Capítulo 1

Lógica de programação

- Fluxograma e pseudocódigo
- Tipos de dados
- Constantes e variáveis
- Operadores
- Comandos
- Vetor
- Matriz
- Programação modular

A lógica computacional é um dos primeiros passos para quem pretende entrar no mundo da computação. Normalmente, não percebemos sua aplicação para resolver diferentes níveis de problemas nem sua relação com outras áreas da ciência. Isso porque, em geral, a lógica computacional está associada à lógica matemática (FORBELONE, 2005). É um recurso que permite a criação de alternativas viáveis e tecnicamente corretas, eficientes e rápidas. Antes de resolver um problema, no entanto, temos de saber interpretá-lo. Assim, evitam-se resultados indesejáveis ou desvios de percurso, da mesma maneira que surge a oportunidade de eliminar eventuais dúvidas, definindo o melhor algoritmo, que pode ser o pseudocódigo ou o fluxograma.

A maioria das atividades que executamos é resultado de uma sequência de passos lógicos. Por exemplo, amarrar o cadarço de um tênis, abrir uma porta, trocar uma lâmpada ou fazer aquele delicioso bolo de chocolate. A sequência das etapas, ou melhor, das operações a executar, é determinada pelo algoritmo de forma lógica. Uma maneira mais simples de dizer isso é que a sucessão de ações faz todo o sentido. Usando o mesmo exemplo do bolo, o começo de tudo é reunir os ingredientes e usá-los na sequência certa. É assim também com as operações de cálculos, representadas pelo pseudocódigo a partir de comandos no formato de texto (exatamente como uma receita) ou pelo fluxograma de forma gráfica. E o fato de ser constituído por símbolos geométricos facilita o processo de visualização das operações.

1.1. Fluxograma e pseudocódigo

Para entender melhor o conceito, vamos elaborar, resumidamente, um algoritmo que represente os passos necessários para que alguém tome um refrigerante em uma lata: 1. pegar o recipiente; 2. abrir a tampa; 3. tomar o conteúdo (veja quadro *Algorítimo com os passos necessários da latinha*). A partir dessa mesma situação, é possível inserir mais operações em nosso algoritmo, deixando-o mais detalhado (observe o quadro *Algoritmo detalhado da latinha*).

Quadro
Algoritmo com os passos necessários da latinha.

1 - Pegar o recipiente. **2 -** Abrir a tampa. **3 -** Tomar o conteúdo.

Quadro
Algoritmo detalhado da latinha.

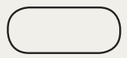
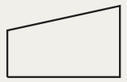
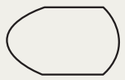
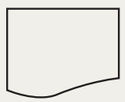
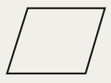
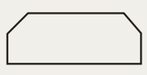
1 - Pegar o refrigerante com a mão esquerda. **2 -** Segurar o lacre com a mão direita. **3 -** Remover o lacre.

4 Eliminar o lacre. **5** Posicionar a lata corretamente na boca. **6 -** Inclinar a lata.

7 - Ingerir o líquido. **8 -** Voltar a lata na posição original.

Entre os diferentes símbolos que constituem a estrutura de um fluxograma, os mais comuns estão incluídos na tabela 1.

Tabela 1

SIMBOLOGIA DO FLUXOGRAMA	
SÍMBOLO	**NOME E FUNÇÃO**
	**NOME = TERMINAL** **FUNÇÃO = Indica INÍCIO ou FIM de um processamento**
	**NOME = PROCESSAMENTO** **FUNÇÃO = Definição de variáveis ou processamentos em geral (Cálculos)**
	**NOME = ENTRADA MANUAL** **FUNÇÃO = Entrada de dados via teclado, idêntico ao comando LEIA**
	**NOME = DISPLAY** **FUNÇÃO = Saída de Dados, mostra um texto e/ou variável na tela, idêntico ao comando ESCREVA**
	**NOME = DOCUMENTO** **FUNÇÃO = Saída de Dados, envia um texto e/ou variável para a impressora, usado em relatórios. Idêntico ao comando IMPRIMA**
	**NOME = DECISÃO** **FUNÇÃO = Decisão a ser tomada, retornando verdadeiro ou falso, idêntico ao comando SE**
	**NOME = CONECTOR** **FUNÇÃO = Desvia o fluxo para uma outra página, sendo interligados pelo conector**
	**NOME = ENTRADA/SAÍDA** **FUNÇÃO = Leitura de gravação de arquivos**
	**NOME = SETA** **FUNÇÃO = Indica a direção do fluxo**
	**NOME = LOOP** **FUNÇÃO = Realiza o controle de loop**

Aproveitando o mesmo exemplo da latinha de refrigerante, vamos reescrever os passos lógicos por meio de um fluxograma (figura 1).

