}

float pmedia()//Procedimento
{
      float n1, n2, m;

      printf("Digite a nota 1 do aluno: ");
      scanf("%f", &n1);
      printf("Digite a nota 2 do aluno: ");
      scanf("%f", &n2);
      m = (n1 + n2)/2;
      printf("\n\nA media do aluno foi %6.2f", m);
}

```

Da mesma forma vista nos pseudocódigos, os programas em linguagem C permitem as funções com passagem de parâmetros.

No exemplo a seguir, exemplificamos um programa em linguagem C com uma função com passagem de parâmetros por valor.

```c
#include<stdio.h>
float fmedia(float n1, float n2)//Função
{
      float m;
      m = (n1 + n2)/2;
      return m;
}

int main()
{
  float nota1, nota2;
  printf("\ndigite as duas notas\n");
  scanf("%f",&nota1);
  scanf("%f",&nota2);
  printf("\n Média = %f",fmedia(nota1, nota2));

}

```

No programa principal, foram declaradas e lidas as variáveis **nota1** e **nota2**, que foram passadas por parâmetro na função **fmedia**. A função **fmedia** recebe os valores, atribuindo para as variáveis **n1** e **n2**. Após o cálculo da média, a instrução **return** devolve o valor da média pela variável **m**.

Nos programas em linguagem C, também é possível utilizar funções com passagem de parâmetros por referência, o que permite alterar o conteúdo da variável passada por parâmetro, que, nesse caso, exige passar o endereço da variável.

A seguir, um exemplo do uso de passagem de parâmetro por referência em linguagem C.

```c
#include <stdio.h>

float altera(float *valornovo) // Define que o parâmetro é uma referência à outra variável
{
    *valornovo = 40.5; // utiliza o operador para alterar o conteúdo da variável
}

int main()
{
   float valor;

    valor = 50.4;
    altera(&valor);  // Passa o endereço da variável valor para a função
    printf("%f", valor); // o valor impresso será 40.5 alterado na função
}
```

Na função principal **main**, a variável **valor** recebe o valor 50.4, e, na sequência, é invocada a função **altera**, levando a variável **valor** como parâmetro, precedida pelo operador **&**, que faz referência ao endereço da variável, e não ao seu conteúdo.

Na função **altera**, a variável que chega como parâmetro é precedida pelo operador *, fazendo referência a outra variável.

Para permitir a alteração da variável usada como parâmetro, é preciso passar o **endereço** da variável, com operador * é usado novamente no corpo da função, caracterizando, dessa forma, uma **passagem por referência**.

Até aqui, aprendemos as diversas formas de representação de funções e procedimentos nos algoritmos e programas em linguagem C. Agora, veremos a representação de funções recursivas em algoritmos e linguagem C.

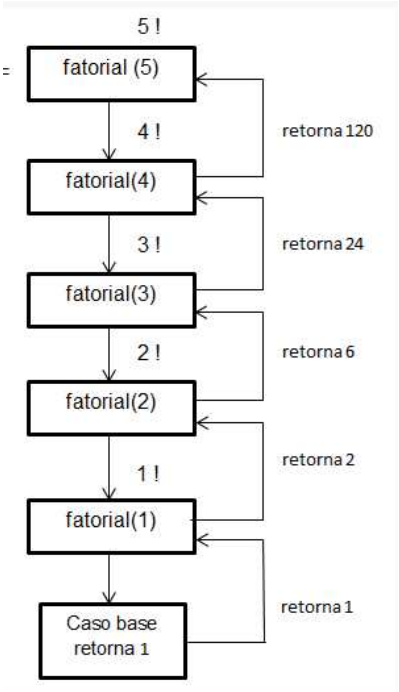
4.3 Representação de funções recursivas em algoritmos e linguagem C

Neste tópico, veremos um tipo especial de funções, chamadas de funções recursivas, ou seja, funções que chamam a si próprias. A recursividade proporciona um desenvolvimento mais preciso e claro, sendo uma técnica muito produtiva para problemas que se adaptam a essa característica.

Você sabia que quando uma função é chamada recursivamente, cria-se um ambiente local para cada chamada. As variáveis locais de chamadas recursivas são independentes entre si, como se estivéssemos chamando funções diferentes.

Para exemplificar, veremos o problema que resolve o fatorial de um número inteiro.

O fatorial de um número n é denotado por n! e calculado da seguinte forma:



$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot (n-1)! & \text{se } n > 0 \end{cases}$$

Podemos afirmar que o fatorial de um número inteiro é a multiplicação desse mesmo número pelo fatorial de seu antecessor, como mostra a figura. Esse exemplo justifica a recursividade que será aplicada a seguir.

```
1  algoritmo "Exemplo Função Recursiva"
2  var
3    nt :inteiro
4    funcao fatorial (v: inteiro): inteiro
5  inicio
6  se v <= 2 entao
7  retorne v
8  senao
9  retorne v * fatorial(v-1)
10 fimse
11 fimfuncao
12  inicio
13   escreval("Este algoritmo mostra um exemplo de função recursiva Fatorial")
14   escreval ("Forneça o número para o cálculo do fatorial")
15   leia (nt)
16  escreva("Fatorial: ",fatorial(nt))
17  fimalgoritmo
18
```

Iniciamos este estudo declarando a variável global **nt** do tipo inteiro para representar o número do qual queremos calcular o fatorial e que é levado como parâmetro na chamada da função **fatorial**.

A função se inicia verificando se o valor do fatorial desejado é dois e, nesse caso, já retorna esse valor, pois 2*1 = 2. Caso contrário, ou seja, se o valor desejado for maior que dois, teremos esse valor multiplicado pela própria função, tendo como parâmetro um valor a menos. Dessa forma ocorre a recursividade, quando a função é invocada dentro dela mesma.

Esse processo se repete até que o valor seja menor ou igual a dois.

Veremos, a seguir, a representação do mesmo problema em linguagem C.

