IFRS – Instituto Federal de Educação Ciência e Tecnologia do

Rio Grande do Sul – Câmpus Porto Alegre

Lógica de Programação

Curso de Redes de Computadores EAD

Prof.ª Fabrícia Py Tortelli Noronha

Atualizada em Abril/2013

O que é Lógica?

"Estudo das leis do raciocínio." (Dicionário Luft)

"Conjunto de regras e princípios que orientam, implícita ou explicitamente, o desenvolvimento de uma argumentação ou de um raciocínio, a resolução de problemas..." (Dicionário Aurélio)

A lógica faz parte do dia-a-dia do ser humano. Qualquer tarefa realizada possui uma sequência de passos lógicos, chamada **algoritmo**. Um algoritmo nada mais é do que uma instrução de como solucionar um problema. Um problema pode ter não apenas uma solução, mas várias.

> **Exemplo:** O preparo de uma vitamina.
>
> Ingredientes: leite, frutas, cereal e gelo.
>
> Modo de Preparo: coloque no liquidificador o leite, as frutas e o gelo. Bata por três minutos. Por último coloque o cereal e bata por mais um minuto.
>
> Está pronta a vitamina!

O exemplo a cima descreve um algoritmo executado naturalmente por um indivíduo que deseja ao final ter uma vitamina pronta. Essa não é a única maneira de se fazer uma vitamina, existem outras receitas, com outras formas de preparo e com outros ingredientes que também chegam ao mesmo final, a vitamina pronta.

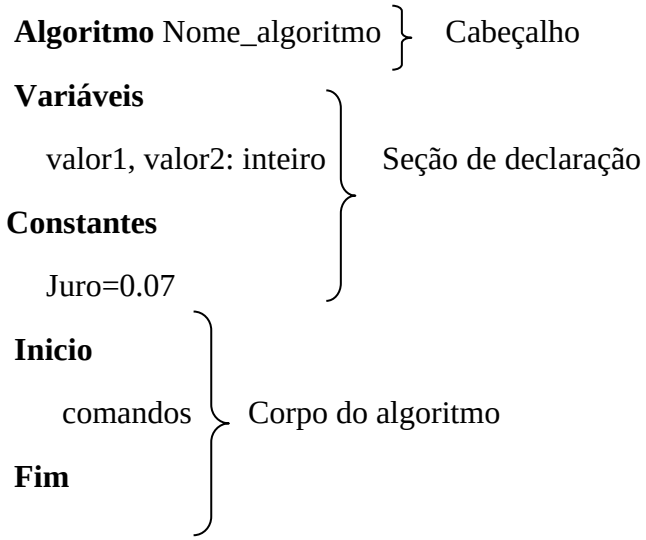
Nesta apostila iremos tratar sobre o algoritmo computacional que difere do algoritmo natural por obedecer a normas de sintaxe (comandos) e semântica (ações dos comandos). Utilizaremos como forma de representação o português estruturado, também chamado de portugol, e o fluxograma.

As linguagens de programação, na sua grande maioria, são escritas na língua inglesa. O português estruturado ou portugol é uma pseudolinguagem que utiliza comandos escritos em língua portuguesa, o que facilita o aprendizado.

Todo algoritmo em português estruturado deve conter **um cabeçalho, uma seção de declaração e um corpo do algoritmo.**

- Cabeçalho: deve começar com a palavra "Algoritmo", seguida de um nome que dê uma idéia do objetivo final do algoritmo.
- Seção de Declaração: é o lugar onde devem ser declaradas as variáveis e as constantes utilizadas durante a execução do algoritmo.
- Corpo do Algoritmo: delimitado pelas palavras início e fim. É onde se encontram os comandos que serão executados no algoritmo.

Exemplo:



Variáveis

As variáveis são utilizadas no decorrer de um algoritmo para armazenar dados na memória. Cada variável armazena um único dado por vez. Quando o conteúdo sofre alguma alteração, o anterior é perdido.

As variáveis devem receber um nome, tipo de dado e um conteúdo.

Nome:

- Iniciar por letra de (a...z, A...Z). Depois pode ser seguido por letras, números ou underline;
- Não pode ser palavra reservada, ou seja, não usar palavras que representem comandos da linguagem;
- Não pode ser caracter especial e nem espaço. Ex: @, #, $, *
- Não é *case sensitive*, ou seja, não existe distinção entre maiúsculas e minúsculas. Ex: valor e VALOR
- Nomes de variáveis devem ter no máximo 127 caracteres.

Conteúdo:

- Dado que vai ser armazenado na variável.

Tipo de dado:

- Numérico: Inteiro (45, -5,...) Ex: idade: inteiro

  Real (1.9, -3.5) Ex: nota: real

- Caracter (letras, números representativos e símbolos. Devem vir sempre entre aspas duplas). Ex: cpf :caracter

- Lógico ou Booleano (verdadeiro ou Falso).

OBS: variáveis com o mesmo tipo de dado podem ser declaradas uma ao lado da outra, separadas por vírgula.

