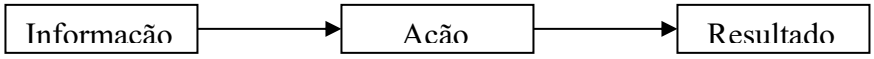
ALGORÍTMOS
INTRODUÇÃO A LÓGICA DE PROGRAMAÇÃO

1 - INTRODUÇÃO E DEFINIÇÃO

1.1 Introdução a Lógica de Programação

É um método pelo qual se aplica o fundamento do Raciocínio Lógico em desenvolvimento de programas de computador, fazendo uso ordenado dos elementos básicos suportados por um dado estilo de programação.

Usar o Raciocínio Lógico no desenvolvimento de programas consiste em:
- ✓ Ler atentamente o enunciado
- ✓ Retirar do enunciado a relação das entradas de dados
- ✓ Retirar do enunciado a relação das saídas de dados
- ✓ Determinar as ações que levarão a atingir o resultado desejado



1.2 Algoritmos

Podemos pensar num algoritmo como um "mecanismo" de transformação de entradas em saídas. Assim, um algoritmo ao ser "executado", receberá algumas entradas, que serão processadas e nos devolverá saídas esperadas.

ALGO*: *Lógica ***RÍTMO*: *Estudo*** → ***ALGORÍTMO* =** ESTUDO DA LÓGICA

→ Aplicação computacional do Estudo da Lógica para se alcançar um objetivo.

→ Linguagem em alto nível que permite ao usuário/programador preparar e programar o computador para resolver problemas através de uma seqüência lógica de ações que deverão ser executados passo-a-passo e que seguirão os conceitos da programação estruturada.

Um algoritmo é um texto estático, onde temos vários passos que são lidos e interpretados de cima para baixo. Para que venhamos a obter o(s) resultado(s) deste algoritmo, necessitamos "executá-lo", o que resulta em um processo dinâmico.

No fluxo de controle identificamos em cada passo da execução qual é o próximo comando a ser executado.

A compreensão da lógica de programação de um algoritmo está diretamente ligada a compreensão de seu fluxo de controle. A partir de uma compreensão correta, podemos traçar as diversas execuções possíveis de um algoritmo. Se testarmos todas essas possibilidades, e obtivermos resultados corretos, podemos ter certeza de estar entregando um produto final confiável.

Antes de escrever um algoritmo que deverá atingir uma solução, deve-se conhecer profundamente o problema em questão e também planejar como se deve resolve-lo.

Exemplo de uma solução para um problema:

Problema em questão: TRABALHAR
Ponto de Partida: DORMINDO
Solução em linguagem natural (alto nível)

1. ACORDAR — Ponto de partida do problema
2. SAIR DA CAMA
3. IR AO BANHEIRO
4. LAVAR O ROSTO E ESCOVAR OS DENTES
5. COLOCAR A ROUPA
6. SE CHOVER, ENTÃO USAR GUARDA-CHUVA, SENÃO USAR BONÉ
7. IR AO PONTO DE ONIBUS
8. ANDAR DE ONIBUS ATÉ ELE CHEGAR AO SERVIÇO
9. BATER O CARTÃO DE PONTO
10. COMEÇAR O EXPEDIENTE — Obtenção da solução

1.3 Método para desenvolvimento de algoritmos

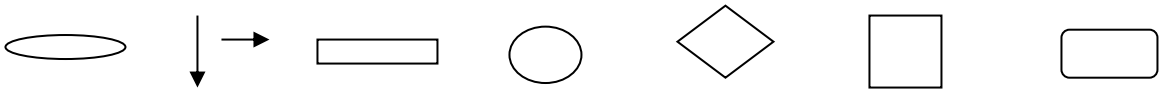
1. Faça uma leitura de todo o problema até o final, a fim de formar a primeira impressão. A seguir, releia o problema e faça anotações sobre os pontos principais.
2. Verifique se o problema foi bem entendido. Questione, se preciso, ao autor da especificação sobre suas dúvidas. Releia o problema quantas vezes for preciso para tentar entendê-lo.
3. Extraia do problema todas as suas entradas (informações, dados).
4. Extraia do problema todas as suas saídas (resultados).
5. Identifique qual é o processamento principal.
6. Verifique se será necessário algum valor intermediário que auxilie a transformação das entradas em saídas. Esta etapa pode parecer obscura no início, mas com certeza no desenrolar do algoritmo, estes valores aparecerão naturalmente.
7. Teste cada passo do algoritmo, com todos os seus caminhos para verificar se o processamento está gerando os resultados esperados.
8. Crie valores de teste para submeter ao algoritmo.
9. Reveja o algoritmo, checando as normas de criação.

1.4 O Pseudocódigo ou Português Estruturado ou Portugol

É uma linguagem natural e informal que ajuda os programadores a desenvolver o algoritmo a fim de se comunicar com a máquina. Consiste principalmente na utilização de uma seqüência estruturada de ações.

1.5 O Fluxograma ou Diagrama de Bloco

É a utilização de símbolos pré-definidos para representar o fluxo de dados durantes a execução dos procedimentos.