```c
//Cálculo de fatorial com função recursiva
#include <stdio.h>
int fatorial(int n);
int main()
{
  int numero;
  int f;
  printf("Digite o numero que deseja calcular o fatorial: ");
  scanf("%d",&numero);
  //chamada da função fatorial
  f = fatorial(numero);
  printf("Fatorial de %d = %d",numero,f);
}
//Função recursiva que calcula o fatorial
//de um numero inteiro n
int fatorial(int n)
{
  int vfat;
  if ( n <= 2 )
    //Caso base: fatorial de n <= 1 retorna 1
    return (n);
  else
  {
    //Chamada recursiva
    vfat = n * fatorial(n - 1);
    return (vfat);
  }
}
```

Como podemos observar, a mesma lógica de resolução foi utilizada. A função principal **main** invoca a função recursiva **fatorial**, levando o valor o qual se deseja calcular o fatorial.

4.4 Síntese da Unidade

Nesta Unidade, vimos o conceito de modularização e programação ***top-down*** como base para o aprendizado de funções e procedimentos.

Em consequência desses conceitos, pudemos desenvolver algoritmos e programas com resolução, utilizando funções e procedimentos.

Também vimos que as funções apresentam diversas representações, como funções com passagem de valor e passagem de referência.

Além disso, você aprendeu que as variáveis podem ter declaração local, tendo efeito apenas dentro de uma função ou procedimento e declaração global. Nesse caso, com efeito em todo o código.

Por fim, vimos as funções recursivas, que contam com as chamadas na própria função e que podem ser aplicadas a problemas que possibilitam a recursividade, mantendo mais clareza no desenvolvimento.

Esperamos que os saberes apresentados tenham ajudado você a compreender os diversos recursos aplicados a procedimentos e funções, que continuarão a ser aplicados nas próximas Unidades desta disciplina.

4.5 Para Saber Mais

Livros

ARAÚJO, Sandro de. **Lógica de programação e algoritmos**. Curitiba: Ed. Contentus, 2020. Capítulo 6.

Unidade V
Vetores e Matrizes

Nesta Unidade, você estudará os conceitos da programação estruturada, a definição das sub-rotinas e as definições das funções. Poderá, também, observar a construção de funções e as sub-rotinas em linguagem, além de discutir sobre os conceitos da recursividade e de como são definidos os algoritmos recursivos e a implementação de algoritmos recursivos em linguagem C.

Introdução

Nesta Unidade, daremos início ao estudo das variáveis indexadas. A partir desse conceito, veremos as representações de vetores e matrizes nos algoritmos e na linguagem de programação C que conseguem armazenar vários valores de um mesmo tipo em uma variável indexada.

Veremos, ainda, como representar os vetores, que são representações unidimensionais, muito semelhantes às matrizes do tipo linha ou coluna da matemática.

Na sequência, estudaremos as representações de matrizes, com representações multidimensionais, sendo que as mais utilizadas são as matrizes bidimensionais.

Esperamos que, ao final desta Unidade, você seja capaz de iniciar o trabalho de resolver algoritmos e programas utilizando os vetores e matrizes, que possuem um amplo campo de aplicações.

Bons estudos!!!

5.1 Variáveis Indexadas, Vetores e Matrizes na Computação

As variáveis indexadas são utilizadas em arranjos, os quais são constituídos por uma estrutura de dados que permite o armazenamento de um conjunto de dados do mesmo tipo. Ou seja, trata-se de uma coleção de elementos do mesmo tipo, em que a posição de cada elemento está definida de forma única por um valor do tipo inteiro. As variáveis indexadas são representadas por:

a) Arranjos unidimensionais – têm apenas um índice;
b) Arranjos multidimensionais – têm dois ou mais índices.

Podemos definir, então, uma variável indexada como o conjunto de valores do mesmo tipo, que utilizam o mesmo nome, referenciados por uma posição ou índice.

Para representar computacionalmente os arranjos unidimensionais, usamos um **vetor**, que é uma estrutura de dados que armazena os itens de dados do mesmo tipo. Por exemplo, para armazenar quatro valores inteiros na memória, precisamos declarar quatro variáveis inteiras, sendo uma em cada valor.

Um vetor pode ser usado para resolver essa situação. Nessa estrutura, informamos que usaremos uma variável indexada do tipo inteiro com capacidade para armazenar quatro valores. Podemos relacionar os vetores com as matrizes unidimensionais da matemática.

Quando declaramos um vetor, informamos um nome para essa variável indexada, seu tamanho (ou número de elementos que ela irá manipular) e um tipo.

Para exemplificar, vamos considerar uma variável **A** com capacidade para manipular 3 elementos inteiros, conforme a representação a seguir:

$$\begin{matrix} A(1) = 10 \\ A(2) = 20 \\ A(3) = 30 \end{matrix} \quad \text{ou } A = \begin{Bmatrix} 10 \\ 20 \\ 30 \end{Bmatrix}$$

A variável A terá o valor 10 na posição 1, o valor 20 na posição 2, e o valor 30 na posição 3. Portanto, para trabalhar com esse tipo de variável, sempre teremos de declarar uma variável inteira para indicar a posição do elemento do vetor. Analogamente, como vemos no estudo da Matemática, um vetor realiza a representação de uma matriz unidimensional, não fazendo distinção em linha ou coluna.

Como toda variável computacional, um vetor deve ser declarado, e será organizado, por uma estrutura de repetição para leitura, escrita e manipulação de seus elementos.

Uma variável indexada também pode ser representada por um arranjo multidimensional, sendo o caso mais comum o uso de matrizes. A representação de matrizes também é análoga à que aprendemos na Matemática, ou seja, manipulamos matrizes nos referindo à posição dos elementos por linhas e colunas.

Dessa forma, precisamos de duas variáveis inteiras para indicar a posição do elemento em relação à linha e à coluna.

Para exemplificar, vamos considerar uma variável B com capacidade para manipular 4 elementos inteiros, conforme a representação a seguir:

B(1,1) = 10 B(1,2) = 30
B(2,1) = 20 B(2,2) = 40

$$B = \begin{matrix} 10 & 30 \\ 20 & 40 \end{matrix}$$

Conforme a representação, a variável B terá o valor 10 na linha 1 e na coluna 1, o valor 30 na linha 1 e na coluna 2, o valor 20 na linha 2 e na coluna 1, e o valor 40 na linha 2 e na coluna 2.

Como toda variável computacional, uma matriz deve ser declarada e será manipulada por uma estrutura de repetição para leitura, escrita e manipulação de seus elementos.

Tanto os vetores como as matrizes apresentam uma grande variedade de aplicações, como na área de computação gráfica, nos problemas matemáticos de álgebra linear, simulações numéricas e outras.

Agora, aprenderemos sobre a representação de vetores e matrizes nos algoritmos e em programas em linguagem nos próximos tópicos.

5.2 Representação de Vetores em Algoritmos e Linguagem C

Nesse Tópico, veremos como utilizar os vetores nos algoritmos e nos programas em Linguagem C. Inicialmente, veremos como realizar a declaração de um vetor e a leitura dos elementos em um algoritmo. No exemplo a seguir, será declarada a variável indexada v com capacidade de 5 elementos inteiros.

Para essa declaração, usamos o nome da variável seguido de dois pontos, e a palavra vetor, indicando que se trata de uma variável indexada. Entre colchetes, indicamos a dimensão do vetor, usando uma notação de valor inicial dois pontos valor final e, por último, o tipo dos elementos que comporão o vetor.

Área dos algoritmos (Edição do código fonte) -> Nome do arquivo: [EXEMPLO VE

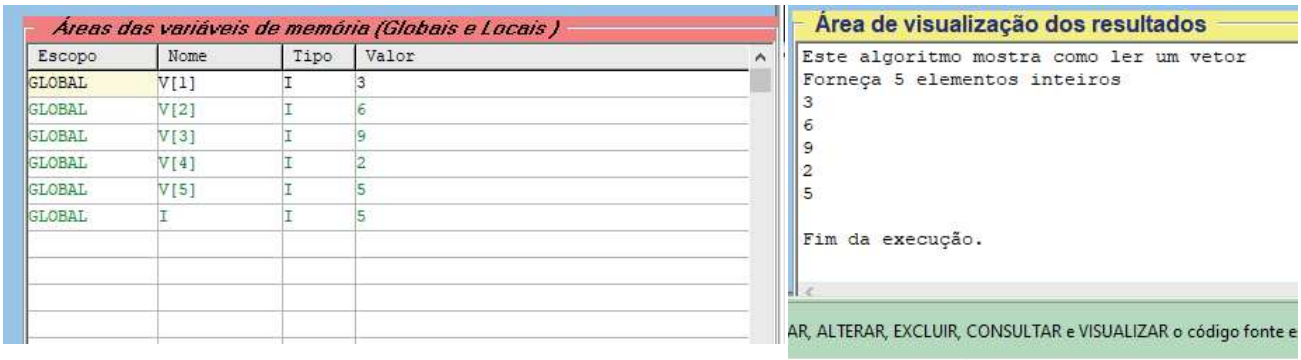
```
1  algoritmo "Exemplo Vetores"
2  var
3     v: vetor[1..5] de inteiro
4     i:inteiro
5
6  inicio
7    escreval("Este algoritmo mostra como ler um vetor")
8    escreval ("Forneça 5 elementos inteiros")
9    para i de 1 ate 5 faca
10      leia (v[i])
11   fimpara
12 fimalgoritmo
13
```

Também foi declarada a variável inteira i, que apresentará o índice ou a posição do elemento dentro do vetor.

Após as mensagens informativa e solicitante, temos uma estrutura de repetição para – faca que manipula a variável i com 5 repetições. Para cada repetição, o comando leia é executado e aguarda a digitação de um valor que será armazenado na variável v, posição i.

Devemos observar que, nos algoritmos, iniciamos um vetor com a primeira posição considerada como 1, e o índice sempre estará entre colchetes.

A seguir, veremos a área de execução e a área de simulação de memória.



Áreas das variáveis de memória (Globais e Locais)

Escopo	Nome	Tipo	Valor
GLOBAL	V[1]	I	3
GLOBAL	V[2]	I	6
GLOBAL	V[3]	I	9
GLOBAL	V[4]	I	2
GLOBAL	V[5]	I	5
GLOBAL	I	I	5

Área de visualização dos resultados

