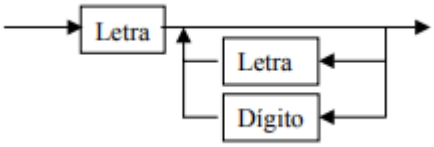
APOSTILA DE ALGORITMOS

Algoritmos: Comandos Básicos

IDENTIFICADORES, CONSTANTES, VARIÁVEIS E TIPOS BÁSICOS

• Identificadores: Representam os nomes escolhidos para rotular as variáveis, procedimentos e funções, normalmente, obedecem às seguintes regras:



1. O primeiro caracter deve ser uma letra

2. Os nomes devem ser formados por caracteres pertencentes ao seguinte conjunto : {a,b,c,..z,A,B,C,...Z,0,1,2,...,9,_}

3. Os nomes escolhidos devem explicitar seu conteúdo.

EX: A, B1, BC3D,SOMA, CONTADOR

Tipos básicos:

INTEIRO (“int, short int ou long int”): : qualquer número inteiro, negativo, nulo ou positivo. Ex: -2, -1, 0... Operações: soma(+), subtração(-), multiplicação(*), divisão inteira(/), resto(%) e comparações.

REAL (“float ou double”): qualquer número real, negativo, nulo ou positivo. Ex: 2.5, 3.1 Operações: soma(+), subtração(-), multiplicação(*), divisão exata(/) e comparações.

CARACTER (“char”): qualquer conjunto de caracteres alfanuméricos. Ex: A, B, "ABACATE " Operações: comparações

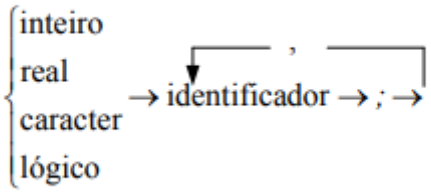
TEXTO OU CADEIA DE CARACTERES (“STRING”): uma variável deste tipo poderá armazenar uma cadeia de caracteres de qualquer tamanho. Caso seja imprescindível para o entendimento pode-se acrescentar, entre parênteses, a quantidade máxima de caracteres. (Exemplo: texto (10)). Obs.: Os textos deverão ser representados sempre entre apóstrofes para que não se confundam com os valores numéricos. Veja que o inteiro 5, é diferente do texto ‘5’.

LÓGICO (“BOOLEAN”): tipo especial de variável que armazena apenas os valores V e F, onde V representa VERDADEIRO e FALSO Ex: e, ou, não Operações: Verdadeiro ou Falso.

DECLARAÇÃO DE VARIÁVEIS

Consiste na definição dos nomes e valores das constantes e dos nomes e tipos das variáveis que serão utilizadas pelos algoritmos, previamente à sua utilização, incluindo comentário, quando se fizerem necessários. Na maioria das linguagens de programação, quando o computador está executando um programa e encontra uma referência a uma variável ou a uma constante qualquer, se esta não tiver sido previamente definida, ele não saberá o que fazer com ela. Da mesma forma, um programador que estiver implementando um algoritmo, em alguma linguagem de programação, terá o seu trabalho simplificado se todas as constantes e variáveis referenciadas no algoritmo tiverem sido previamente declaradas. As constantes são declaradas antes das variáveis. Vejamos os formatos da declaração e alguns exemplos.

O significado da declaração de variáveis corresponde à criação de locais na memória rotulada com o nome da variável (identificador) e marcada com o tipo de valores que ela pode conter. Para que os programas manipulem valores, estes devem ser armazenados em variáveis e para isso, devemos declará-las de acordo com a sintaxe:



Ex:

Inteiro X1;

obs.: X1 é o nome de um local de memória que só pode conter valores do tipo inteiro

real SOMA, MÉDIA;

caracter frase, nome;

inteiro X1;

real A,B;

lógico TEM;

PALAVRAS RESERVADAS

São palavras que terão uso específico no nosso pseudo-código e que não deverão ser usadas como identificadores, para não causar confusão na interpretação.

Exemplo: Algoritmo, Programa, Bloco, Procedimento, Inteiro, Real, Texto, Const, Var, Tipo, Início, Imprima, Se, Então, Senão, Enquanto, Repita, Variando, Faça, Caso, Até, Vetor, Matriz, Registro, Fim, Execute, Procedimento, Função, etc.... O significado de cada um desses termos será visto e entendido nos itens e capítulos que se seguem.