Ex: nome, cpf: caracter

Constantes

Uma constante é um valor que permanece inalterado durante o processamento do algoritmo. Recebe um nome que obedece as mesmas regras da variável (ver página anterior Variáveis – Nome)

Ex: Constantes Ano=2014

Juro=0.05

Para que possamos realizar operações dentro do algoritmo é necessário conhecer alguns operadores abaixo relacionados:

Operadores Matemáticos

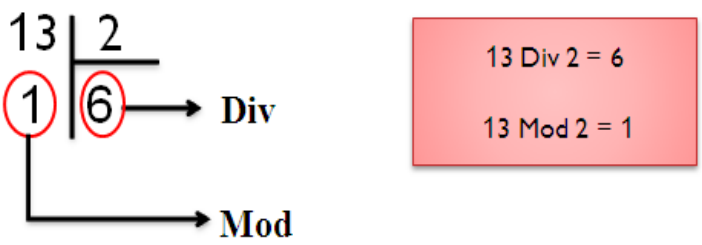
+	Adição
–	Subtração
*	Multiplicação
/	Divisão
^	Potenciação

Operadores Especiais

DIV – operador de divisão inteira

MOD – operador de resto da divisão inteira

EX:



Operadores Relacionais

Maior que >

Menor que <

Maior ou igual >=

Menor ou igual <=

Igual =

Diferente < >

Operadores Lógicos

E: Retorna verdadeiro se ambas as partes forem verdadeiras

OU: Basta que uma parte seja verdadeira para retornar verdadeiro.

NÃO: Inverte o estado

Tabela verdade

P	Q	P e Q	P ou Q	Não P
V	V	V	V	F
V	F	F	V	F
F	V	F	V	V
F	F	F	F	V

Algumas funções pré-definidas

ABS(x) – Retorna o valor absoluto de x

SQR(x) – Eleva x ao quadrado

SQRT(x) – Raiz quadrada de x

INT(x) – Retorna a parte inteira de x

FRAC(x) – Retorna a parte fracionária de x

ROUND(x) – Retorna o valor arredondado de x

LOG(x) – Retorna o logaritmo de x

Precedência de Operadores

Ordem	Operador
1º	( )
2º	Não, funções, ^
3º	*, /, Div, Mod
4º	=, < >, <, >, <=, >=
5º	E
6º	OU

Comando de Atribuição

Utilizado para atribuir um valor a uma variável. O valor atribuído a uma variável deve ser sempre do mesmo tipo definido na seção de declaração.

Ex: Nome ← 'Maria'

Soma ← valor1 + valor2

A atribuição é representada pela seta virada para a esquerda. A leitura dos exemplos acima é feita da seguinte maneira:

- Nome recebe Maria
- Soma recebe valor um mais valor dois

Comando Saída de Dados

O comando escrever é utilizado para imprimir informações na tela que devem estar entre aspas simples. Usa-se vírgula para concatenar o valor de uma variável com um texto explicativo.

EX1: escrever ('Informe um valor')

Significa que será impresso na tela a mensagem **Informe um valor**.

EX2: escrever ('O nome do funcionário é:', nome)

Significa que será impresso na tela a mensagem **O nome do funcionário é:** acrescido do conteúdo que estiver armazenado na variável nome. Usa-se vírgula para concatenar o valor de uma variável com um texto explicativo.

Comando de Entrada de Dados

O comando ler armazena informações do teclado em uma variável.

EX: ler(valor)

Significa que o usuário vai digitar um número e que este número será armazenado em uma variável chamada valor.

Comentários

São textos explicativos sobre determinadas linhas de código que não interferem no programa. São de extrema importância especialmente em códigos mais complexos. Devem aparecer entre chaves ou parênteses com asteriscos.

EX: {imprime o nome do funcionário} ou (*imprime o nome do funcionário*)

Cuidado!!! Comentar é diferente de Frasear

Comentando: **total←valorunitario*quantidade** {calcula o total à pagar}

Fraseando: **total←valorunitario*quantidade**{atribui à total o valor unitário vezes a quantidade}

Teste de Mesa

É a execução do algoritmo passo-a-passo, como se ele fosse executado pelo computador mostrando a evolução do conteúdo das variáveis.

EX: Algoritmo que leia as duas notas de um aluno e calcule a média.

```
Algoritmo calcula_media
Variáveis
        nota1, nota2, media: real
Inicio
    escrever ('Informe nota1') {aparece na tela Informe nota1}
    ler (nota1) {armazena a nota digitada na variável nota1}
    escrever ('Informe nota2') {aparece na tela Informe nota2}
    ler (nota2) {armazena a nota digitada na variável nota2}
    media ←(nota1+ nota2)/2 {calcula a média}
    escrever ('Média:', media) {aparece na tela a palavra Média: e o valor
                                atribuído a variável média}

Fim
```

Teste de Mesa

nota1	nota2	media
8	6	7

Identação

É o recuo estabelecido nas estruturas de controle.

Estético? NÃO

Fundamental para a legibilidade do programa. Contribui para o desenvolvimento e entendimento do algoritmo.