Inicio e Fim	Fluxo de dados	Sentenças e Comandos	Conectores de fluxos	Estrutura de Seleção	Estruturas de Repetição	Processo Alternativo

2. Estrutura de Controle

Também conhecida como programação estruturada. Divide-se em 3 estruturas:

- 2.1 Seqüencial
- 2.2 Seleção ou Condicional
- 2.3 Repetição

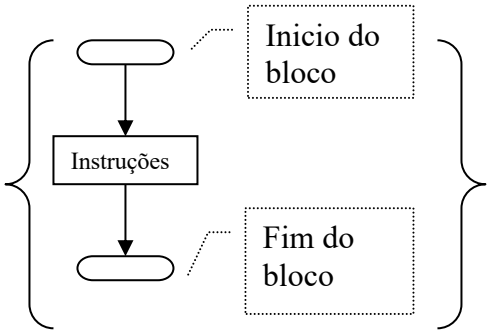
Esta estrutura não são independentes, por isso interagem entre si

2.1 Programação Estruturada Seqüencial

Uma programação seqüencial é aquela cujas ações são executadas uma após a outra (TOP-DOWN) e identificadas pelo início e fim de cada bloco de instruções.

Exemplo de Programação Seqüencial:

```
INICIO
  |
  |  {Corpo do Algoritmo}
  v
FIM
```



FLUXOGRAMA

É sempre importante utilizar técnicas de parametrização e identação na hora de se escrever os algoritmos. Isto facilita na leitura de seus procedimentos e melhor apresenta sua organização.

A Programação Estruturada é definida por uma seqüência de ações que seguem a seguinte composição:

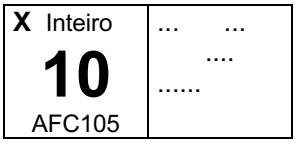
- 2.1.1 Constantes e Variáveis
- 2.1.2 Símbolos
- 2.1.3 Operadores
- 2.1.4 Funções
- 2.1.5 Comandos
- 2.1.6 Sentenças

2.1.1 Constantes e Variáveis

Constantes - Faz referências a dados que não sofrem alterações ao longo do algoritmo.
Variáveis - Vaz referências aos dados do problema e que deverão ser armazenadas na memória do computador e sofrem alterações ao longo do algoritmo. Toda variável deve ser classificada a um tipo exclusivo de informação pela qual a memória da máquina será preparada para armazenar o dado, e este tipo será incluído já no inicio dos procedimentos.

Modelo de uma Célula de Memória

Nome: X
Tipo: Inteiro
Conteúdo: 10
Endereço: AFC105



Modelo de armazenamento de variáveis na memória do computador

Composição de uma variável

- Nome da Variável:
  Nome que será utilizado para identificar a informação e determinar onde a mesma se encontra na memória

- Tipo de dados:
  INTEIRO → Dados numéricos com valores inteiros. Ex. IDADE
  REAL → Dados numéricos e que suportam casas decimais. Ex. ALTURA, PESO
  CARACTER → Dados texto (conjunto de caracteres não numéricos ou para fins numéricos) Ex. NOMES, ENDERECOS, CEP, PALAVRAS.
  DIMENSIONADAS → Conjunto dinâmico de dados armazenados em forma de tabelas ou seqüências limitadas homogêneas (VETORES e MATRIZES) ou heterogêneas (CAMPOS e REGISTROS)
  LÓGICOS → Assumem valores únicos e distintos: V (verdade), F (falso).

- Conteúdo:
  Valor atribuído à variável e que será colocado na memória.

Tabela de Operadores Lógicos		
V e V = V	V ou V = V	não V = F
V e F = F	V ou F = V	não F = V
F e V = F	F ou V = V	
F e F = F	F ou F = F	

- Endereço:
  Endereço físico de armazenamento que será gerenciado pelo sistema operacional do computador e este evitará que outras informações invadam a área restrita a uma variável

Declaração das Variáveis:

A declaração é dada sempre no topo do algoritmo.

Exemplo:

```
INICIO

    INTEIRO A, X, idade;
    REAL peso, L;
    CARACTER nome, cep, K;
    LOGICO resposta;
    VETOR notas[10];

FIM.
```

2.1.2 Símbolos

Representação por símbolos que o computador identifica e interpreta a fim de satisfazer um procedimento.

Os principais e mais utilizados são:

- ← atribuição de dados a variáveis
- (início de um conjunto dados
-) fim de um conjunto de dados
- " inicio e fim de uma expressão caracter
- // comentário
- ; terminadores de linhas de instruções
- . fim do algoritmo
- , separador de conjunto de dados

Exemplo:

```
INICIO
    // variáveis
        INTEIRO A, X, idade;
        REAL peso, L;
        CARACTER nome, cep, K;
        LOGICO resposta;
    // principal
        A ← 5;
        Peso ← 65.4;
        nome ← "FEPI" ;
        resposta ← falso;
FIM.
```

2.1.3 Operadores

Símbolos aritméticos que geram processamento e retornam resultados.
São classificados em:

- Aritméticos (cálculos e processamentos)

Operador	Descrição	Exemplo	Resultado
+	Adição	10 + 15	25
-	Subtração	20 – 10	10
*	Multiplicação	3 * 5	15
/	Divisão (resultado será um número real)	5 / 2	2,5
^, **	Exponenciação	5 ^ 2 ou 5**2	25

- Relacionais (equivalência ou igualdade entre dois ou mais valores)

Operador	Descrição	Exemplo	Resultado
=	Igualdade	10 = 15	Falso
>	Maioridade	10 > 15	Falso
<	Menoridade	10 < 15	Verdade
>=	Maioridade e Igualdade	10 >= 15	Falso
<=	Menoridade e Igualdade	10 <= 15	Verdade
<>	Diferenciação	10 <> 15	Verdade

Exemplos de uso:

```
INICIO
        // variáveis
                INTEIRO A, B, C;
        // principal
                A ← 5; B ← 4;
                C ← A + B;
                C > A;          ( condição verdadeira )
                B = A;          ( condição falsa )
                A <> B;         ( condição verdadeira )
FIM.
```

2.1.4 Funções

Rotinas matemáticas prontas no processador que geram processamento sobre um determinado valor especificado e retornam resultados.