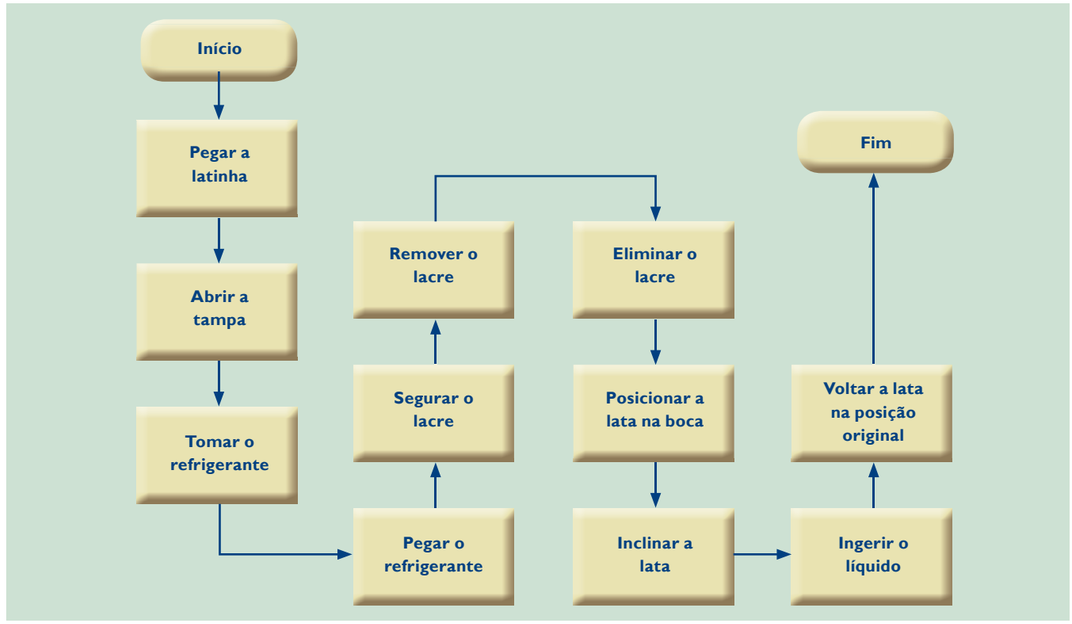


Figura 1
Fluxograma dos passos lógicos.

Existem muitos softwares destinados à construção de fluxogramas, mas é possível fazê-los também com o uso do Microsoft Word®. Os exemplos desse material foram elaborados pelo Microsoft Visio®.

Pseudocódigo

Os algoritmos são desenvolvidos em uma linguagem denominada pseudocódigo ou, como preferem alguns autores, português estruturado. Podem ser criados sem o formalismo das linguagens de programação, mas obedecem a uma regra básica de estruturação: cabeçalho, declaração e instruções, conforme o seguinte modelo:

```
Programa nome_do_programa          ——— Título do Algoritmo
Declare
        {declaração de variáveis}
Início                             ——— Declarações de variáveis
        {instruções e comandos}
Fim                                ——— Comandos e instruções ordenados de forma lógica
```

Um algoritmo deve ter como característica um número finito de passos, descritos de forma precisa e que possam ser executados em um determinado tempo. Pode permitir zero ou mais entradas de dados e gerar saídas de dados refinados. Para construir um algoritmo, é preciso usar informações claras e objetivas, geralmente compostas de um verbo ou de uma frase direta.

1.2. Tipos de dados

A manipulação de dados em um algoritmo – ou até mesmo nas linguagens de programação – só é possível com base em algumas regras fundamentais para a definição dos tipos de dados. É preciso diferenciar um dado com números ou outros tipos de caracteres.

• Numéricos inteiros

Esse tipo de dado é representado por números não fracionados positivos ou negativos, como: 10, 50, -56, -1.000 etc.

• Numéricos flutuantes

São representados por números fracionados positivos ou negativos, por exemplo: 10.5, -45.78, 178.67 etc.

• Caracteres ou literais

É o maior conjunto de dados. São representados por letras, números, símbolos e espaço em branco, sempre entre aspas. Exemplo: “cadeira”, “100”, “R$ 10.00”, “11h00” etc.

1.3. Constantes e variáveis

As constantes são expressões que recebem um valor qualquer que não será modificado durante o processo de execução do algoritmo. Já as variáveis terão um valor passível de modificação a qualquer momento. Um exemplo de constante é o valor de Pi (3,14). E uma variável que todos conhecemos bem pode ser a temperatura do dia, registrada de hora em hora.

• Atribuição e declaração

Atribuir um valor qualquer para uma constante ou uma variável requer o uso do sinal de “←” ou “=”. Isso quer dizer que o valor na frente do sinal será armazenado no elemento anterior ao sinal de atribuição. Veja:

```
nome ← “Joãozinho”      idade ← 10      pi ← 3.14
```

Independentemente do tipo de dado adotado – caracteres variados (“Joãozinho”), números inteiros (10) ou flutuantes (3.14) –, o processo de atribuição é o mesmo. Mas a declaração requer cuidado. Por isso, procure escolher o nome da variável de acordo com o tipo de dado que será recepcionado, inicie sempre com letras o nome de sua variável ou constante e não utilize espaço em branco ou caracteres especiais, com exceção do *underline* (_). Exemplos:

```
nome          ←   “Joãozinho”
idade         ←   0
minha_idade   ←   15
pag_final     ←   “R$ 1.479,00”
```

No pseudocódigo (figura 2), as constantes ou variáveis deverão estar relacionadas com os seus respectivos tipos de dados:

```
Programa variáveis

Declare

nome como caractere

idade como inteiro

salário como flutuante

Início

nome ← “Joãozinho”

idade ← 15

salário ← 1.500,00

Fim
```

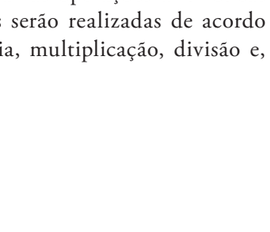


Figura 2
Estrutura de pseudocódigo.