Estruturas de Seleção

Comandos de Seleção permitem que determinados comandos sejam executados se uma determinada condição for satisfeita ou não.

- **Se .. então**

  senão

- **Caso .. seja**

  Senão {opcional}

  Fim

Se condição então
Comando

Vamos a um exemplo prático: Criar um algoritmo que lê dois valores, calcula a soma e mostra na tela. Ao final, apresenta uma mensagem, **"Está no intervalo dos números maiores ou iguais a 100"** se a soma for maior ou igual a 100 (cem) ou "**Está no intervalo dos números menores que 100"**, caso a soma seja menor que 100(cem).

```
Algoritmo soma
 Variaveis
        valor1, valor2, soma:  real
 inicio
        escrever('Informe um valor')
        ler(valor1)
        escrever('Informe outro valor')
        ler(valor2)
        soma←valor1+valor2
        escrever('Soma:',soma)
        se soma > = 100 então
           escrever('Está no intervalo dos números maiores ou iguais a 100')
        se soma < 100 então
           escrever('Está no intervalo dos números menores que 100')
fim
```

```
Se condição então
    Comando1
Senão comando2
```

Utilizando o mesmo exemplo ficaria:

```
Algoritmo soma
 Variaveis
        valor1, valor2, soma:  real
 inicio
        escrever('Informe um valor')
        ler(valor1)
        escrever('Informe outro valor')
        ler(valor2)
        soma←valor1+valor2
        escrever('Soma:',soma)
        se soma > =100 então
           escrever('Está no intervalo dos números maiores ou iguais a 100')
        senão
           escrever('Está no intervalo dos números menores que 100')
fim
```

Fazendo uma análise sobre os dois algoritmos apresentados a diferença que ocorre quando **se utiliza o comando senão** é que, caso a soma seja maior ou igual a 100(cem) é impresso na tela a mensagem "Está no intervalo dos números maiores ou iguais a 100" e o programa é finalizado, sem testar a próxima linha de comando.

Já no algoritmo anterior, que **não utiliza o senão**, se o número for maior ou igual a cem também escreve a mesma mensagem, mas segue executando a próxima linha do algoritmo para testar se o número é menor do que cem. Perda de tempo!!!

```
Se condição então
inicio
    comando1
    coamando2
fim
Senão
inicio
    comando3
    comando4
fim
```

Utilizando o mesmo exemplo e acrescentado mais duas linhas de comando em cada estrutura de decisão é necessário delimitar o bloco de comandos através das palavras inicio e fim, pois o comando **Se** executa somente uma linha de comando abaixo dele.

```
…

se soma >= 100 então
inicio
     escrever(‘Está no intervalo dos números maiores ou iguais a 100’)
     escrever(‘O número tem no mínimo três algarismos’)
fim
senão
inicio
     escrever(‘Está no intervalo dos números menores que 100’)
     escrever(‘O número tem no máximo dois algarismos)
fim
fim
```

```
Se condição então
    comando1
Senão se condição então
        comando2
    senão
        comando3
```

Seleção Encadeada

Modificando um pouquinho o exemplo anterior, criar um algoritmo que lê dois valores, calcula a soma e mostra na tela. Ao final, apresenta uma mensagem:

“Está no intervalo dos números maiores ou iguais a 1000” se a soma for maior ou igual a 1000;
“**Está no intervalo de 100 a 999”** caso a soma seja maior ou igual a 100 até 999;
“**Está no intervalo de 0 a 99**” se a soma for menor ou igual a 99.

```
Algoritmo soma
 Variaveis
        valor1, valor2, soma:  real
 inicio
        escrever(‘Informe um valor’)
        ler(valor1)
        escrever(‘Informe outro valor’)
        ler(valor2)
        soma←valor1+valor2
        escrever(‘Soma:’,soma)
        se soma > =1000 então
          escrever(‘Está no intervalo dos números maiores ou iguais a 1000’)
        senão se (soma < 1000) e (soma >= 100) então
           escrever(‘Está no intervalo de 100 a 999’)
        senão
           escrever(‘Está no intervalo de 0 a 99’)
fim
```

Toda vez que for utilizada uma seleção encadeada, **Senão Se,** a próxima condição somente será testada se a anterior não for satisfeita.

```
Caso variável seja
   valor:comando1
   valor,valor: comando2
   valor..valor: comando3
Senão comando4 {opcional}
fim
```

Colocando em prática.

Fazer um algoritmo que leia um número de 1 a 7 e retorne o dia da semana correspondente. Caso o usuário informe um número diferente retornar a mensagem:Dia inválido.

```
Algoritmo dias_semana
Variaveis
        dia:  caractere
 inicio
        escrever('Informe um número de 1 a 7')
        ler(dia)
        Caso dia seja
           "1":escrever('Domingo')
           "2":escrever('Segunda')
           "3":escrever('Terça')
           "4":escrever('Quarta')
           "5":escrever('Quinta')
           "6":escrever('Sexta')
           "7":escrever('Sábado')
        senão escrever('Dia inválido')
        fim
fim
```

O comando **senão** somente será executado se nenhuma das opções anteriores for satisfeita e seu uso pode ser opcional, conforme a necessidade do algoritmo.