```
Este algoritmo mostra como ler um vetor
Forneça 5 elementos inteiros
3
6
9
2
5

Fim da execução.
```

AR, ALTERAR, EXCLUIR, CONSULTAR e VISUALIZAR o código fonte e

Na área de visualização de resultados, vemos os valores digitados durante a execução. Na área das variáveis de memória, podemos observar que cada valor digitado é atribuído à variável v, na posição i.

A reprodução de um vetor na execução também ocorre dentro de uma estrutura de repetição. Veremos a seguir um exemplo mais completo.

Área dos algoritmos (Edição do código fonte) -> Nome do arquivo

```
1  algoritmo  "Soma 10 elementos - vetor"
2  var
3  num:vetor[1..10] de real
4  soma : real
5  i : inteiro
6  inicio
7      soma <- 0
8      escreval("Algoritmo que soma 10 números")
9      escreval("Digite 10 números")
10     para  i de 1 ate 10 faca
11        leia (num[i])
12        soma <- soma + num[i]
13     fimpara
14     escreval ("Soma = ", soma)
15 fimalgoritmo
```

Iniciamos declarando a variável indexada **num** com capacidade para armazenar 10 elementos do tipo real, a variável **soma** que irá acumular a soma dos elementos lidos e a variável **i** que será a variável de controle da estrutura **para – faca** e será responsável por indicar a posição do elemento no vetor **num**. Para cada valor digitado e armazenado no vetor **num**, a soma é realizada. Ao final das iterações, a soma é mostrada, ou seja, a resposta é uma variável e não um vetor

Você sabia que na Linguagem de programação C podemos utilizar os operadores compostos de atribuição +=, -=, *= e /= como atalhos para as operações aritméticas mais simples. Veja os exemplos com a variável cont:

cont += 1 equivale a cont = cont +1

cont -= 1 equivale a cont = cont -1

cont *= 1 equivale a cont = cont *1

cont /= 1 equivale a cont = cont /1

No próximo exemplo, será calculada a média de um aluno, mostrando na resposta um vetor.

Área dos algoritmos (Edição do código fonte) -> Nome do arquivo: [EXEM