1.4. Operadores

Os operadores são utilizados pelos pseudocódigos. O mesmo acontece com as linguagens de programação, cuja função é realizar as diferentes operações de aritmética e de lógica em um ambiente computacional.

• Aritméticos

Esse grupo tem como responsabilidade realizar as operações aritméticas (tabela 2) de um pseudocódigo. As operações serão realizadas de acordo com a prioridade, ou seja, parênteses, potência, multiplicação, divisão e, por último, soma e subtração.

Tabela 2

OPERADORES ARITMÉTICOS	
+	Adição de números flutuantes ou inteiros
+	Concatenação de valores literais (caracteres)
-	Subtração de números flutuantes ou inteiros
*	Multiplicação de números flutuantes ou inteiros
/	Divisão de números flutuantes e inteiros
**	Potência de números flutuantes e inteiros

• Relacionais

Os operadores relacionais (tabela 3) permitem a execução de testes entre constantes e variáveis.

Tabela 3

OPERADORES RELACIONAIS	
=	Igual
<>	Diferente
>	Maior
<	Menor
<=	Menor igual
>=	Maior igual

• Lógicos e Tabela Verdade

Os operadores lógicos (tabela 4) executam funções especiais dentro dos pseudocódigos, quando relacionados aos testes condicionais ou a outros valores lógicos.

Tabela 4

OPERADORES LÓGICOS	
Não / Not	Inverte o resultado de uma expressão
E / And	Retorna verdadeiro caso todas as condições retornem verdadeiro
Ou / Or	Retorna verdadeiro quando uma das condições retorna verdadeiro

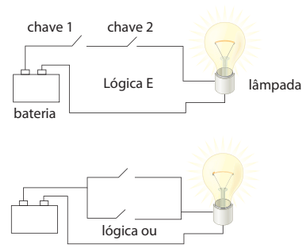
Para compreender o processo lógico de dados, observe as tabelas abaixo, mostrando os valores originais e os resultados adquiridos a partir dos operadores lógicos. Esse tipo de quadro é conhecido como Tabela Verdade (tabela 5).

Tabela 5

NÃO // NOT	
Valor Entrada 01	**Resultado**
Verdadeiro	Falso
Falso	Verdadeiro

E // AND		
Valor Entrada 01	**Valor Entrada 02**	**Resultado**
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Falso
Falso	Falso	Falso

OU // OR		
Valor Entrada 01	**Valor Entrada 02**	**Resultado**
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Falso	Verdadeiro	Verdadeiro
Falso	Falso	Falso



Simulação dos operadores "E" e "OU"

1.5. Comandos

Uma ação para o computador é definida com um comando que executa determinada operação (mostrar os dados e realizar um cálculo, por exemplo). Essas ações serão representadas nos pseudocódigos por meio de expressões predefinidas na lógica computacional.

• Comando Escreva()

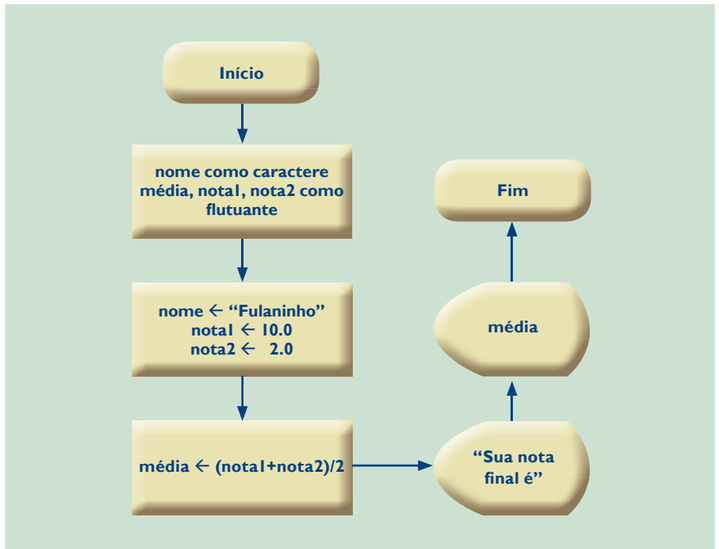
É utilizado para mostrar a saída de dados (figura 3), ou seja, quando uma mensagem ou resultado de uma operação deverá ser exibida ao usuário. Para que seja possível a visualização do dado, alguns cuidados são fundamentais: as expressões devem aparecer entre aspas, exceto as variáveis e valores (inteiros ou flutuantes).

Figura 3
Comando Escreva().

O pseudocódigo a seguir representa um exemplo mais completo, assim como o fluxograma, na sequência (figura 4):

```
Programa exemplo_escreva

Declare

   nome como caractere

   média, nota1, nota2 como flutuante

Inicio

   nome ← "Fulaninho"

   nota1 ← 10.0

   nota2 ← 2.0

   media ← (nota1 + nota2)/2

   Escreva("Sua nota final é:")

   Escreva(média)

Fim
```

Figura 4
Fluxograma Escreva.