Estruturas de Repetição

Laço Condicional:

- Pode ser desconhecido ou não o número de vezes que os comandos no interior da estrutura serão executados. Amarrado a uma condição.

  Enquanto .. Faça (laço condicional com teste no início)

  Repetir .. Até (laço condicional com teste no final)

Laço Contado:

- Conhecimento prévio de quantas vezes os comandos no interior da estrutura serão executados.

  Para .. até .. faça
  (Dotado de mecanismos para contar o número de vezes que o laço será executado)

Para utilizar estruturas de repetição é necessário o conhecimento prévio de dois conceitos:

Contador

Acumula somas constantes, ou seja, cresce de valor em intervalos constantes.

Ex: cont ← cont + 1
numero ← numero + 2

Acumulador

Variável que acumula a soma de outras variáveis.

Ex: soma ← soma + valor
media_total← media_total + media

Enquanto .. faça **comando**	**Enquanto .. faça** **inicio** **comando1** **comando2** **...** **fim**

Ex: Elaborar um algoritmo que leia o salário de cada um dos 150 funcionários de uma empresa e a categoria a qual pertence. Calcule o novo salário de acordo com o aumento concedido a cada categoria conforme a tabela que segue:

Categoria	**Aumento**
A	8%
B	6%
C	3%

```
Algoritmo novo_salario
 Variaveis
        sal, novo_sal,:  real
        categoria: caracter
        cont:inteiro
 inicio
        cont ←1
        enquanto cont <= 150 faça
        inicio
                escrever('Informe o salário')
                ler(sal)
                escrever('Informe a categoria a qual pertence')
                ler(categoria)
                caso categoria seja
                   "A": novo_sal ← sal * 0.8
                   "B": novo_sal ← sal * 0.6
                   "C": novo_sal ← sal * 0.3
                 fim
                 escrever('Seu novo salário é:', novo_sal)
                cont ← cont + 1
        fim
fim
```

Nesse algoritmo foi utilizada uma estrutura de repetição que testa antes de executar. Para isso foi utilizado um contador, chamado cont, de forma que o programa seja executado as 150 (cento e ciquenta) vezes, que corresponde ao número de funcionários da empresa. A cada nova execução é acrescentado 1 (um) ao contador, através da linha de comando **cont ← cont + 1**.
Enquanto a condição do comando **enquanto** for verdadeira, segue executando o algoritmo. Quando a condição for falsa, o laço de repetição será encerrado.

```
Repetir
   comando1
   comando2
      ...
até
```

Utilizando o mesmo exemplo teremos:

```
Algoritmo novo_salario
 Variaveis
        sal, novo_sal,:  real
        categoria: caracter
        cont:inteiro
 inicio
        cont ←1
```

```
repetir
        escrever('Informe o salário')
        ler(sal)
        escrever('Informe a categoria a qual pertence')
        ler(categoria)
        caso categoria seja
           "A": novo_sal ← sal * 0.8
           "B": novo_sal ← sal * 0.6
           "C": novo_sal ← sal * 0.3
        fim
        escrever('Seu novo salário é:', novo_sal)
        cont ← cont + 1
ate cont > 150
fim
```

Nesse algoritmo foi utilizada uma estrutura de repetição que tem uma entrada vazia, pois não há teste no início. O teste é feito no final da primeira execução, na linha que tem o comando **ate cont > 150**.

Enquanto .. Faça	Repetir / Até
Precisa de início e fim se for bloco de comandos	Não precisa de início e fim. O bloco de comandos é delimitado pelo Repetir e o até.
Pode não executar o(s) comando(s)	Entrada vazia. Executa o(s) comando(s) pelo menos 1 vez
Teste no início	Teste no final
Executa o comando ou bloco de comandos se condição **Verdadeira**	Executa o comando ou bloco de comandos se condição **Falsa**
Condição Falsa: Encerra o laço	Condição Verdadeira: Encerra o laço

Para .. ate .. faça **comando**	**Para .. ate .. faça** **inicio** **comando1** **comando2** **...** **fim**

A estrutura de repetição **Para** incrementa automaticamente a variável de controle (incremento+1). A variável deve ser do tipo inteiro (ver variável **cont** no próximo exemplo).

Ex: Criar um algoritmo em português estruturado que receba o nome e o ano de nascimento dos 80 funcionários de uma empresa e ao final mostre na tela a média de idade dos funcionários.

```
Algoritmo media_idade
Constantes
        ano_atual=2014
 Variaveis
        total_idade:  real
        nome: caracter
        ano, idade,cont:inteiro
 inicio
        total_idade ← 0
        para cont ←1 ate 80 faça
        inicio
                escrever('Informe seu nome')
                ler(nome)
                escrever('Informe seu ano de nascimento')
                ler(ano)
                idade ← ano_atual -  ano
                total_idade ← total_idade + idade
         fim
         escrever('Média de Idade dos funcionários:', total_idade/80)
 fim
```

Observe que a estrutura de repetição para é um laço contado, ou seja, é sabido o número de vezes que o algoritmo será executado.