Principais funções:

- SEN(x) → retorna o seno do valor especificado entre os parentes
- COS(x) → retorna o cosseno do valor especificado entre os parentes
- MOD(x) → retorna a o resto da divisão de um valor especificado por parentes
- ABS(x) → retorna o valor absoluto do valor especificado entre os parentes
- SQRT(x) → retorna a raiz quadrada do valor especificado entre os parentes

Exemplos de uso:

```
INICIO
// variáveis
        INTEIRO A, B, C;
// principal
        A ← 25;
        B ← -4;
        C ← SQRT(A);        ( c  = 5 )
        C ← ABS(B);         ( c  = 4 )
FIM.
```

2.1.5 Comandos

Palavras chaves ou instruções pré-definidas que são interpretadas pelo processador e passam produzir iterações entre o usuário e a máquina.

Principais comandos:

- LEIA(x) → permite ao usuário informar um valor para a variável durante a execução do algoritmo
- ESCREVA(x) → mostra na tela do computador o valor da variável em questão
- IMPRIMA(x) → envia para impressora o valor da variável em questão

Exemplo

```
INICIO
// variáveis
        INTEIRO A, B, C;
// principal
        Leia(A);
        Escreva(" Informe um valor para variável B: ");
        Leia(B);
        C ← A / B;
        Escreva(A);
        Escreva(" Resultado da divisão = ", C);
        Imprima(" O resultado da divisão de ",A, " por ", B, " é  ",C);

FIM.
```

2.1.6 Sentenças

É a mistura das todas as estruturas afim de se obter um resultado ou uma solução para um problema. Formam uma sentença o conjunto de regras, comandos, variáveis, funções e símbolos, agrupados de forma lógica permitindo que o computador possa processar e interagir com os dados.

Exemplo

```
INICIO
// variáveis
        INTEIRO A, B, C;
// principal
        Leia(A);
        Leia(B);
        C ← A / ABS(B);
        Escreva( "o valor de C é :" , C);
FIM.
```

Todas linhas de ações por mais simples que sejam, formam uma sentença.

Lista de exercícios...

Entregar em: _____/_____/_____

2.2 Programação Estruturada de Seleção ou Condicional

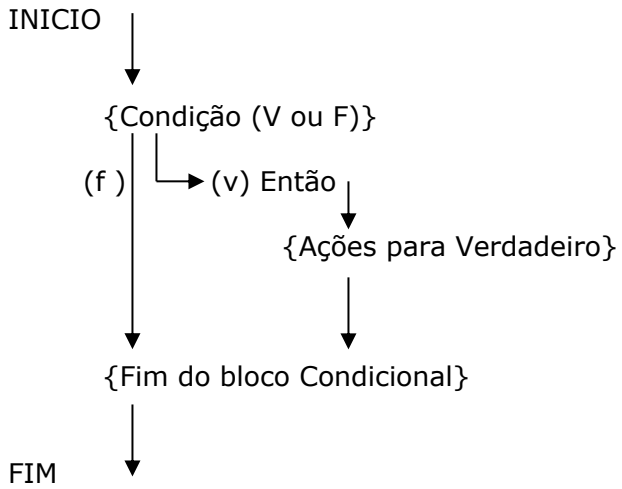
Uma estrutura de Seleção ou Condicional é aquela cujas ações são determinadas por uma verificação de entrada. Uma análise de condição será posta em questão e o computador através de um processamento de dados contidos na memória reconhecerá se a condição é verdadeira ou falsa e assim determinará quais ações serão executadas.

→ Existem 3 tipos de estruturas de seleção:

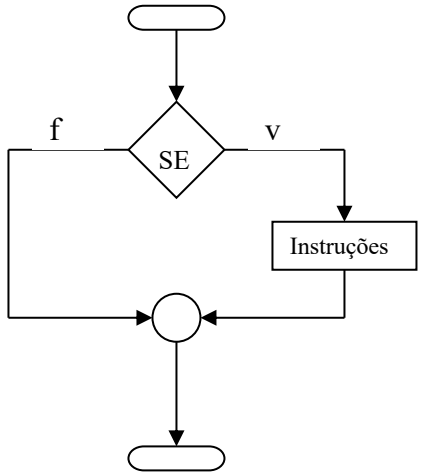
2.2.1 Seleção Simples (SE-ENTÃO)
2.2.2 Seleção Composta (SE-ENTÃO-SENÃO)
2.2.3 Seleção Múltipla (CASO)

2.2.1 Seleção Simples ou Única (SE ENTÃO): O bloco de instruções somente será executado se a verificação de entrada for verdadeira, caso isto não ocorra, o bloco condicional não será executado.