• Comando Leia()

Ao contrário do comando Escreva(), o Leia() permite a inserção de dados pela interação com o usuário. Eles serão armazenados em variáveis devidamente definidas (figura 5).

```
Leia(nome)

Leia(nota1)

Leia(nota2)

Leia(temp)
```

Figura 5
Entrada de Dados.

Ao realizar uma entrada de dados, o ideal é que o usuário seja notificado sobre o tipo de informação que deverá ser inserida. Veja como:

```
Programa exemplo_leia

Declare

   nome como caractere

   média, nota1, nota2 como flutuante

Início

   Escreva("calculando a média")

   Escreva("Qual o seu nome?")

   Leia(nome)

   Escreva("Qual a sua nota 1?")

   Leia(nota1)

   Escreva("Qual a sua nota 2?")

   Leia(nota2)

   media ← (nota1 + nota2)/2

   Escreva("Sua média final é:")

   Escreva(media)

Fim
```

Observe na figura 6, a mesma representação, no formato de fluxograma:

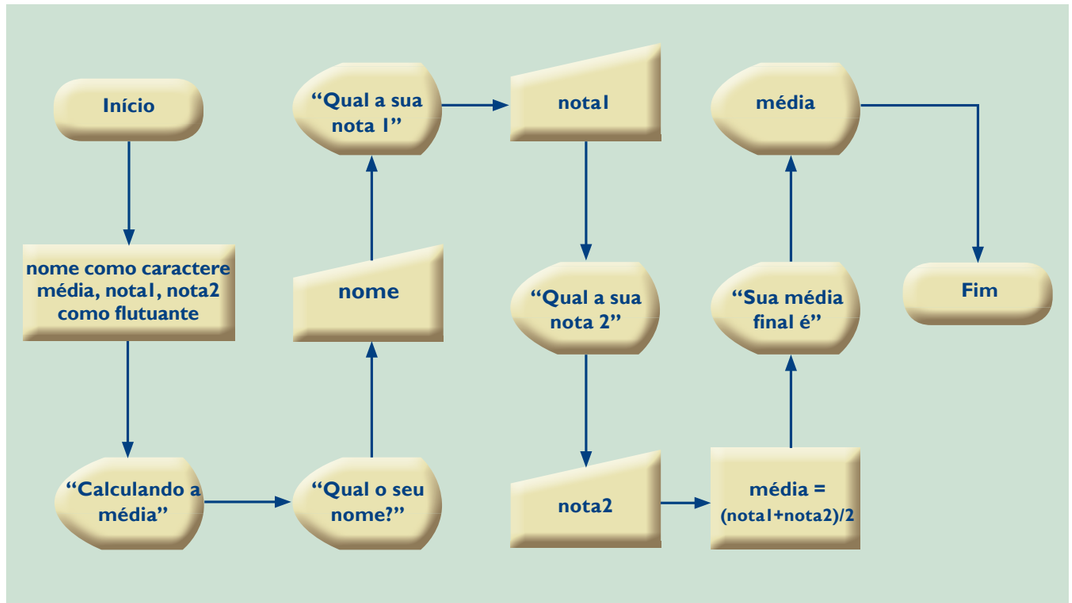


Figura 6
Fluxograma Leia.

• Estrutura de decisão – Se...Fim-Se

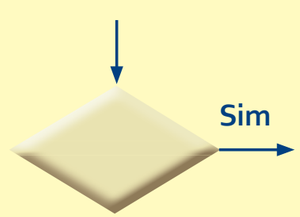
Os algoritmos, assim como as linguagens de programação, executam as atividades de forma sequencial. Mas, às vezes, devemos definir alguns desvios de acordo com as condições estabelecidas no processo de execução do algoritmo conforme a interferência do usuário. Para que isso ocorra, é preciso executar o comando Se(), que permite realizar desvios condicionais com base em testes lógicos. A estrutura dos algoritmos é composta por um teste condicional e por um comando ou conjunto de comandos a serem executados a partir desse teste, mas somente quando o resultado for verdadeiro (figura 7).

Figura 7
Se...Fim-Se.

Se (condição) **Então**

{comando(s)}

Fim-Se



No próximo exemplo, para estabelecer a condição vamos utilizar os operadores lógicos já vistos.

Se (média >= 7.00) **Então**

Escreva("Você está aprovado!!!! ")

Fim-Se

• Estrutura de decisão – Se...Senão...Fim-Se

Podemos organizar o comando Se(), para criar desvios condicionais de verdadeiro ou falso no mesmo teste (figura 8):

Figura 8
Se...Senão...Fim-Se.

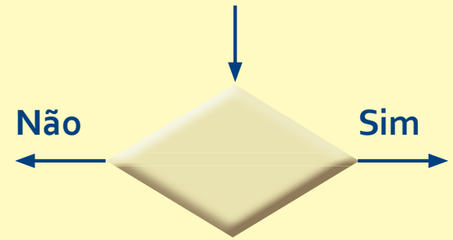
Se (condição) **Então**

{comando(s) condição verdadeira}

Senão

{comando(s) condição falsa}

Fim-Se



Como ficaria a repetição do teste com a variável média.