PARA REFLETIR

Imagine se o algoritmo acima pedisse que ao final fossem listados os nomes e as respectivas idades dos 80 funcionários da empresa e a média final.

O QUE FAZER???

Criar 80 variáveis diferentes para receber nome e 80 para receber idade???

Variaveis nome1, nome2, nome3, nome4, nome5, ..., nome80 : caracter

idade1, idade2, idade3, idade4, idade5, ... , idade80 : inteiro

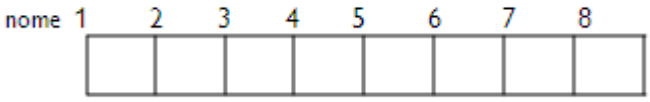
Nesses casos é que precisamos utilizar **vetores.**

Um vetor é uma variável homogênea, unidimensional formada por uma sequência de variáveis, todas do mesmo tipo, com o mesmo nome.

VETOR

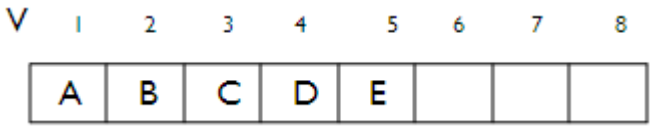
Declaração

Variáveis nome: vetor [1..8] de caracter



Para acessar o conteúdo do vetor é necessário determinar a posição ou índice do elemento no vetor.

nome[posição]



v [3] o retorno será**? C**

v[2*2] o retorno será**? D**

v[6] ← v[2] o vetor na posição 6 recebe**? B**

x ← 5

v[x] o retorno será**? E**

LEITURA DO VETOR

escrever (‘informe 10 valores’)
ler (v[1], v[2], v[3], ..., v[10])

Não se usa

escrever (‘informe 10 valores’)
para i ← 1 até 10 faça
ler (v[i])

Forma Correta

ou

```
para i ← 1 até 10 faça
   Início
       escrever (‘informe valor’)
      ler (v[i])
```

Forma Correta

O índice **i**, na estrutura de repetição **para**, vai variar de 1 até 10 preenchendo assim as 10 posições do vetor.

EX: Fazer um algoritmo que leia o nome e o ano de nascimento dos 80 funcionários de uma empresa e ao final liste os nomes e as respectivas idades dos 80 funcionários da empresa e a média final.

```
Algoritmo media_idade
Constantes
      ano_atual = 2014
 Variaveis
       total_idade,:  real
       ano,cont:inteiro
       nome: vetor[1..80] de caracter
       idade: vetor[1..80] de inteiro

 inicio
       total_idade ← 0
       para cont ←1 ate 80 faça
       inicio
              escrever(‘Informe seu nome’)
              ler(nome[i])
              escrever(‘Informe seu ano de nascimento’)
              ler(ano)
              idade[i] ← ano_atual -  ano
              total_idade ← total_idade + idade[i]
        fim
        para cont ←1 ate 80 faça
        inicio
             escrever(‘Nome:’,nome[i])
             escrever(‘Idade:’,idade[i])
        fim
        escrever(‘Média de Idade dos funcionários:’, total_idade/80)
 fim
```

MATRIZ

É um vetor com mais de uma dimensão, ou seja, possui referência de linha e referência de coluna. Funciona exatamente igual ao vetor exceto por ter um índice a mais (linha e coluna).

Declaração:

M : matriz [1..3,1..3] de inteiro

1º dimensão: Linha 2º dimensão: Coluna

EX:

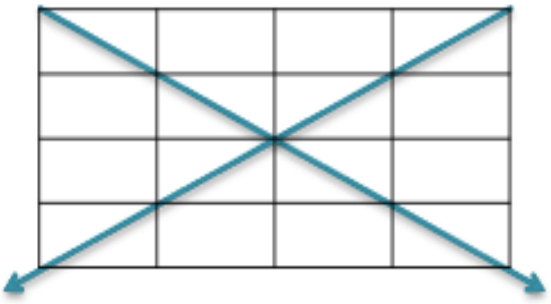
M	1	2	3
1	7	2	3
2	1	2	8
3	4	6	1

L C

M [1,2] ← 5

M [3,3] ← M [3,2] + 3

M	1	2	3
1	7	5	3
2	1	2	8
3	4	6	9



Diagonal Secundária

Diagonal Principal

M [1,1]	M [1,4]
M [2,2]	M [2,3]
M [3,3]	M [3,2]
M [4,4]	M [4,1]

LEITURA DA MATRIZ

EX: Ler uma matriz 5x5.

```
escrever('Matriz 5x5:')
para j ←1 até 5 faça
   ler ( M [1, j ])
para j ←1 até 5 faça
   ler ( M [2, j ])
para j ←1 até 5 faça
   ler ( M [3, j ])
para j ←1 até 5 faça
   ler ( M [4, j ])
para j ←1 até 5 faça
   ler ( M [5, j ])
```

Tabela 1

```
Escrever('Matriz 5x5:')
para i ←1 até 5 faça
    para j ←1 até 5 faça
        ler ( M [i, j ])
```

Tabela 2

Tanto a forma de leitura da matriz na tabela 1 quanto a forma da tabela 2, estão corretas.