```
1  algoritmo  "Média"
2  Var
3  nota : vetor [1..3] de real
4  soma, media : real
5  i : inteiro
6  Inicio
7  soma <- 0
8  escreval ("Digite as três notas do aluno: ")
9
10 para i de 1 ate 3 faca
11    leia (nota[i])
12    soma <- soma + nota[i]
13 fimpara
14
15 media <- soma / 3
16 escreval ("A média é ", media)
17
18 para i de 1 ate 3 faca
19    escreval ("Nota ", i + 1, ": ", nota[i])
20 fimpara
21 Fimalgoritmo
```

A variável indexada **nota** armazena as 3 notas na primeira repetição e acumula o valor digitado na variável **soma**. A média é calculada e mostrada. Na segunda repetição, as notas são mostradas.

Veremos, na sequência, a representação de vetores em Linguagem C, vetores esses que seguem a mesma lógica computacional dos algoritmos.

No exemplo a seguir, inicialmente é definida a constante **LIM,** com valor 3, que corresponde ao dimensionamento do vetor.

São declaradas as variáveis **notas**, **soma** e **media**, como **float** e a variável **int i**. A variável **notas** é o vetor com dimensão que corresponde ao valor de **LIM**, ou seja, 3.

Na primeira estrutura de repetição **for**, as notas são digitadas e acumuladas na variável **soma**. A média é calculada e, na segunda estrutura de repetição, cada nota é comparada com o valor da média; caso seja superior, é mostrada na mensagem.

```c
#include <stdio.h>
#define LIM 3
int main()
{
float notas[LIM],soma=0.0, media;
int i;
for (i=1; i<LIM; i++)
{
printf("Digite a nota do aluno %d: ",i);
scanf("%f",&notas[i]);
soma+=notas[i];
}
media = soma/LIM;
printf("Media das notas: %.2f e \n",media);
for (i=1; i<LIM; i++) {
if (notas[i]>= media){
printf("\n O aluno %d tem nota %.2f maior que a média.",i,notas[i]);
}
}
}
```

No próximo exemplo, alimentamos um vetor com 5 elementos do tipo **float** e lemos um número para pesquisar se ele pertence ao vetor.

Na segunda estrutura de repetição, a pesquisa é realizada e, caso o valor seja encontrado, é emitida uma mensagem com o número e sua posição.

```c
#include <stdio.h>
int main()
{
float num[5], pesq;
int i;
printf("\n Verifica se um número pertence ao vetor\n");
printf("\n Forneça 5 números para o vetor \n");
for (i=0; i<5; i++)
{
scanf("%f",&num[i]);
}
printf("Digite o número que quer pesquisar");
scanf("%f", &pesq);
for (i=0; i<5; i++) {
if (num[i] == pesq){
printf("\n O número %f está na posição %d do vetor.",num[i], i);
}
}
}
```

Veremos uma aplicação do uso de vetores com o método da Bolha de Classificação, ou *Buble Sort,* que consiste em um método de ordenação. Nessa resolução, cada um dos elementos do vetor é comparado com seus vizinhos e, caso estejam fora de ordem, trocam de posição entre si. Esse processo se repete até que o vetor esteja ordenado.

A variável **j** da resolução é responsável por garantir que todos os elementos do vetor sejam comparados com os demais, esse controle é feito na estrutura de repetição **while**.

A estrutura de repetição **for** dentro da estrutura **while** faz a varredura comparando os elementos e trocando-os de lugar quando necessário.

```c
#include <stdio.h>
int main()
{
float numeros[10];
int i, j;
float aux;
printf("\n Buble sort\n");
for (i=0;i<10;i++) {
   scanf("%f", &numeros[i]);
}
j=10;
while (j>1)
{
   for (i=0; i< j-1; i++) {
       if(numeros[i] > numeros [i+1])
       {
        aux = numeros[i];
        numeros[i] = numeros[i+1];
        numeros[i+1] = aux;
       }
    }
    j = j-1;
}
printf("\n Vetor Ordenado\n");
for (i=0;i<10;i++) {
   printf("\n %f", numeros[i]);
}
}
```

Agora, finalizaremos esta Unidade aprendendo como os vetores são utilizados nos algoritmos e na linguagem de programação C.

5.3 Representação de Matrizes em Algoritmos e Linguagem C

Neste Tópico, veremos as representações de matrizes em algoritmos e na linguagem de programação C.

As matrizes trabalham de forma análoga aos vetores, porém com dois índices, sendo um para a posição em linha e um para a posição em coluna.

O exemplo a seguir mostra um algoritmo que lê e escreve uma matriz. Começamos declarando a variável indexada **mat1**, com dimensão de duas linhas e duas colunas. Foram declaradas também duas variáveis de controle **i** e **j**.

Para a manipulação dos elementos da matriz, precisamos de duas estruturas de controle, uma para percorrer o posicionamento da linha e uma para percorrer os elementos da coluna. A primeira estrutura fixa a linha 1 com a variável **i**, e a segunda estrutura percorre as colunas com a variável **j**. Somente após a variável **j** atingir seu valor 2, ou segunda coluna, o fluxo retorna para atualizar a variável **i**, passando a ter o valor 2, ou segunda linha. A variável **j** inicia novamente com o valor 1, ou seja, estamos na linha 2, coluna 1. O fluxo se repete até que a matriz esteja com todos os quatro elementos.

A mesma lógica se repete para mostrar os elementos da matriz.