Se (media >= 7.00) **Então**

Escreva("Você está aprovado!!!! ")

Senão

Escreva("Você está reprovado!!!! ")

Fim-Se

Vamos organizar todo o pseudocódigo, apresentando-o, também, no formato de fluxograma (figura 9).

```
Programa exemplo_teste

Declare

        nome como caractere

        media, nota1, nota2 como flutuante


Início

        Escreva("calculando a média")

        Escreva("Digite o seu primeiro nome.")

        Leia(nome)

        Escreva("Qual a sua nota 01?")

        Leia(nota1)

        Escreva("Qual a sua nota 02?")

        Leia(nota2)

        média ← (nota1 + nota2)/2

        Escreva("Sua nota final é:")

        Escreva(média)

                Se (media >= 7.00) Então

                        Escreva("Aprovado")

                Senão

                        Escreva("Reprovado")

                Fim-Se

        Escreva("Fim de programa")

Fim
```

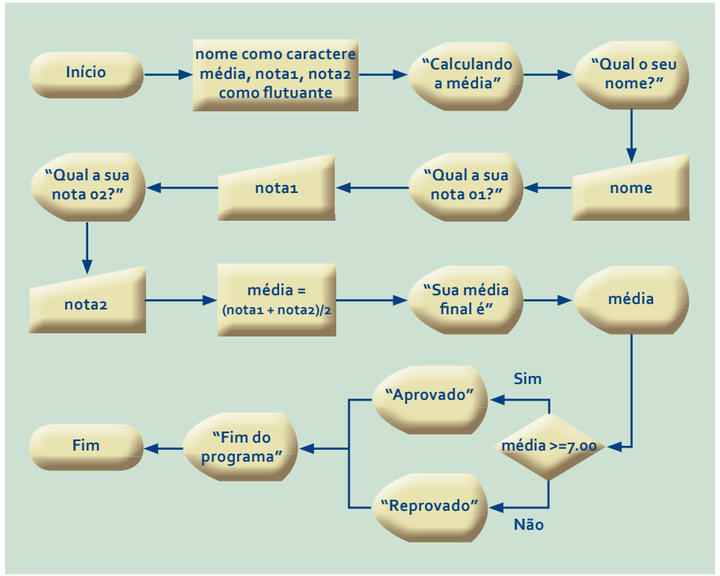


Figura 9
Exemplo de teste.

• Estrutura de decisão – Selecione Caso...Senão...Fim-Selecione

Em alguns momentos, haverá necessidade de testar a mesma variável diversas vezes, como acontece no menu de opções. Para isso, utilizamos o comando "Selecione Caso" da seguinte forma (figura 10):

```
Selecione Caso {variável}

Caso condição 01

        {comando(s)}

Caso condição 02

        {comando(s)}

Caso condição 03

        {comando(s)}

Senão

        {comando(s)}

Fim-Selecione
```

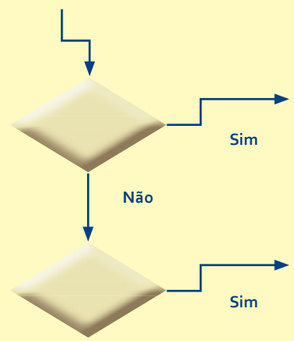
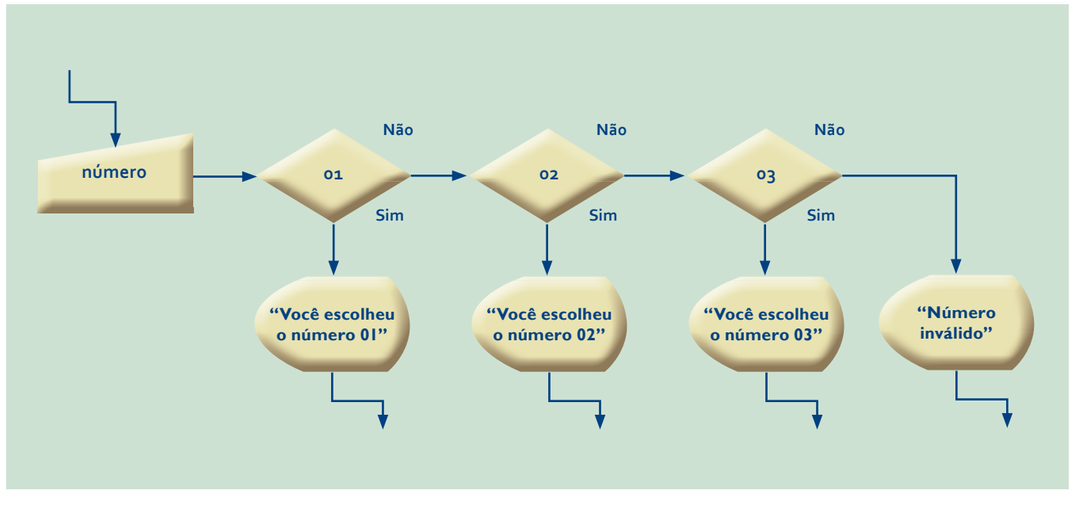


Figura 10
Selecione Caso.

É preciso descobrir, em determinada situação, se o usuário escolheu os números 1, 2 ou 3. Qualquer outro diferente, deverá gerar a mensagem "número inválido" (figura 11):

```
Leia(número)
        Selecione Caso número
            Caso 1
                Escreva("Você escolheu o número 01")
            Caso 2
                Escreva("Você escolheu o número 02")
            Caso 3
                Escreva("Você escolheu o número 03")
            Senão
                Escreva("Número inválido")
        Fim-Selecione
```

Figura 11
Selecione caso.