Repare na diferença de linhas de código que cada uma gerou. Essa leitura foi para uma matriz 5x5. Imagine se fosse 30x30! Por isso, é que utilizamos a segunda forma de leitura em que são utilizadas duas estruturas de repetição (para). Uma para percorrer a linha da matriz e a outra para percorrer a coluna.

EX: Escreva um programa que leia uma matriz M (5,5) e calcule as somas:

a) da linha 4 de M
b) da coluna 2 de M
c) da diagonal principal
d) da diagonal secundária
e) de todos os elementos da matriz M

Escreva essas somas e a matriz.

```
Algoritmo matriz_5x5
Var M : matriz[1..5,1..5] de inteiro
      j , i , somaA, somaB, somaC, somaD, somaE : inteiro
inicio
         para j ←1 ate 5 faça
           para i ←1 ate 5 faça
              ler(M[j,i])
         somaA ← 0
         somaB ← 0
         somaC ← 0
         somaD ← 0
         somaE ← 0
         para j←1 ate 5 faça
         inicio
            somaA ←somaA +M[4,j]
            somaB ←somaB +M[j,2]
            somaC ←somaC +M[j,j]
            somaD ←somaD +M[j,6-j]
         fim
         para i←1 ate 5 faça
          para j←1 ate 5 faça
             somaE ←somaE + M[i,j]
         escrever('Soma da linha 4', somaA)
         escrever('Soma da coluna2', somaB)
         escrever('Soma da Diagonal Principal', somaC)
         escrever('Soma da Diagonal Secundária', somaD)
         escrever('Soma de todos os elementos da Matriz', somaE)
         escrever ('Matriz 5x5')
         para i←1 ate 5 faça
           para j←1 ate 5 faça
              escrever(M[i,j]
fim
```

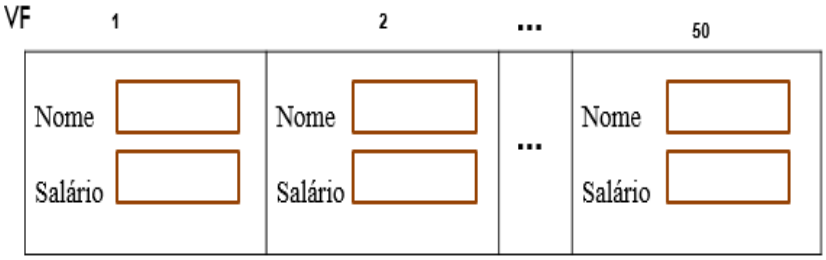
REGISTRO

O vetor do tipo registro possui um ou mais campos que podem ser de tipos de dados diferentes armazenados em uma mesma posição.

EX:

```
VF : vetor[1..50] de registro
                        nome: caracter
                        salário: real
                    fim
```

- Variável do tipo registro é uma ficha
- Vetor do tipo registro é um fichário cheio de fichas



Ler (VF[1].nome)

Posição (índice) **Campo**

Lê o campo nome da ficha na posição 1.

Qual a diferença entre Vetor e Vetor do Tipo Registro???

- **Vetor**: Estrutura homogênea. Variável tem o mesmo tipo de dado. Só caracter, só real, só inteiro, ...

- **Vetor do tipo registro**: Pode-se armazenar em cada registro várias informações de diferentes tipos de dados. Em uma mesma posição, podemos ter variáveis de tipos diferentes como no exemplo citado anteriormente (nome é caracter e salário real).

EX: Criar um algoritmo que leia 80 nomes de funcionários de uma empresa e suas respectivas idades e ao final imprima na tela o nome e a idade de cada funcionário.

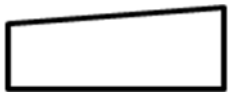
```
Algoritmo  nome_idade
variaveis
        Funcionarios: vetor [1..80] de registro
                                        nome:caracter
                                        idade:inteiro
                                    fim
        i:inteiro
 inicio
      escrever('Informe Nome e Idade')
      para i ← 1 até 80 faça
          ler (Funcionarios[i].nome, Funcionarios[i]. idade)
      para i ← 1 até 80 faça
          escrever (Funcionarios[i].nome,':', Funcionarios[i]. idade,'anos')
 fim
```

Perceba que na declaração do vetor do tipo registro **Funcionarios** são definidas as variáveis que serão utilizados nos campos do vetor (nome e idade) e logo após, é colocado um **fim** para separar as variáveis que fazem parte do registro da que faz parte do algoritmo (i).