```
1  algoritmo  "Le escreve matriz"
2  var
3  //Declaração do vetor
4  mat1 : vetor [1..2,1..2] de inteiro
5  i, j : inteiro
6  inicio
7   escreval ("Algoritmo que le e mostra uma matriz: ")
8   escreval ("Digite os valores da matriz: ")
9   para i de 1 ate 2 faca
10    para j de 1 ate 2 faca
11      escreva( "Entre com o elemento [", i , ",", j , "]: ")
12      leia( mat1[i, j] )
13    fimpara
14  fimpara
15  escreval ("Mostra a matriz: ")
16  para i de 1 ate 2 faca
17    para j de 1 ate 2 faca
18      escreval( "   Elemento [", i , ",", j , "]: ", mat1[i,j])
19    fimpara
20  fimpara
21 Fimalgoritmo
22
```

No próximo exemplo, mostraremos que podemos manipular os índices da matriz para atribuir o valor zero para os elementos da diagonal principal (quando **i** igual a **j**) e o valor um para os demais elementos.

```
algoritmo  "Zeros e uns - matriz"
var
//Declaração do vetor
mat1 : vetor [1..3,1..3] de inteiro
i, j : inteiro
inicio
 escreval ("Algoritmo que mostra uma matriz com zeros e uns: ")
 escreval ("Cria e Mostra a matriz: ")
 para i de 1 ate 3 faca
   para j de 1 ate 3 faca
     se (i = j) entao
        mat1[i,j] <-0
     senão
        mat1 [i,j] <- 1
     fimse
     escreval( "   Elemento [", i , ",", j , "]: ", mat1[i,j])
   fimpara
 fimpara
Fimalgoritmo
```

Na linguagem de programação C, o uso de matrizes segue a mesma lógica dos algoritmos. Veremos, a seguir, o programa que lê e escreve uma matriz.

Podemos observar a notação usada para os índices, separados e englobados por colchetes.

```
#include <stdio.h>
int main()
{
float mat[4][4];
int i, j;
printf("\n Realiza a leitura da matriz\n");
for (i=0;i<4;i++) {
    for (j=0; j<4; j++){
       scanf("%f", &mat[i][j]);
    }
}

printf("\n Mostra matriz\n");
for (i=0;i<4;i++) {
    for (j=0;j<4; j++){
       printf("\n %f", mat[i][j]);
}
}
}
```

Você sabia o que é um erro de overflow? A linguagem de programação C não faz nenhum teste de verificação dos índices usados para acessar os elementos de um vetor ou de uma matriz. Para evitar resultados indesejados ou erros, atente-se aos limites definidos, não tentando incluir mais elementos que o definido, assim evitando o erro de overflow.

A seguir, veremos um programa que soma os elementos da diagonal principal de uma matriz com 3 linhas e 3 colunas com elementos do tipo int.

Após a leitura dos valores da matriz, os índices são comparados e, quando o valor das variáveis **li** e **co** são iguais, significa que o elemento pertence à diagonal principal, onde é somado.

```c
#include<stdio.h>

int main()
{
    int li,co, soma=0;
    int matriz[3][3];

    printf("Digite os valores: ");
    for(li=0; li<3; li++)
    {
        for(co=0; co<3; co++)
        {
           scanf("%d", &matriz[li][co]);
           if (li == co){
           soma=soma+matriz[li][co];
           }
        }
    }
    printf("\n\n A soma dos valores da diagonal principal: %d\n", soma);