Pseudocódigo:



Fluxograma:



Exemplo Prático: (utilizando o português estruturado)

a.
```
Ir para escola
 Procurar minha agenda
 Se  (encontrar a agenda)
     Então
     Levá-la para escola
 Fim Se
 Pegar o ônibus
 Chegar na escola
```

b.
```
Ir a um baile
 Chegar a portaria do clube
 Mostrar a identidade
 Se  (idade menor que 18 anos)
     Então
     Apresentar acompanhante
 Fim Se
 Entrar no baile
 Curtir a festa
```

Exemplo Prático: (utilizando algoritmo)

a.
```
Inicio
// variáveis
    inteiro idade;
    caracter resposta;
// principal
    Escreva("Informe a idade");
    Leia(idade);
    Se (idade >= 18)
        Então
        Resposta ← "Maioridade";
        Escreva(resposta);
    Fim se;
Fim.
```

b.
```
Inicio
// variáveis
        inteiro idade;
        caracter resposta;
// principal
        Escreva("Informe a idade");
        Leia(idade);
        Se (idade>10) e (idade<30)
            Então
            Resposta ← "Jovem";
        Escreva(resposta);
        Fim se;
Fim.
```

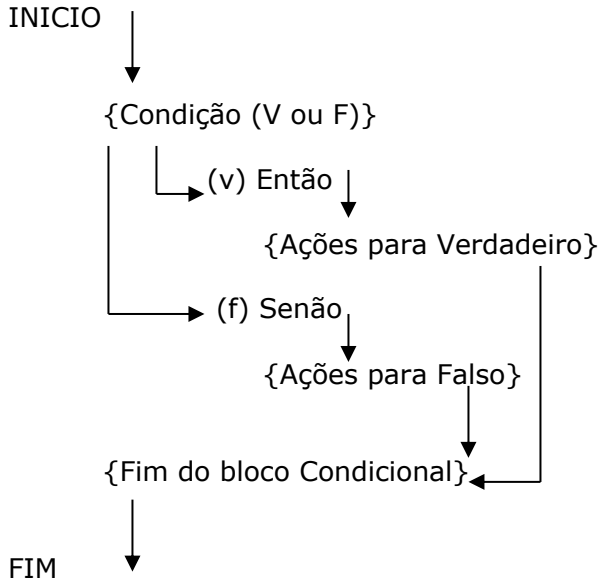
c.
```
Inicio
// variáveis
    inteiro idade;
    caracter resposta;
// principal
    Escreva("Informe a idade");
    Leia(idade);
    Se (idade=70)ou (idade=80)
        Então
        Resposta ← "3ª idade";
        Escreva(resposta);
    Fim se;
Fim.
```

d.
```
Inicio
// variáveis
        inteiro idade;
        caracter resposta;
// principal
        Escreva("Informe a idade");
        Leia(idade);
        Se ((idade>1) e (idade<9))
            ou (idade<1) Então
            Resposta ← "Criança";
            Escreva(resposta);
        Fim se;
    Fim.
```

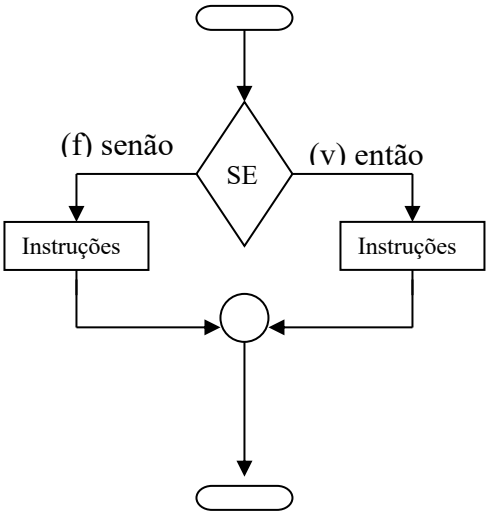
2.2.2 Seleção Composta e/ou Seleção Dupla (SE ENTÃO SENÃO) neste caso passa a existir mais de um bloco de instruções porém apenas um será executado, e isto se dará pelo resultado obtido na verificação de entrada. Quando esta verificação resultar em caminhos distintos para verdadeiro ou para falso, chamamos de Seleção Dupla, mas quando na verificação existir várias possibilidades de respostas então a chamamos de composta.

Modelo de Seleção Dupla

Pseudocódigo:



Fluxograma:



Exemplo Prático: (utilizando o português estruturado)

a.
```
Ir para escola
Se (fazer Frio)
    Então
        Vestir blusa
    Senão
        Vestir Camiseta
Fim se
Chegar a escola
```

b.
```
Sair a noite
Chegar a portaria do clube
Mostrar a identidade
Se  (idade >= 18)
   Então
        Entrar no baile
        Curtir a festa
   Senão
        Não entrar no baile
Fim Se
Voltar para casa
```

c.
```
Fazer a prova
Obter a nota
Se (nota < 30) Então
    Reprovado
Senão
    Se (nota >= 30) e (nota < 60) Então
        Recuperação
    Senão
        Aprovado
    Fim se
Fim se
```