• Estrutura de repetição – Para...Fim-Para

Além dos desvios sequenciais, é possível criar desvios em *loop*, ou seja, repetir trechos do algoritmo sobre determinada condição e controlar a forma com que os *loops* serão executados. O comando Para...Fim-Para permite que uma variável realize a contagem do número de loops a executar, conforme a indicação inicial e final dessa contagem (figura 12), e também identifique o formato em que essa tarefa será realizada.

```
Para <variável> = <vlr.inicial> Até <vlr.final> Passo <arg> Faça
        {comando(s)}
Fim-Para
```

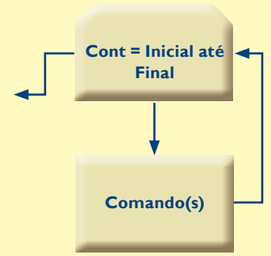


Figura 12
Para...Fim-Para.

Alguns exemplos:

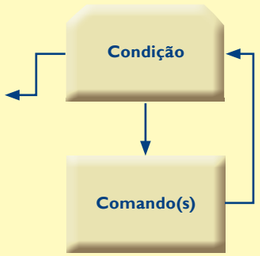
```
Para x = 1 Até 5 Faça
        Escreva(x) // Serão visualizados os números: 1,2,3,4 e 5.
Fim-Para


Para num = 3 Até 6 Faça
        Escreva(num) // Serão visualizados os números: 3,4,5 e 6.
Fim-Para


Para vlr = 0 Até 10 Passo 2 Faça
        Escreva(vlr) // Serão visualizados os números: 0,2,4,6,8 e10.
Fim-Para


Para cont = 3 Até 0 Passo -1 Faça
        Escreva(cont) // Serão visualizados os números: 3,2,1 e 0.
Fim-Para
```

• Estrutura de repetição – Enquanto...Fim-Enquanto

Assim como o comando Para...Fim-Para permite realizar loop, a instrução Enquanto...Fim-Enquanto executa o mesmo processo (figura 13). Mas é preciso lembrar que esse comando não realiza a contagem automaticamente. Por isso, é necessário implementar um contador de passos.

Figura 13
Enquanto...
Fim-Enquanto.

Enquanto (condição) **Faça**
{comando(s)}
Fim-Enquanto



Vamos repetir os exemplos da instrução **Para...Fim-Para**, mas agora com o comando **Enquanto**.

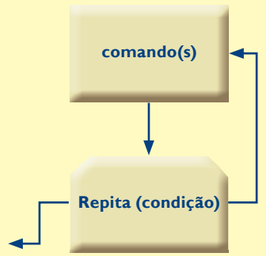
```
x ← 1
Enquanto (x<=5) Faça
        Escreva(x) // Serão visualizados os números: 1,2,3,4 e 5.
        x ← x + 1
Fim-Enquanto


num ← 3
Enquanto (num <=6) Faça
        Escreva(num) // Serão visualizados os números: 3,4,5 e 6.
        num ← num + 1
Fim-Enquanto


vlr ← 0
Enquanto (vlr <=10) Faça
        Escreva(vlr) // Serão visualizados os números: 0,2,4,6,8 e 10.
        vlr ← vlr + 2
Fim-Enquanto


cont ← 3
Enquanto (cont >=0) Faça
        Escreva(cont) // Serão visualizados os números: 3,2,1 e 0.
        cont ← cont -1
Fim-Enquanto
```


• Estrutura de repetição – Repita...Até que

O comando **Repita** tem a mesma finalidade do comando **Enquanto**, quando o assunto é funcionalidade. A única diferença é a localização da condição de teste, que ocorre no final da cadeia de loop (figura 14), garantindo a execução dos comandos no mínimo uma vez.

Figura 14
Repita...Fim-Repita.

Repita
{comando(s)}
Até que (condição)




```
x ← 1
Repita
        Escreva(x) // Serão visualizados os números: 1,2,3,4 e 5.
        x ← x + 1
Até que (x<=5)


num ← 3
Repita
        Escreva(num) // Serão visualizados os números: 3,4,5 e 6.
        num ← num + 1
Até que (x<=6)

vlr ← 0
Repita
        Escreva(vlr) // Serão visualizados os números: 0,2,4,6,8 e 10.
        vlr = vlr + 2
Até que (vlr <=10)


cont ← 3
Repita
        Escreva(cont) // Serão visualizados os números: 3,2,1 e 0.
        cont ← cont -1
Até que (cont >=0)
```

1.6. Vetor

Definido também como matriz unidimensional, um vetor (tabelas 6 e 7) é uma variável que possui vários dados, acessados por meio de uma posição relativa, seguindo a mesma regra da concepção e atribuição das variáveis.

Tabela 6
Vetor de Letras.

1	2	3	4	5	6	7	8	9	10
A	H	M	B	R	J	G	Q	S	P

Tabela 7
Vetor de Números.