}

```

Como foi possível compreender, neste tópico, abordamos as matrizes representadas nos algoritmos e nos programas em linguagem C.

5.4 Síntese da Unidade

Nesta Unidade, vimos o conceito de variáveis indexadas.

Em consequência desses conceitos, pudemos desenvolver algoritmos e programas com resolução utilizando vetores e matrizes.

Também vimos que os vetores são arranjos unidimensionais, que representam um conjunto de variáveis do mesmo tipo referenciadas por um nome e uma posição.

Além disso, você aprendeu que as matrizes são muito semelhantes aos vetores, com a diferenciação de usar dois índices ou mais.

Por fim, vimos que as representações de vetores e matrizes são muito semelhantes às que usamos na matemática.

Esperamos que os saberes apresentados tenham ajudado você a compreender os diversos recursos aplicados a vetores e matrizes, os quais continuarão a ser aplicados nas próximas Unidades desta disciplina.

5.5 Para Saber Mais

Livros

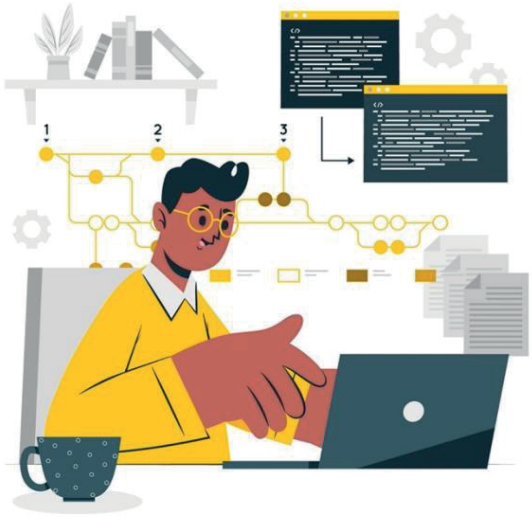
ARAÚJO, S. de. **Lógica de programação e algoritmos.** Curitiba: Ed. Contentus, 2020.

Unidade VI
Strings

Objetivo

Nesta Unidade, você terá a oportunidade de estudar sobre os conceitos de dados string e a implementação de dados string em linguagem C. A partir de funções disponibilizadas pela linguagem de programação C, você conhecerá os principais modos de manipulação de uma string e as situações em que a manipulação de strings é necessária para a construção de sistemas computacionais reais.

Introdução



Nesta Unidade, daremos início ao estudo de caracteres e de **strings**.

A representação de caracteres e da cadeia de caracteres é muito importante no estudo da lógica computacional.

Para garantir a funcionalidade de nossos programas, é necessário saber manipular caracteres e **strings** de forma a dar respostas mais compatíveis que envolvam textos.

Assim, estudaremos as formas de representação dos caracteres e **strings** nos algoritmos e nos programas em linguagem C e exploraremos as funções pré-definidas que auxiliam a manipulação desse tipo de dado.

Esperamos que, ao final desta Unidade, você seja capaz de compreender os principais conceitos e aplicações das **strings**.

Bons estudos.

6.1 Tipos Especiais de Dados

Até o momento, a maioria das aplicações estudadas por nós envolveu problemas numéricos. Nessa Unidade, daremos ênfase a outros tipos de dados e suas aplicações.

Na Unidade 1, vimos que dentre os tipos primitivos de dados temos os literais, os compostos pelos caracteres e os compostos pelo conjunto, ou cadeia, de caracteres.

Um caractere é a representação de um único digito, muito útil quando queremos trabalhar com respostas do tipo verdadeiro ou falso (V ou F), masculino ou feminino (M ou F), sim ou não (S ou N), entre outras. Os caracteres podem ser qualquer dígito, mesmo os números, porém sem a finalidade de cálculo.

Importante salientar que o caractere ‘a’ é diferente do caractere ‘A’, portanto para esse tipo de dado sempre ocorrerá ***case sensitive***.

Um conjunto, ou cadeia, de caracteres é composto por palavras, ou seja, por mais de um dígito. É útil para armazenar nomes, endereços, dentre outros dados.

Tanto os caracteres como os conjuntos de caracteres devem ser referenciados englobados por aspas ou aspas duplas dependendo da aplicação, ou seja, nos pseudocódigos, ou nas linguagens de programação, tendo cada um suas regras.

Esses tipos de dados também devem ser declarados como as outras variáveis e podem ser manipulados por todas as estruturas vistas até o momento.

As cadeias, ou conjuntos, de caracteres também são denominadas por ***strings***.

Devemos observar que um caractere é um símbolo usado para escrever um texto em determinada língua, e que cada língua utiliza um determinado número de caracteres; por exemplo, o português usa 127 caracteres, o inglês usa 94, e assim por diante. Para nomear os vários caracteres, foram atribuídos nomes numéricos a eles.

O ASCII é um código que foi proposto como uma solução para unificar a representação de caracteres alfanuméricos em computadores. Normalmente, as **strings** representam cadeias de caracteres em código ASCII.

No próximo Tópico, veremos como aplicar esses tipos de dados nos algoritmos.

6.2 Representação de *Strings* e Cadeia de Caracteres em Algoritmos

Nesse Tópico, veremos como aplicar os caracteres e as cadeias de caracteres nos algoritmos.

A declaração tanto de um caractere como de uma cadeia de caracteres no pseudocódigo é feita pelo tipo de caractere.

A seguir, um exemplo de declaração de um caractere e de uma **string**, suas leituras e exibições.

```
1 algoritmo  "exemplo declaração caractere"
2 var
3 ca1, ca2 : caractere
4 inicio
5  escreval ("Algoritmo que exemplifica uso de caracteres ")
6  escreval ("Digite um caractere ")
7  leia(ca1)
8  escreval ("Digite uma cadeia de caracteres ou string")
9  leia (ca2)
10  escreval( "Mostra o caractere -    ",ca1)
11  escreval( "Mostra a string -    ",ca2)
12 Fimalgoritmo
13
```

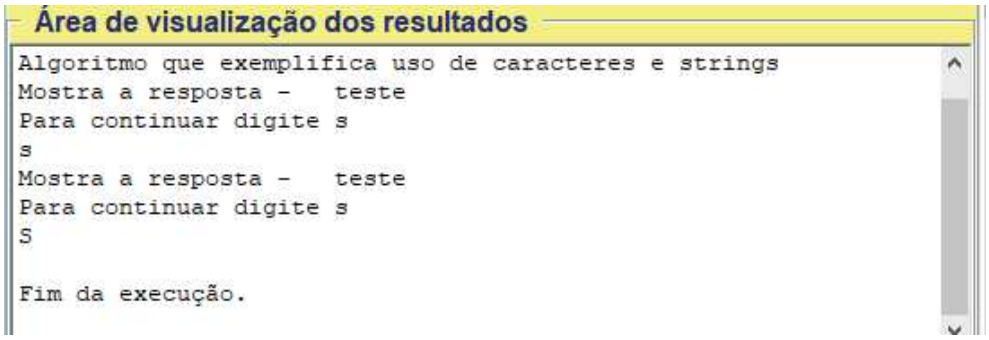
Foram declaradas as variáveis ca1 e ca2 do tipo caractere, porém o algoritmo não faz distinção se será apenas um caractere ou uma **string**.

No próximo exemplo, veremos como manipular essas variáveis.

```
1 algoritmo  "exemplo caractere e string"
2 var
3 ca1, ca2 : caractere
4 inicio
5  escreval ("Algoritmo que exemplifica uso de caracteres e strings")
6 ca1 <- "s"
7 ca2 <- "teste"
8  enquanto (ca1 = "s") faca
9     escreval( "Mostra a resposta -    ",ca2)
10     escreval( "Para continuar digite s")
11     leia (ca1)
12  fimenquanto
13 Fimalgoritmo
14
```

As variáveis ca1 e ca2 receberam valores iniciais, e devemos observar o uso de aspas duplas englobando seus conteúdos. Observando a área de visualização dos resultados a seguir, temos que

quando foi digitado o valor S em maiúsculo a execução finalizou, pois o caractere da condição se refere ao s minúsculo.



Nessa situação, uma boa prática é o uso do operador OU, como será mostrado a seguir, aceitando, dessa forma, a utilização de maiúsculas e de minúsculas.