Exemplo Prático: (utilizando algoritmo)

a.
```
Início
    // variáveis
        inteiro hora;
        caracter Ação1, Ação2, Ação3, Ação4, Ação5;
    // principal
        leia(hora);
        Se hora > 6 e hora < 18 então
                Ação1 = Levantar, tomar  café, trabalhar, almoçar, trabalhar e  ir embora;
                Se hora  > 6 e hora < 12 então
                        Ação2 = Levantar, tomar  café, trabalhar e almoçar;
                        Se hora > 6 e hora < 9 então
                                Ação3 = Levantar, tomar  café e trabalhar;
                                Se hora > 6 e hora < 8 então
                                        Ação4 = Levantar e  tomar  café ;
                                        Se hora > 6 e hora < 7 então
                                                Ação5 = Levantar;
                                        Fim se;
                                Fim se;
                        Fim se;
                Fim se;
        Fim se;
    Fim.
```

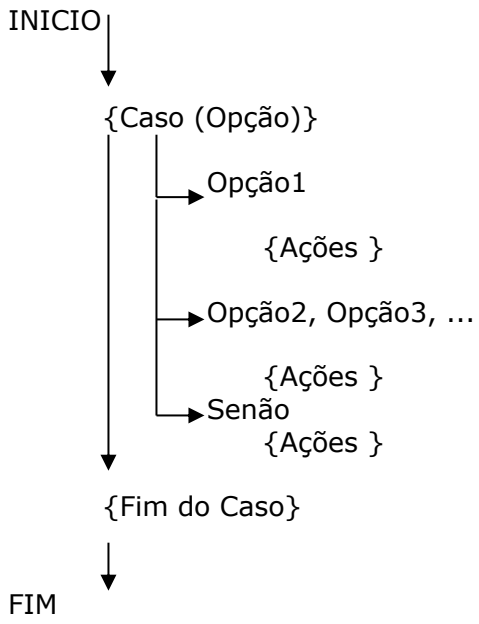
b.
```
Início
    // variáveis
        inteiro hora;
        caracter período;
    // principal
        leia(hora);
        Se hora > 6 e hora < 12 então
                período = manha;
        Senão
                Se hora  > 12 e hora < 18 então
                        período = tarde;
                Senão
                        Se hora > 18 e hora < 24 então
                                período = noite;
                        Senão
                                Se hora > 0 e hora < 6 então
                                        período = madrugada;
                                Fim se;
                        Fim se;
                Fim se;
        Fim se;
    Fim.
```

c.
```
Início
        inteiro hora;
        caracter período;
        leia(hora);
        Se hora > 6 e hora < 18 então          período = dia;
        Fim se;
        Se hora  > 6 e hora < 12 então         período = manha;
        Fim se;
        Se hora > 18 e hora < 24 então         período = noite;
        Fim se;
        Se hora > 12 e hora < 18 então         período = tarde;
        Fim se;
    Fim.
```

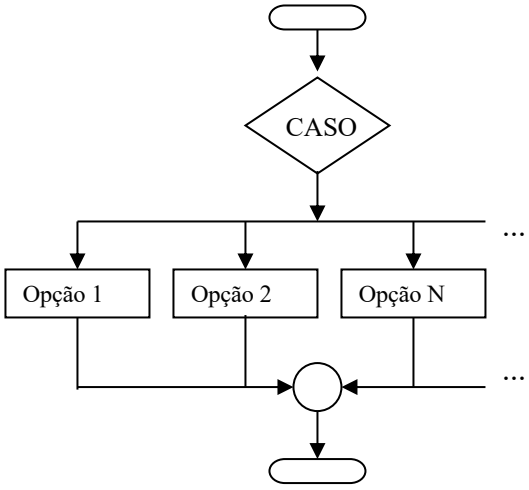
2.2.3 Seleção Múltipla (CASO) Utilizada quando temos muitas possibilidades para uma determinada situação, onde a aplicação da estrutura se...então...senão, tornaria o algoritmo muito complexo.

Modelo de Seleção Multipla

Pseudocódigo:



Fluxograma:




```
Caso <expressão>
    valor1 :  <comando 1>;
    valor2 : valor5 : <comando 2>;
    ...
    senão <comando n>;
fim-caso;
```

As opções podem apresentar valores individuais ou uma faixa de valores.

Exemplo Prático: (utilizando o português estruturado)

a.
```
Cumprimentar alguém
Olhe as Horas
Caso(Horas)
   >=6   e <11: Bom Dia
   >=12 e <18: Boa Tarde
   >=18 e <24: Boa Noite
Fim do caso
```

b.
```
Caixa Eletrônico
Informe a opção
Caso(opção)
      saque:  Agência, nº.conta, senha,
              valor. Retirar dinheiro
      extrato:Informar Agência,nº.conta,
              senha. Retirar extrato
      deposito:Informar Agência,nº.conta,
              valor. Retirar comprovante
Fim do caso
```