COMANDO SIMPLES

É uma instrução simples.

leia(x);

COMANDO COMPOSTO

um grupo de comandos simples que executam alguma tarefa.

Início

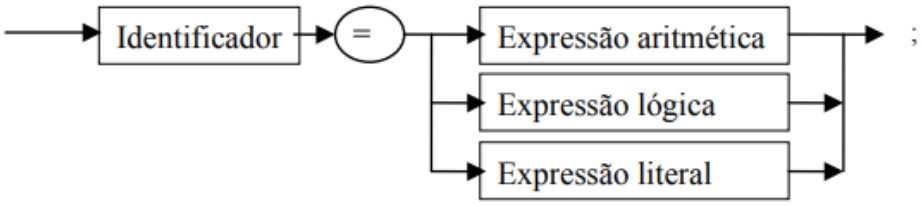
leia(x);

y = 2*x;

fim

OPERADORES Na solução da grande maioria dos problemas é necessário que as variáveis tenham seus valores consultados ou alterados e, para isto, devemos definir um conjunto de **OPERADORES**, sendo eles:

- **OPERADOR DE ATRIBUIÇÃO**



Ex: O valor da expressão é atribuído ao identificador(variável).

X = 2; y = 5-x;

Este comando permite que se forneça ou altere o valor de uma determinada variável, onde o tipo desse valor seja compatível ao tipo de variável na qual está sendo armazenado, de acordo com o especificado na declaração.

NUM = 8 {A variável NUM recebe o valor 8}

NOME = “Guilherme” {A variável NOME recebe o valor ‘Guilherme’}

CONT = 0

AUXNOME = NOME {A variável AUXNOME recebe o conteúdo da variável NOME}

ACHOU = falso {A variável ACHOU recebe o valor falso}

OPERADORES ARITMÉTICOS:

+ = Adição	/ = Quociente da divisão de inteiros 5/2=2
* = Multiplicação	% = Resto da divisão de inteiros 5%2=1
- = Subtração ou inverso do sinal.	a**b = Exponenciação a^b
/ = Divisão	TRUNCA(A) - A parte inteira de um número Fracionário.
SINAL(A) - Fornece o valor -1, +1 ou zero conforme o valor de A seja negativo, positivo ou igual a zero.	ARREDONDA (A) - Transforma por arredondamento, um número fracionário em inteiro.

Obs. FUNÇÕES PRIMITIVAS:

As funções mais comuns de matemática são também definidas e válidas no nosso Pseudo Código.

Exemplos:

LOG (X) , {dá o logaritmo na base 10 de X}

SEN (X), {dá o seno de X}

ABS (X), {dá o valor absoluto de X}

INT (X), {dá a parte inteira de um real X}

ARREDONDA (x), {arredonda um real para inteiro}

RAIZ (X), {dá a raiz quadrada de X} etc. ...

Operadores Relacionais

São utilizados para relacionar variáveis ou expressões, resultando num valor lógico (Verdadeiro ou Falso), sendo eles:

== igual	!= diferente de
< menor	> maior
<= menor ou igual	>= maior ou igual

Exemplos:

NUM = 3

NOME = ‘DENISE’

NUM > 5 é falso (0) {3 não é maior que 5}

NOME < ‘DENIZ’ é verdadeiro (1) {DENISE vem antes de DENIZ}

(5 + 3) > = 7 é verdadeiro

NUM != (4 – 1) é falso {lembre-se que NUM “recebeu” 3}

OPERADORES LÓGICOS:

São utilizados para avaliar expressões lógicas, sendo eles:

e ("and") Ù : conjunção

ou ("or") Ú :disjunção

não ("not") Ø : negação Exemplos: ACHOU = falso NUM =9 (4>5) e (5>3) =>falso e verdadeiro (1) = falso (0)

não ACHOU =>verdadeiro

PRIORIDADE DE OPERADORES:

Durante a execução de uma expressão que envolve vários operadores, é necessário a existência de prioridades, caso contrário poderemos obter valores que não representam o resultado esperado.