1	2	3	4	5	6	7	8	9	10
10	56	-4	60	2	6	99	3	1	-10

No exemplo anterior, podemos verificar que cada elemento de um vetor é caracterizado por uma posição relativa. Observando o vetor de letras, teremos:

```
Posição 1 à valor A

Posição 5 à valor R

Posição 8 à valor Q
```

Em um pseudocódigo, realizamos a declaração de um vetor da seguinte forma:

```
Programa vetores
Declare
        vet como conjunto[n..n1] de <tipo>
Início
        {comandos ou bloco de comandos}
Fim
```

O nome do vetor é representado pela expressão **"vet"**, cujo conjunto de informações é identificado por **"n"**, posição inicial, e **"n1",** posição final, ou seja, um vetor de 10 posições poderá ser representado como [**1..10**]. E o **"tipo"** de informação que este vetor deverá receber.

```
Programa vetores
Declare
        vet como conjunto[1..5] de inteiro
Início
        vet[1] ← 90
        vet[2] ← 1
        vet[3] ← 100
        vet[4] ← 4 - 1
        vet[5] ← 21
Fim
```

Para realizar o acesso à posição de um vetor, é preciso identificar a posição que está entre os sinais de **[]**, logo após o nome do vetor:

```
        Escreva(vet[3])

        Leia(vet[2])
```

• Trabalhando com vetor

A entrada de dados em um vetor é realizada por intermédio de seu identificador. Para isso, utiliza-se um contador crescente ou decrescente, de acordo com a necessidade. Vamos carregar 10 informações numéricas dentro de um vetor.

```
vet como conjunto[1..10] de inteiro
Para i = 1 Até 10 Faça
        Escreva("Digite um valor na posição", i, "do vetor")
        Leia(vet[i])
Fim-Para
```

Seguindo a mesma lógica, podemos realizar a impressão do conteúdo desse vetor.

```
Para i = 1 Até 10 Faça
        Escreva("Valor da posição:", i, "do vetor é", vet[i])
Fim-Para
```

Vamos reunir os conceitos de leitura e impressão no mesmo algoritmo.

```
Programa vetor_dados
Declare
        vet como conjunto[1..10] de inteiro
        i como inteiro
Início
        Para i = 1 Até 10 Faça
                Escreva("Digite um valor na posição", i, "do vetor")
```

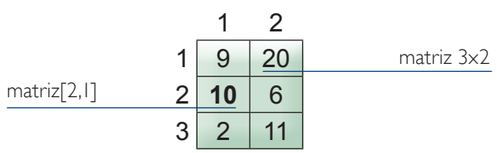
```
            Leia(vet[i])
        Fim-Para
        Para I = 1 Até 10 Faça
            Escreva("Valor da posição:", i, "do vetor é", vet[i])
        Fim-Para
Fim
```

Outro exemplo para analisar:

```
Programa média
Declare
        vetor como conjunto[1..8] de inteiro
        soma, média como flutuante
        ct como inteiro
Início
        soma = 0
        Para ct = 1 Até 8 Faça
            Escreva("Digite um valor na posição", ct, "do vetor")
            Leia(vetor[ct])
            soma ← soma + vetor[ct]
        Fim-Para
        média ← soma / 8      ---- ou ----    média ← soma / ct
        Escreva("A média final é:", média)
Fim
```

1.7. Matriz

A matriz, ao contrário do vetor, é multidimensional. Nesse caso, os dados são armazenados ou lidos de acordo com a relação entre os eixos, e seu índice é representado por dois ou mais valores. Uma matriz 3x2, por exemplo, possui três linhas por duas colunas. Portanto, qualquer referência aos dados deverá apontar para ambas as posições da forma matriz[2,1], indicando que a informação referenciada está na linha 2 e na coluna 1 da matriz. Graficamente, tem-se:



Para implementação, realizamos a declaração da matriz da seguinte forma:

```
Programa matriz
Declare
        mat como conjunto[1..5][1..3] de inteiro
Início
        {comando ou bloco de comandos}
Fim
```

Assim como os vetores, para a atribuição de valores em uma matriz, devemos identificar a linha e a coluna.

```
Programa matriz
Declare
        mat como conjunto[1..2][1..2] de inteiro
Início
        mat[1,1] ← 19
        mat[1,2] ← 2
        mat[2,1] ← 77
        mat[2,2] ← 16
Fim
```

• Trabalhando com matriz

A leitura e a impressão das matrizes seguem o mesmo esquema dos vetores. Mas será necessário utilizar dois contadores: um para identificar a linha e outro, para coluna. Vejamos o exemplo de uma matriz 2x2.

```
    Programa matriz
    Declare
            matriz como conjunto[1..2][1..2] de flutuante
            lin, col como inteiro
    Início
    Para lin = 1 Até 2 Faça
            Para col = 1 Até 2 Faça
                    Escreva("Linha", lin, "da matriz")
                    Escreva("Coluna", col, "da matriz")
                    Escreva("Qual o valor?")
                    Leia(matriz[lin,col])
            Fim-Para
    Fim-Para
    Escreva("Visualizando os dados da matriz:")
    Para lin = 1 Até 2 Faça
            Para col = 1 Até 2 Faça
                    Escreva("Linha", lin, "da matriz")
                    Escreva("Coluna", col, "da matriz")
                    Escreva("O valor é:")
                    Escreva(matriz[lin, col])
            Fim-para
    Fim-para
Fim
```