Exemplo Prático: (utilizando o algorítmo)

a.
```
Inicio
  // declaração de variáveis
  Inteiro Numero;
  Caracter Extenso;
  // Principal
  leia(Numero);
  caso(Numero)
        1: Extenso ← 'Um';
        2: Extenso ← 'Dois';
        3: Extenso ← 'Três';
        4: Extenso ← 'Quatro';
        5: Extenso ← 'Cinco';
        6: Extenso ← 'Seis';
        7: Extenso ← 'Sete';
        8: Extenso ← 'Oito';
        9: Extenso ← 'Nove';
        senão:  Extenso ← 'Erro';
  fim-caso;
Fim.
```

b.
```
Inicio
  // declaração de variáveis
  Inteiro A,B, Opcao;
  Real C;
  // Principal
     A ← 3;
     B ← 2;
     escreva(" Escolha sua opção:");
     escreva(" 1 – Somar.");
     escreva(" 2 – Subtrair.");
     escreva(" 3 – Multiplicar.");
     escreva(" 4 – Dividir.");
     leia(Opcao);
     caso(Opcao)
          1: C ← A+B;
          2: C ← A-B;
          3: C ← A*B;
          4: C ← A/B;
          senão: escreva("Inválido");
     fim-caso;
     escreva("O resultado será: ",C);
Fim.
```

Lista de exercícios...

Entregar em: _____/_____/_____

2.3 Programação Estruturada de Repetição

Uma estrutura de Repetição é aquela cujas ações são executadas repetidamente, enquanto uma determinada condição permanece válida. O fator que diferencia os vários modelos de estrutura de repetição é o ponto em que será colocada a condição.

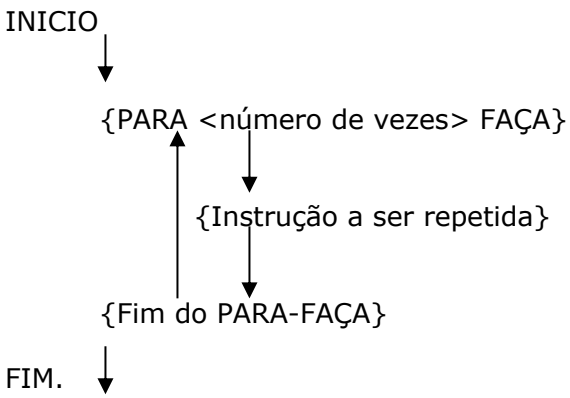
→ Existem 3 tipos de estruturas de repetição:

2.3.1 Para-Faça
2.3.2 Enquanto-Faça
2.3.3 Repita-Até

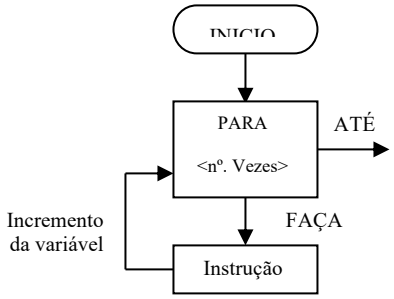
2.3.1 Para-Faça Usamos a estrutura Para-Faça quando precisamos repetir um conjunto de comandos em um número pré-definido de vezes. É conhecida como estrutura de repetição por laços (loops) definidos, pois se utiliza uma variável de controle que é incrementada em um número determinado de unidades de um valor inicial até um valor final.

Modelo de Repetição Para-Faça

Pseudocódigo:



Fluxograma:




```
PARA  Inicio  ATÉ  Fim  FAÇA
        <Sentença 1>;
        <Sentença 2>;
        <Sentença n>;
Fim-Para;
```

Execução enquanto Início for menor ou igual a Fim

Quando o algoritmo encontra a instrução fim-para, incrementa a variável INICIO em 1 unidade (default) ou mais. Cada vez que é completada o bloco de instrução, ele retorna a linha da estrutura PARA e testa se INICIO é menor ou igual a FIM, se for menor ou igual o processo continua no laço (loop), caso não, o processo é abandonado.

Obs: O valor da variável INICIO não pode ser alterado no interior da estrutura.

Exemplo Prático: (utilizando o português estruturado)

Ir de elevador do primeiro ao quinto Andar
Chamar o elevador
Entrar no elevador
Informar o andar
Para andar do primeiro até o quinto faça
    mover ao próximo andar
Fim do Movimento
Sair do elevador

Exemplo Prático: (utilizando o algoritmo)

a.
```
Inteiro var, resultado;
para var ← 1 até 10 faça
        resultado ← 2 * var;
fim-para;
escreva(resultado);
```

b.
```
Inteiro var, resultado;
para var ← 1 até 10 faça
        resultado ← 2 * var;
        escreva(resultado);
fim-para;
```

c.
```
Inteiro var, resultado;
para var←1 até 5 passo 2 faça
        resultado ← 2 * var;
        escreva(resultado);
fim-para;
```

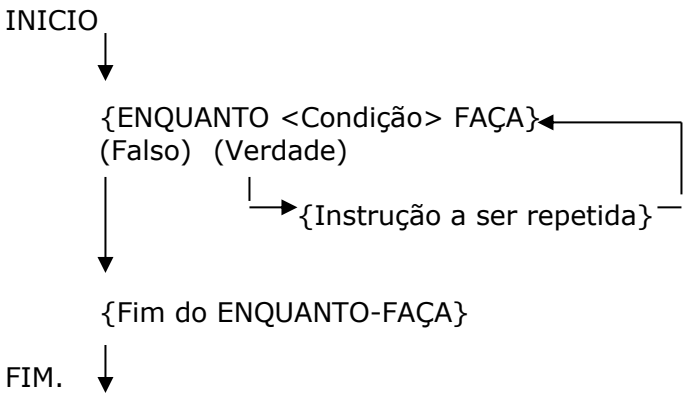
:: Exercícios propostos ::

```
Dadas as informações a seguir escreva a série
Variavel  x
Valor inicial  1
Valor final  30
Regra de ciclo x<-x+1
Formula padrao x+(x*2)/(x^2)
```