A maioria das linguagens de programação utiliza as seguintes prioridades de operadores :

1º - Efetuar operações embutidas em parênteses "mais internos"

2º - Efetuar Funções

3º - Efetuar multiplicação e/ou divisão

4º - Efetuar adição e/ou subtração

5º - Operadores Relacionais

6º - Operadores Lógicos Ou seja,

Primeiro: Parênteses e Funções

Segundo: Expressões Aritméticas

1)+, - (Unitários)

2) **

3) *,/

4)+,- (binário)

Terceiro: Comparações

Quarto: Não

Quinto: e

Sexto: ou

OBS: O programador tem plena liberdade para incluir novas variáveis, operadores ou funções para adaptar o algoritmo as suas necessidades, lembrando sempre, de que, estes devem ser compatíveis com a linguagem de programação a ser utilizada.

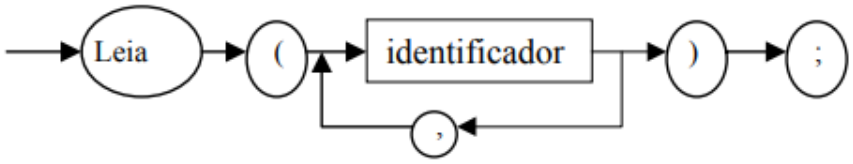
COMANDOS DE ENTRADA E SAÍDA

No algoritmo é preciso representar a troca de informações que ocorrerá entre o mundo da máquina e o nosso mundo, para isso, devemos utilizar comandos de entrada e saída, sendo que, a nível de algoritmo esses comandos representam apenas a entrada e a saída da informação, independe do dispositivo utilizado (teclado, discos, impressora, monitor,...), mas, sabemos que nas linguagens de programação essa independência não existe, ou seja, nas linguagens de programação temos comandos específicos para cada tipo de unidade de Entrada/Saída.

Comando de Entrada de Dados

Para que possamos obter dados do meio exterior para uso do computador (memória principal), estes têm de vir através dos dispositivos de entrada. Da mesma forma, as informações que são produzidas, tem de ser levadas ao meio externo (um arquivo, um a impressora, uma tela etc.) através de um dispositivo de saída. Para isso, utilizamos dois comandos assim definidos: Comando Leia(scanf no c): lê, do meio externo, a próxima informação disponível para leitura e armazena na(s) variável(eis) discriminada(s) após o comando, entre parênteses. Mais tarde aprenderemos como especificar a leitura de um dado que está armazenado em um arquivo e de que arquivo está sendo lido o dado.

Leia(variável_1, variável_2,...);



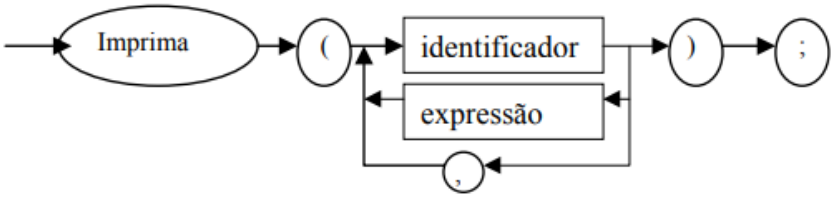
Ex: leia (v); O valor da variável (v) é dado por um dispositivo de entrada.

Comando de Saída de Dados

Comando Imprima (printf no c): Imprime (na tela ou na impressora) o conteúdo da(s) variável(eis) especificada(s) após o comando, entre parênteses. Não será preocupação nossa a formatação de relatórios, mas o comando permite a impressão de texto (entre “), se for necessária para clareza ou especificação do que está sendo impresso.

- **Comando de Saída de Dados**

Comando **Imprima** (printf no c): Imprime (na tela ou na impressora) o conteúdo da(s) variável(eis) especificada(s) após o comando, entre parênteses. Não será preocupação nossa a formatação de relatórios, mas o comando permite a impressão de texto (entre “), se for necessária para clareza ou especificação do que está sendo impresso.



ESTRUTURAS BÁSICAS DE CONTROLE

SELEÇÃO

Também chamada de estrutura de decisão ou de processamento condicional, a estrutura de seleção é utilizada quando a execução de um comando (ou uma seqüência de comandos) depende de um teste anterior (uma ou mais comparações). A seleção pode ser simples ou composta.

SELEÇÃO SIMPLES

Quando a execução de um comando (ou de uma seqüência de comandos) depender de uma condição verdadeira, e não há comandos a executar se a condição for falsa.

- **Nosso Pseudo-código**

```
Se <condição> Então
      (comandos);
Fim Se;
```

EX: Se o valor de A for menor do que 5 leia A.

```
Se (a<5) Então
  a=5;
Fim Se;
```

SELEÇÃO COMPOSTA

Quando se executa um comando (ou seqüência de comando) se uma condição é verdadeira, e se executa um outro comando (ou seqüência de comandos) se a condição é falsa.