Fluxograma

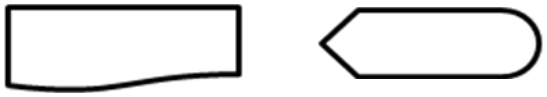
Composto por formas geométricas que representam os processos envolvidos na lógica do algoritmo. Abaixo, veja alguns dos símbolos mais utilizados.

Utilizado para representar o início e o fim de um algoritmo

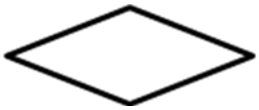


Representa uma entrada de dados que será armazenado em uma variável.

Representa uma atribuição onde uma variável recebe um valor ou uma expressão.



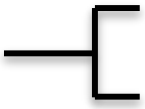
Representa uma saída de dados, ou seja, retorna dados na tela do computador ou para a impressora.



Representa uma decisão. Através da decisão é definido o fluxo do algoritmo.

Representa a direção do fluxo do algoritmo.

Conecta linhas de diferentes direções conforme o fluxo do algoritmo.



Representa um comentário sobre o código utilizado no algoritmo.

Conector de fluxo em outra página. Utilizado quando o fluxograma necessita continuar em outra página

Colocando em Prática

EX: Fazer um algoritmo representado em fluxograma que some dois valores informados pelo usuário e ao final retorne a soma dos valores.

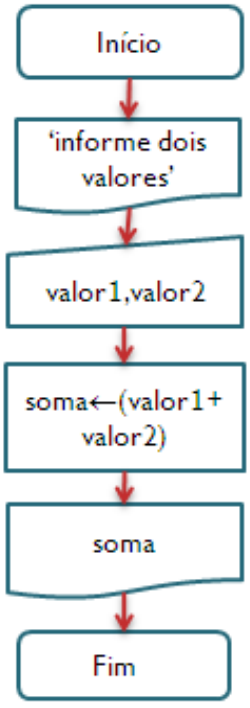


Figura 1

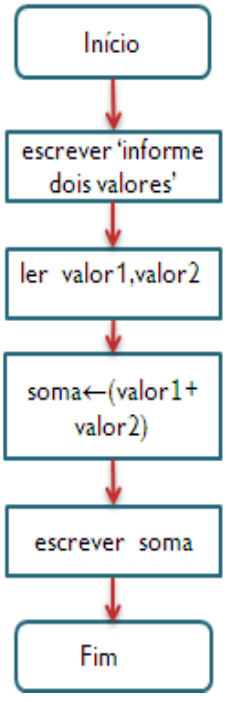


Figura 2

Na figura 1 representamos o algoritmo através das formas geométricas específicas para cada processo lógico envolvido.

Na figura 2, com exceção do início e fim, todos os processos lógicos do fluxograma vêm acompanhados do comando que será executado (escrever, ler, ...), pois a forma geométrica utilizada foi a mesma para todos, o retângulo.

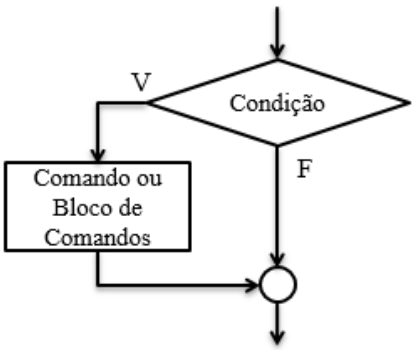
Agora vamos fazer o mesmo algoritmo em português estruturado.

Algoritmo soma_valores
Variáveis
valor1, valor2, soma: inteiro
Inicio
escrever('Informe valor1')
ler(valor1)
escrever('Informe valor2')
ler(valor2)
soma ←(valor1+ valor2)
escrever('Soma:', soma)
Fim

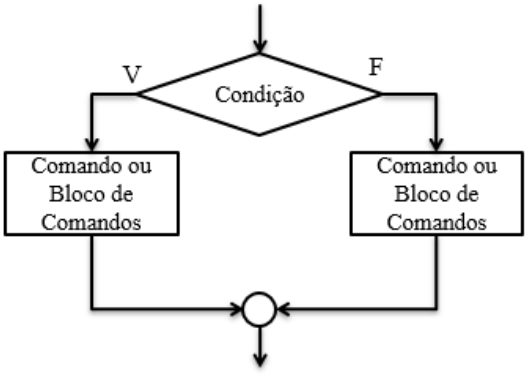
Repare que no português estruturado o algoritmo começa pelo cabeçalho (nome do algoritmo) e logo após a seção de declaração de variáveis. No fluxograma essas duas partes não aparecem. O fluxograma é uma forma mais enxuta de representar o algoritmo.