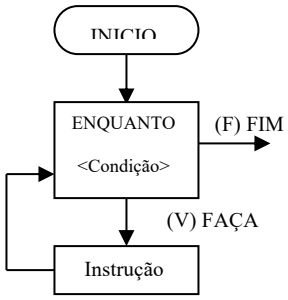
2.3.2 Enquanto-Faça Utilizada quando não sabemos o número de repetições e quando possuímos uma expressão que deve ser avaliada para que os comandos da estrutura sejam executados. Assim, enquanto o valor da <condição> for verdadeiro, as ações dos comandos são executadas. Quando for falso, a estrutura é abandonada, passando a execução para a próxima linha após o comando FIM-ENQUANTO. Se já da primeira vez o resultado for falso, os comandos não serão executados.

Modelo de Repetição Enquanto-Faça

Pseudocódigo:



Fluxograma:




```
Enquanto <condição> Faça
    <comando 1>;
    <comando 2>;
    <comando n>;
Fim-Enquanto;
```

execução enquanto a condição for verdadeira

É sempre importante observar que primeiro se analisa a condição para depois dependendo do resultado obtido executar o bloco a ser repetido. Caso a condição não seja satisfeita nada será feito e a próxima linha após o fim-enquanto será requisitada. Também é necessário caso a condição seja verdadeira permitir o incremento para a variável em condição (se necessário) para que a estrutura não entre em loop infinito.

Exemplo Prático: (utilizando o português estruturado)

```
Ir de elevador do primeiro ao quinto Andar
Chamar o elevador
Entrar no elevador
Informar o Andar
Enquanto Andar atual  for menor que 5 faça
    mover o elevador para cima
    Andar passa para o próximo
Fim do Movimento
Sair do elevador
```


Exemplo Prático: (utilizando o algoritmo)

```
a.
aux ← 1;
enquanto (aux <= 10) faça
        resultado ← 5 * aux;
        aux ← aux + 1;
fim-para
```

:: Exercícios propostos ::

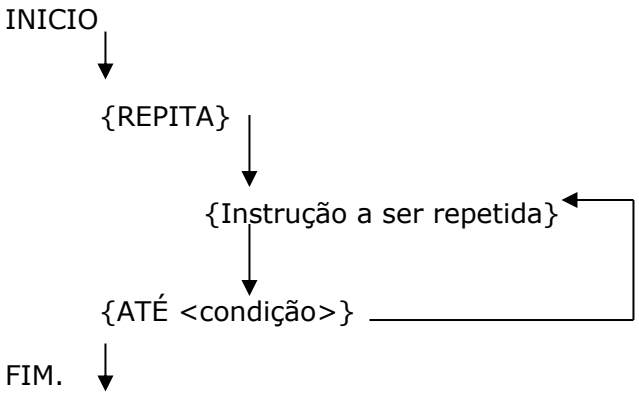
Analisando Séries com estruturas de repetição

S = 3 / 6
Variáveis envolvidas:
Valores de Inicio:
Fórmula Padrão:
Regra de Ciclo:

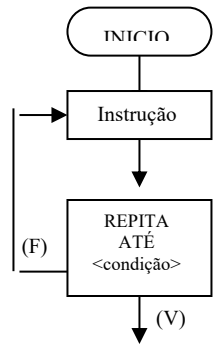
2.3.3 Repita-Até Utilizada quando não sabemos o número de repetições e quando os comandos devem ser executados pelo menos uma vez, antes da expressão ser avaliada. Assim, o programa entra na estrutura Repita...Até que executa seus comandos pelo menos uma vez. Ao chegar no fim da estrutura, a expressão será avaliada. Se o resultado da expressão for verdadeiro, então o comando é abandonado.

Modelo de Repetição Repita-Até

Pseudocódigo:



Fluxograma:



Repita

<comando 1>;
<comando 2>;
<comando n>;

Até <condição>;

É executado pelo menos uma vez

Exemplo Prático: (utilizando o português estruturado)

a.
Ir de elevador do primeiro ao quinto Andar
Chamar o elevador
Entrar no elevador
Informar o Andar
Repita
    mover o elevador para cima
    Andar passa para o próximo
Até Andar atual igual a 5
Sair do elevador

b.
Executar um aplicativo com senha
Abrir o Windows
Abrir a pasta do aplicativo
Executar o arquivo principal
Repita
    Digitar a senha
Até senha ser válida
Usar o aplicativo

Exemplo Prático: (utilizando o algoritmo)

```
a.
   aux ← 1;
   repita
         resultado ← 5 * aux;
         escrever resultado;
         aux ← aux + 1;
   até (aux > 10);
```

:: Exercícios propostos ::

```
inicio
      inteiro x,  y;
      x <- 0;
      y <- 0;
      enquanto x > 0 faça
            y <- y + 3;
            x <- x + 1;
      fim-enquanto;
fim.
```

```
inicio
      inteiro x,  y;
      x <- 0;
      y <- 0;
      repita
            y <- y + 3;
            x <- x + 1;
      até x > 0;
fim.
```

3. Aplicação de variáveis dimensionais homogêneas

3.1 Vetores – Variáveis Unidimensionais

Os vetores são estruturas de dados que permitem o armazenamento de um conjunto de dados de mesmo tipo. Por este motivo, são chamadas de estruturas homogêneas. Os vetores são unidimensionais, pois cada elemento do vetor é identificado por um índice.