- **Nosso Pseudo-Código**

```
Se <condição> Então
  (Comandos);
Senão
  (comandos);
Fim Se;
```

ANINHAMENTO DE SELEÇÕES

A estrutura de Seleção permite o aninhamento, ou seja, o comando a ser executado dentro de uma seleção (por exemplo, no "Senão") pode ser uma outra seleção. Outro aninhamento poderá ocorrer também com esta última seleção e assim por diante. Nos casos de vários aninhamentos subseqüentes, uma boa identação será fundamental para o entendimento de algoritmo quando utilizando pseudo-código.

- **Nosso Pseudo-Código**

```
Início
 -
 Se CONDIÇÃO_A Então {V}
   comando1;
 Senão {F}
   Se CONDIÇÃO_B Então
      comando2;
   Senão
      comando3;
   Fim Se;
 Fim se;
 -
 -
Fim.
```

ESTRUTURAS DE REPETIÇÃO - LAÇOS(LOOPS)

LAÇO ENQUANTO(WHILE)

A estrutura de repetição (enquanto) é utilizada quando um conjunto de comandos deve ser executado repetidamente, enquanto uma determinada condição (expressão lógica) permanecer verdadeira. Dependendo do resultado do teste da condição, o conjunto de comandos poderá não ser executado nem uma vez (se for falsa no primeiro teste), ou será executado várias vezes (enquanto for verdadeira). Chama-se a isso um laço ("loop").

Da mesma forma que a estrutura de seleção, ela permite o aninhamento de repetições, ou seja, a existência de uma estrutura de repetição dentro de outra. Poderão haver também aninhamentos de seleções dentro de estruturas repetitivas e vice-versa. Dois cuidados ao criar estruturas de repetição (enquanto):

1. Inicializar a(s) variável(eis) que controla(m) o laço antes do início do laço;

2. Inicializar a(s) variável(eis) que controla(m) o laço dentro do laço (seja por leitura ou por atribuição), pois se isto não for feito cairemos no que chamamos um laço infinito e de lá o nosso programa não sairá.

- **Nosso Pseudo-Código**

```
Enquanto <Condição> Faça
    (Comandos);
Fim Enquanto;
```

Exemplo:

```
Início
   Inteiro x;
   x = 2;
   Enquanto (x<10) Faça
     Imprima (x);
     x=x+1;
   Fim Enquanto;
   Imprima(x);
Fim.
```

CONTROLADOS POR CONTADOR

Uma variável é fornecida com o n.º de vezes que será repetido o laço.

```
Algoritmo Abono_por_Contador
 Início
   Leia (Numero_de_Funcionários);
   Contador = 0;
   Enquanto Contador < Número_de_Funcionários Faça
     Leia (Nome);
     Contador = Contador+1;
   Fim Enquanto;
 Fim.
```

REPETIÇÃO COM TESTE NO FINAL

Quando se deseja executar a série de comandos uma vez pelo menos, pode se fazer o teste no final. Essa variação tem um uso bastante efetivo na validação de dados de entrada, pelo teclado, mas pode ser sempre substituída por um enquanto. Uma vantagem do repita é que não é preciso inicializar a(s) variável(eis) de controle do laço antes de entrar no mesmo. Deve-se, contudo, ter o cuidado de modificá-la(s) dentro do laço para que não caiamos em um laço infinito. Executa uma instrução e faz um teste lógico. Dependendo da resposta, fica repetindo o processo até o teste lógico dar Verdadeiro.

- **Nosso Pseudo-Código**

```
repita
     C1;
     C2;
    .
    .
    Cn;
até <condição>;
```

Significado deste comando:

Os comandos C1, C2,....,Cn são executados pelo menos uma vez. Quando a condição é encontrada, ela é testada, se for verdadeira o comando seguinte será executado, se for falsa, os comandos C1, C2,...,Cn são reexecutados até que a condição se torne verdadeira.

O comando repita-até é equivalente ao comando enquanto, conforme será mostrado no exemplo abaixo

- Repita – até

```
Início
  Inteiro x;
  X = 2;
  Repita
    Imprima(x);
    X = x+1;
  Até (x>=10);
  Imprima(x);
Fim.
```

REPETIÇÃO COM VARIÁVEL DE CONTROLE –PARA (for)

para v de i até l passo p faça

onde:

v: variável de controle.

i: valor inicial de v.

l: valor final de v.

p: valor do incremento de v.