```
 1 algoritmo  "exemplo caractere e string"
 2 var
 3 ca1, ca2 : caractere
 4 inicio
 5  escreval ("Algoritmo que exemplifica uso de caracteres e strings")
 6 ca1 <- "s"
 7 ca2 <- "teste"
 8  enquanto (ca1 = "s") OU (ca1 = "S") faca
 9     escreval( "Mostra a resposta -   ",ca2)
10     escreval( "Para continuar digite s")
11     leia (ca1)
12  fimenquanto
13 Fimalgoritmo
14
15
```

Para exemplificar, veremos um algoritmo que encontra a maior nota de um aluno, indicando também seu nome.

Na estrutura de repetição para – faca são lidas as notas e os nomes dos alunos de acordo com a quantidade de alunos informada.

Na estrutura **se-fimse**, ocorre a verificação da maior nota e, quando encontrada, também é atribuído o nome do aluno correspondente à nota.

```
1  algoritmo "Maior nota"
2  var
3  nota,maior : real
4  qtde, cont : inteiro
5  nome, nome_m : caractere
6  inicio
7     Escreval ("Mostra o aluno com maior nota")
8     Escreval ("Forneça o número de alunos")
9     Leia (qtde)
10    maior <-0
11    para cont de 1 ate qtde faca
12       Escreval ("Aluno ",cont)
13       Escreval ("Digite o nome do aluno: ")
14       Leia (nome)
15       Escreval ("Digite a nota de ",nome)
16       Leia (nota)
17       Se (nota > maior) entao
18          maior <- nota
19          nome_m <- nome
20       FimSe
21    Fimpara
22    Escreval ("O melhor aluno foi ",nome_m," com a nota ",maior)
23 fimalgoritmo
24
25
```

O IDE VisuAlg permite o uso de algumas funções pré-definidas para a manipulação de **strings,** tal como apresentado na tabela a seguir:

Função	Significado
Asc(x : caractere)	Retorna um número inteiro com o código ASCII do primeiro caractere da expressão.
Carac(x : inteiro)	Retorna o caractere cujo código ASCII corresponda à expressão.
Caracpnum(x : caractere)	Retorna o inteiro ou real representado pela expressão matemática (ou valor numérico no formato literal) – converte caractere em numérico.
Compr(x : caractere)	Retorna um inteiro contendo o comprimento (quantidade de caracteres) da **string**.
Copia(x : caractere; p, n : inteiro)	Retorna uma **string** contendo uma cópia parcial da expressão, a partir do caractere p, contendo n caracteres. Os caracteres são numerados da esquerda para a direita, começando em 1.
Maiusc(x : caractere)	Retorna uma **string** contendo a expressão em maiúsculas.
Minusc(x : caractere)	Retorna uma **string** contendo a expressão em minúsculas.
Numpcarac(x : inteiro ou real)	Retorna a representação de x como uma cadeia de caracteres (converte um tipo numérico para caractere).
Pos(seq, x : caractere)	Retorna um inteiro que indica a posição em que a cadeia de caracteres (seq) se encontra dentro da cadeia x, ou zero se seq não estiver contida em x.

Para exemplificar, vejamos, no algoritmo a seguir, o uso de algumas dessas funções.

```
1  algoritmo "Exemplo Funções"
2  var
3    valor : caractere
4    convertida, sequencia : caractere
5    comprimento, posicao : inteiro
6  inicio
7     escreval("Exemplo de funções pré-definidas")
8     escreval("Forneça uma string")
9     leia(valor)
10    convertida <- maiusc(valor)
11    escreval ("Mostra a string em letras maiúsculas  ",convertida)
12    comprimento <- Compr(valor)
13    escreval ("A palavra ", valor, " possui ", comprimento, " caracteres")
14    escreval("Digite uma letra ou sequência de caracteres da string:")
15    leia(sequencia)
16    posicao <- Pos(sequencia, valor)
17    escreval("O caractere foi encontado na posição ", posicao)
18 fimalgoritmo
19
```

A primeira função usada foi **maiusc**, que converteu o conteúdo da variável **valor** em letras maiúsculas. Na sequência, a função **Compr** retorna o número de dígitos da **string** lida e, por fim, a função **Pos** retorna a posição do valor informado na variável **sequencia**.

A área de visualização dos resultados mostra a execução, na qual foi digitada a palavra **unitau,** que foi armazenada na variável **valor,** sendo possível observar o resultado da aplicação de cada função.

Área de visualização dos resultados

```
Forneça uma string
unitau
Mostra a string em letras maiúsculas  UNITAU
A palavra unitau possui  6 caracteres
Digite uma letra ou sequência de caracteres da string:
i
O caractere foi encontado na posição  3

Fim da execução.
```

6.3 Funções e procedimentos da biblioteca *string.h* para manipulação de uma *string* em Linguagem C

Nesse Tópico, veremos como aplicar os caracteres e as cadeias de caracteres nos programas em linguagem C.

```c
#include <stdio.h>
int main ()
{
char primeiro_nome[6]= {'M', 'a', 'r', 'i', 'a', '\0'};
char ultimo_nome[6]="Alves";
printf("Ola! : %s %s\n", primeiro_nome, ultimo_nome);
}
```

Uma **string** em linguagem C pode ser considerada um vetor de caracteres que finaliza com um caractere nulo **\0**. O programa a seguir mostra um exemplo.

```c
#include <stdio.h>
int main ()
{
char texto[100];
int i;
printf("\n Mostra o número de caracteres de uma string");
printf("\n Digite uma palavra:");
scanf("%s", texto);
for (i=0; texto[i] != '\0'; i++);
printf("\n A string possui %d caracteres\n", i);
}
```

Foram declarados dois vetores do tipo **char**, com as duas possíveis representações, o **primeiro _nome** com dimensão 6 atribui os caracteres englobados por aspas simples, finalizando com o caractere nulo e o **ultimo_nome** com a **string** entre aspas duplas.

Essa representação possibilita que os caracteres que formam a **string** sejam acessados individualmente, o que proporciona grande flexibilidade na sua manipulação.

Você sabia que existe um tipo em linguagem C chamado **void**, que significa vazio. Ele nos permite fazer funções que não retornam nada, ou seja, que não têm valor de retorno.
Observe um exemplo de funções que usam o tipo **void**:

```
#include <stdio.h>
void frase (void);
int main ( )
{
        frase( );
        printf ("\n Qual a frase:\n");
        frase( );
        return 0;
}
void frase (void)
{
        printf ("\n O planeta Terra é lindo! ");
}
```

A leitura de uma **string** é feita da mesma forma que as demais variáveis. No exemplo a seguir, a variável **texto** é lida e a estrutura **for** é finalizada na mesma linha, com o terminador ponto-e-vírgula tendo o objetivo apenas de contar o número de caracteres da **string**.

Outra forma de realizar a leitura de uma **string** é através do comando **fgets**.