Estruturas de Repetição

Se condição **então**
comando

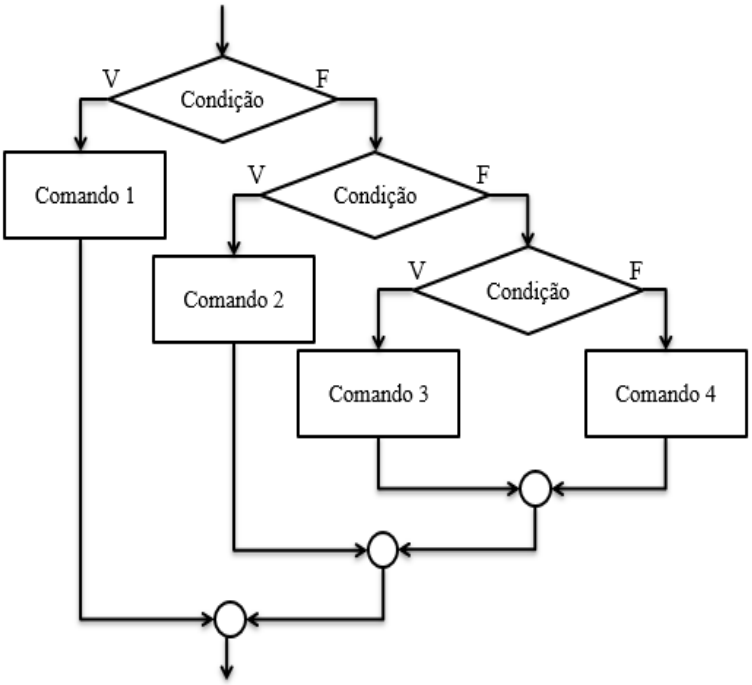


Se condição **então**
comando1
Senão comando2

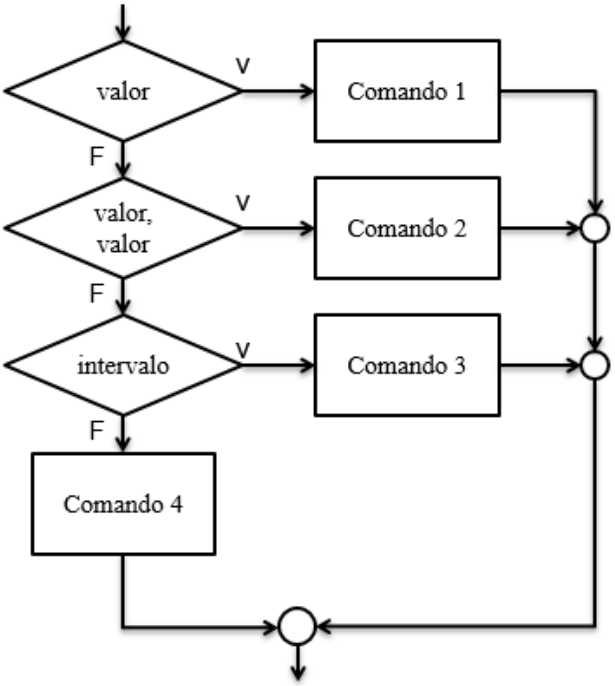


Seleção Encadeada

Se condição **então**
comando1
Senão Se condição **então**
comando2
Senão Se condição **então**
comando3
Senão comando4

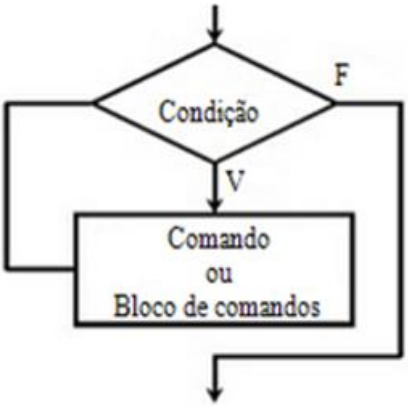


Caso variável **seja**
valor: comando 1
valor, valor: comando 2
valor..valor: comando 3
senão comando 4 {opcional}
fim



Estruturas de Repetição

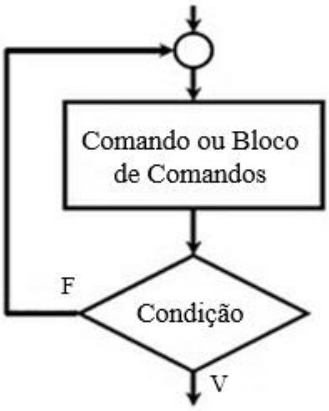
Enquanto condição **faça**
comando



Repetir

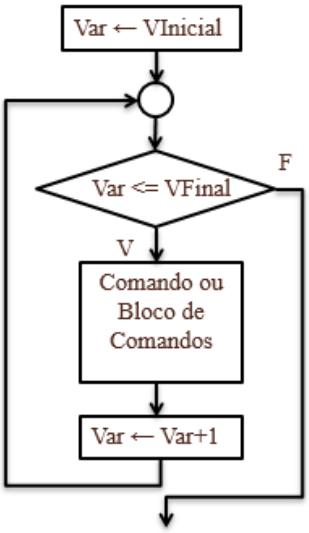
comando

Até condição



Para var ← valor_inicial **ate** valor_final **faça**

comando



No fluxograma, a estrutura de repetição **para** é igual a estrutura **enquanto** pois a variável de controle não tem incremento automático. É necessário utilizar um contador. Ex: var ← var+1