Outro exemplo:

```
Programa soma
Declare
        a,b,c como conjunto[1..8][1..4] de flutuante
        ln, cl como inteiro
Início
        Para ln = 1 Até 8 Faça
                Para cl = 1 Até 4 Faça
                        Escreva("Digite o valor para a Matriz A:")
                        Leia(a[ln, cl])
                        Escreva("Digite o valor para a Matriz B:")
                        Leia(b[ln, cl])
                        Escreva("Valores somados para a Matriz C:")
                        c[ln, cl] = a[ln, cl] + b[ln, cl]
                        Escreva(c[ln, cl])
                Fim-Para
        Fim-Para
Fim
```

1.8. Programação modular

Uma das técnicas bem empregadas para o desenvolvimento de programas é a modulação de parte do código. Isso significa que o programa principal é quebrado em partes menores. E cada parte representa uma unidade funcional. Essas sub-rotinas são identificadas como procedimentos ou funções. Permitem que parte do código seja reaproveitada em outra parte do código principal ou, até mesmo, em projetos diferentes, por outros programadores.

• Procedimento

Um procedimento, assim como o programa principal, é formado por toda a estrutura lógica (início-fim, variáveis e instruções). Pode ser utilizado em qualquer parte de um programa que seja referenciado. Dessa forma, ao ser acionado, o procedimento assume o controle da execução. E, quando é finalizado, o controle retorna ao programa principal.

```
Procedimento nome-do-procedimento ( parâmetro:tipo )
Declare {variáveis}
        {comando ou bloco de comando
Fim-Procedimento
```

O **nome-do-procedimento** faz a identificação do programa – e é por essa expressão que ele será referenciado. Logo após essa etapa, encontramos os parâmetros e a definição de seu tipo, os quais representam variáveis que receberão valores no momento da chamada. Assim como o código principal, o procedimento possui as variáveis locais de manipulação de dados e os comandos necessários para a sua execução.


```
Programa procedimento
Declare
        op como caractere
Procedimento bomdia
        Escreva("Bom dia!!!!!")
Fim-procedimento
Início
        Escreva("Deseja ver nossa mensagem? (S/N)")
        Leia(op)
        Se op = 'S' Ou op = 's' Então
                bomdia          //  chamada ou procedimento
        Fim-Se
Fim
```

• Variáveis locais ou globais

As variáveis locais são consideradas apenas em uma sub-rotina (procedimento ou função). Dessa forma, são desconhecidas pelo algoritmo principal, ao contrário das globais, que são consideradas por todo o algoritmo.

```
Programa tipos_variáveis
Declare
        a, b como INTEIRO // são variáveis do tipo global
Procedimento teste
        Declare
                x, y como inteiro // são variáveis locais
                // esta área reconhece (a),(b),(x) e (y)
Fim-procedimento
Início
        // esta área reconhece (a) e (b).
Fim
```

O quadro em seguida mostra o campo de visualizações das variáveis, conforme o pseudocódigo apresentado anteriormente.

Programa tipos_variaveis

Visualização das variáveis

(a) e (b)

Procedimento teste

Visualização das variáveis (a),(b),(x) e (y)

Acompanhe um exemplo para o cálculo da área do quadrado usando o procedimento.

```
Programa quadrado
Declare
        op como caractere
Procedimento quadrado
        Declare
                lado, área como inteiro
        Escreva("Digite um número inteiro:")
        Leia(lado)
        área ← lado * lado
        Escreva("A área do quadrado é:", área)
Fim-Procedimento
Início
        Escreva("Gostaria de saber a área de um quadrado (S/N)?")

        Leia(op)
        Se op = 'S' Ou op = 's' Então
                quadrado
        Fim-Se
Fim
```

• Passagem de parâmetro

A passagem de parâmetros por valor é utilizada para que as sub-rotinas possam receber informações e realizar qualquer tipo de manipulação fora do pseudocódigo principal.

```
Programa quadrado
Declare
        op como caracter
        lado como inteiro

Procedimento quadrado (ld como inteiro)
        Declare
                area como inteiro
        área ← ld * ld
        Escreva("A área do quadrado é:", area)
Fim-Procedimento
Início
        Escreva("Gostaria de saber a área de um quadrado (S/N)?")
        Leia(op)
        Se op = 'S' Ou op = 's' Então
                Escreva("Digite o valor:")
                Leia(lado)
                quadrado(lado)

        Fim-Se
Fim
```

• Função

Uma função tem as mesmas características estruturais de um procedimento, no que diz respeito a fatores como funcionalidade, visualização de variáveis e chamadas. Mas a sua principal vantagem é a possibilidade de retorno de valores em tempo real. Isso quer dizer que, quando chamamos determinada função, podemos receber valores em tempo real, armazenando o resultado em uma variável.

```
Função nome-da-função ( parâmetros:tipo ) : <tipo-de-retorno>
Declare {variáveis}
        {comando ou bloco de comandos}
Fim-Função
```

Diferentemente do procedimento, é preciso fazer a identificação do tipo de valor a ser retornado.

```
Programa quadrado
Declare
        op como caractere
        lado como inteiro

Função quadrado (ld como inteiro): inteiro
        Declare
                área como inteiro
        área ← ld * ld
Fim-Função
Início
        Escreva("Gostaria de saber a área de um quadrado (S/N)?")
        Leia(op)
        Se op = 'S' Ou op = 's' Então
                Escreva("Digite o valor:")
                Leia(lado)
        Escreva("A área do quadrado é:")
        Escreva(quadrado(lado))
        Fim-Se

Fim
```