```
1  #include <stdio.h>
2  int main ()
3  {
4  char s[10];
5  fgets(s, 10, stdin);
6  printf("%s\n", s);
7  }
8
```

O programa acima declara uma variável **s** com capacidade para 10 elementos do tipo char. Apenas realiza a leitura da **string** e a mostra em seguida. O comando **fgets** lê os valores até o tamanho definido menos 1 e o copia para a **string s**. Outra vantagem dessa instrução é que a leitura não se interrompe com o espaço, como é o caso do comando **scanf**.

Assim como vimos nos algoritmos, a linguagem de programação C também apresenta um conjunto de funções pré-definidas invocadas pela biblioteca **<string.h>.**

Para exemplificar algumas das funções pré-definidas em linguagem C, veremos as mais utilizadas.

```c
#include <stdio.h>
#include <string.h>
int main ()
{
printf("\n copia o conteúdo de uma string em outra");
char st1[20], st2[20];
printf("\n Digite uma string:");
scanf("%s", st1);
strcpy(st2, st1); // copia st1 para st2
printf("string 1 = %s\nstring 2 = %s\n", st1, st2);
}
```

No programa acima, devemos observar a necessidade da inclusão da diretiva **include** com a biblioteca **<string.h>**. Após a declaração das **strings** e a leitura da **string st1**, o comando **strcpy** faz a cópia do conteúdo da **string st1** para a **string st2,** sendo sua sintaxe o primeiro parâmetro, o destino, e o segundo parâmetro, a origem.

Na sequência, um exemplo em que se concatenam duas **strings** e se mostra o tamanho da **string** concatenada.

```c
#include <stdio.h>
#include <string.h>
int main ()
{
  char s1[20], s2[20];
  printf("\n Concatena duas strings");
  printf("\nDigite a primeira string");
  scanf("%s", s1);
  printf("\nDigite a segunda string");
  scanf("%s", s2);
  strcat(s1, s2);
  printf("s1 = %s\ns2 = %s\n", s1, s2);
  printf("\n retorna o tamanho de uma string s1");
  printf(" s1 Contém %d caracteres.\n", strlen(s1));
}
```

O programa declara e lê as strings **s1** e **s2**. Com a função **strcat**, o conteúdo de **s2** é copiado ao final do conteúdo de **s1**, que tem seu valor alterado para o conteúdo concatenado. Também é mostrado o tamanho da **string s1** após a concatenação com a função **strlen**.

O próximo exemplo usa a função **strcmp** para comparar duas **strings** e retorna 0 se as duas **strings** forem iguais, retorna –1 se a posição da primeira **string** for menor e retorna +1 se a posição da primeira **string** for maior que a posição da segunda **string**.

```c
#include <stdio.h>
#include <string.h>
int main ()
{
  char s1[20], s2[20];
  printf("\n Comapara strings");
  printf("\n Digite a primeira string = ");
  scanf("%s", s1);
  printf("Digite a segunda string = ");
  scanf("%s", s2);
  if (strcmp(s1, s2) == 0)
    printf("strings iguais!\n");
  else
  {
    strcat(s1, s2);
    printf("%s\n", s1);
  }
}
```

Após a declaração e leitura das duas **strings**, a função **strcmp** é usada na estrutura **if** para verificar se o resultado da comparação é igual a zero. Caso essa comparação seja verdadeira, é mostrada uma mensagem; caso contrário, é usada a função **strcat** para concatenar as duas **strings**.

A tabela a seguir mostra um resumo de algumas funções pertencentes à biblioteca <string.h>.

Função	Significado
strcpy	Realiza a cópia do conteúdo de **string** para outra.
strncmpy	Realiza a cópia do conteúdo de **string** para outra com especificação do tamanho a ser copiado.
strcat	Realiza a concatenação de duas **strings**.
strncat	Realiza a concatenação de duas **strings** com especificação do tamanho a ser concatenado.
strlen	Determina o tamanho de uma **string**.
strcmp	Compara o conteúdo de duas **strings**.
strncmp	Compara o conteúdo de duas **strings**, devendo ser especificado o tamanho a ser comparado
strtok	Permite dividir uma **string** em partes.

6.4 Síntese da Unidade

Nesta Unidade, vimos que podemos manipular caracteres e cadeias de caracteres, ou seja, **strings**.

Conversamos sobre a diferença entre um caractere e uma cadeia de caracteres, evidenciando que há diferenças na forma de declaração nos algoritmos e na linguagem de programação C.

Também vimos que tanto para os algoritmos como para os programas em linguagem C podemos trabalhar dentro da lógica convencional, ou podemos utilizar funções pré-definidas.

Além disso, você aprendeu que para usar as funções pré-definidas em linguagem C é necessário invocar a diretiva **include** com a biblioteca <string.h>.

Por fim, vimos que a aplicação das funções pré-definidas oferece diversas funcionalidades em relação à manipulação de **strings**.

Esperamos que os saberes apresentados tenham ajudado você a compreender os conceitos sobre caracteres e **strings**.

6.5 Para Saber Mais

ARAÚJO, S. **Lógica de Programação e algoritmos.** Ed. Contentus: Curitiba, 2020. Capítulo 5.

UNITAU digital

ISBN: 978-65-86914-52-8