- **Nosso Pseudo-Código**

```
Para v de i até l passo p Faça
      C1;
      C2;
      .
      .
      Cn;
Fim Para;
```

Significado do comando: v, i, l, p são variáveis quaisquer e que, de acordo com as regras da programação estruturada, não devem ser modificadas nos comandos C1, C2, . . ., Cn. O comando para é, na verdade, o comando enquanto utilizando-se uma variável de controle, escrito numa notação compactada. Neste caso existirá sempre uma inicialização da variável de controle, um teste para verificar se a variável atingiu o limite e um acréscimo na variável.

Exemplo para comparação entre **Enquanto** e **Para**

```
Início
  Inteiro x;
  X = 1 /*inicialização*/
  Enquanto x<=10 Faça /*teste*/
    Leia(v[x]);
    x = x+1; {acréscimo}
  Fim Enquanto;
Fim.
```

```
Início
  Inteiro x;
  Para x de 1 até 10 Passo 1 Faça
    Leia(v[x]);
  Fim Para
Fim.
```

VETORES

Um vetor ou agregado homogêneo, ou ainda variável composta homogênea, é uma estrutura de dados que contêm elementos de mesmo tipo, que podem ser referenciados como um todo. Ao declararmos um vetor, estamos reservando na memória principal do computador uma série de células para uso da variável daquele tipo. O nome do vetor aponta para a base das células e o seu início dá a posição relativa do elemento referenciado ao primeiro (base). Nem sempre os tipos básicos (inteiro, real, caracter e lógico) são suficientes para exprimir estruturas de dados em algoritmos. Por exemplo consideremos um problema em que um professor com 5 alunos deseja imprimir a nota e a média de seus alunos. Nesse caso seria necessário se considerar cinco variáveis reais para contar as notas dos cinco alunos. Imagine que o número de alunos da turma seja 80. Só a declaração destas variáveis tornaria impraticável a redação do algoritmo. Daí a necessidade de novos tipos serem criados. Um destes tipos, o vetor, será estudado.

Os vetores podem ser unidimensionais ou multi-dimensionais (matrizes). Um vetor unidimensional, como uma lista de notas dos 50 alunos de uma turma, tem apenas um índice. Se existirem porém várias turmas poderemos utilizar um vetor com dois índices (o número da turma e o número do aluno da turma). Abordaremos este assunto ainda neste capítulo.

DECLARAÇÃO DE VETORES

tipo_da_variável nome_da_variável [li:lf];

li - Posição Inicial do Vetor

lf - Posição Final do Vetor

Quando o C vê uma declaração como esta ele reserva um espaço na memória suficientemente grande para armazenar o número de células especificadas em tamanho. Por exemplo, se declararmos:

float exemplo [20];

o C irá reservar 4x20=80 bytes. Estes bytes são reservados de maneira contígua. Na linguagem C a numeração começa sempre em zero. Isto significa que, no exemplo acima, os dados serão indexados de 0 a 19. Para acessá-los vamos escrever:

```
exemplo[0]
exemplo[1]
.
.
exemplo[19]
```

Algoritmo Notas {Sem Vetor}

```
Início
   Caracter Nome1, Nome2, Nome3;
   Leia(Nome1, Nota1);
   Leia (Nome2, Nota2);
   Leia (Nome3, Nota3);
   Média = (Nota1+Nota2+Nota3)/3.0 ;
   Se Nota1 > Média Então
      Imprima (Nome1);
   Fim Se;
   Se Nota2 > Média Então
      Imprima (Nome2);
   Fim Se;
   Se Nota3 > Média Então
      Imprima (Nome3);
   Fim Se;
Fim.
```

Algoritmo Notas {Com Vetor}

```
Início
  Caracter Nome[0:2]
  Real Nota[0:2];
  Para I de 0 até 2 Faça
      Leia (Nome[I], Nota[I]);
  Fim Para;
  /*Cálculo da Média*/
  Soma = 0.0 ;
  Para I de 0 até 2 Faça
      Soma = Soma + Nota [I] ;
  Fim Para;
  Média = Soma/3.0;
  Para I de 0 até 2 Faça
      Se Nota[I] > Média Então
        Imprima (Nome[I]);
      Fim Se;
  Fim Para;
Fim.
```