Similarmente, podemos definir vetores como posições de memória, identificadas por um mesmo nome, individualizadas por índices e cujo conteúdo é de mesmo tipo.
Para acessarmos um elemento de um vetor, referimo-nos ao nome do vetor acompanhado pelo seu índice que virá entre colchetes ([e]). Pense num prédio com 120 apartamentos.

Para enviar uma correspondência a um determinado apartamento, devemos colocar no endereço de destinatário, o número do prédio mais o número do apartamento. O vetor funciona de forma similar.

Veja a sintaxe da declaração de um vetor:

Tipo Básico de Dados: Nome do vetor[nº de elementos]

Para fazermos referência a um elemento do vetor, colocamos:

Nome do vetor[elemento]

Cada elemento de um vetor é tratado como se fosse uma variável simples.

Exemplo:

Supondo que pedíssemos para criar um algoritmo para ler o nome de 5 pessoas, e mostrasse esses nomes na ordem inversa de leitura. A princípio, vocês pensariam em cinco variáveis: nome1, nome2, nome3, nome4 e nome5.

Veja como ficaria a solução, nesse caso:

```
Inicio
   caracter: nome1, nome2, nome3, nome4, nome5;
    escreva('Informe o nome de 5 pessoas: ');
   leia(nome1);    //ANA
    leia(nome2);    //PAULA
    leia(nome3);     //CRISTINA
    leia(nome4);    //GUSTAVO
    leia(nome5);    //ANTONIO
   escreva('Ordem Inversa de Leitura ');
   escreva(nome5);
   escreva(nome4);
   escreva(nome3);
   escreva(nome2);
   escreva(nome1);
Fim
```

Assim, na memória teríamos ...

Nome1	Nome2	Nome3	Nome4	Nome5
ANA	PAULA	CRISTINA	GUSTAVO	ANTONIO

Todavia, se alterássemos esse algoritmo para ler o nome de 100 pessoas, a solução anterior se tornaria inviável. Para casos como este, podemos fazer uso de vetores. Se tivéssemos criado 100 variáveis, teríamos que declarar e usar: nome1, nome2, nome3, ..., nome99, nome100. Com o vetor passamos a ter: nome[1], nome[2], nome[3], nome[99], nome[100], onde a declaração do vetor se limita à linha: caracter: nome[100].

Veja que para todos os elementos nos referimos ao mesmo nome de vetor "Nome".

Assim, veja a solução do algoritmo anterior com o uso de vetores:

```
Início
    Caracter: nome[5];
    Inteiro: aux;
    para aux ← 1 até 5 faça
       escreva('Informe o Nome ', aux);
       leia(nome[aux]);
    fim-para;

   escreva('Ordem Inversa de Leitura ');

    para aux ← 5 até 1 faça
       escreva (nome[aux]);
    fim-para
fim
```

Veja a representação da memória:

NOME				
1	2	3	4	5
ANA	PAULA	CRISTINA	GUSTAVO	ANTONIO

3.2 Matrizes – Variáveis Multidimensionais

As matrizes são estruturas de dados que permitem o armazenamento de um conjunto de dados de mesmo tipo, mas em dimensões diferentes. Os vetores são unidimensionais, enquanto as matrizes podem ser bidimensionais (duas dimensões) ou multidimensionais.

Similarmente podemos conceituar matrizes como um conjunto de dados referenciado por um mesmo nome e que necessitam de mais de um índice para ter seus elementos individualizados.

Para fazer referência a um elemento da matriz serão necessários tantos índices quantas forem as dimensões da matriz.

Veja a sintaxe da declaração de uma matriz:

Tipo básico de Dados : Nome da matriz [Li, Ls, Nq]

onde:

Li – Limite inferior *
Ls – Limite superior *
Nq – Número de quadrantes (somente para matrizes multidimensionadas)
* Valores obrigatórios

Para fazermos referência a um elemento da matriz, colocamos:

Nome da matriz [linha, coluna]

O número de dimensões de uma matriz pode ser obtido pelo número de vírgulas (,) da declaração mais 1. O número de elementos pode ser obtido através do produto do número de elementos de cada dimensão.

Obs: Quando você desejar percorrer uma matriz, linha por linha, crie uma estrutura de repetição, fixando a linha e variando a coluna. Para percorrer uma matriz, coluna por coluna, fixe a coluna e varie a linha.

Vamos pensar numa estrutura onde as colunas representem os cinco dias úteis da semana, e as linhas representem as três vendedoras de uma loja. Na interseção de cada linha x coluna, colocaremos o faturamento diário de cada vendedora.

	(Segunda) **COLUNA 1**	(Terça) **COLUNA 2**	(Quarta) **COLUNA 3**	(Quinta) **COLUNA 4**	(Sexta) **COLUNA 5**
(SANDRA) **LINHA 1**	1050,00	950,00	1241,00	2145,00	1256,00
(VERA) **LINHA 2**	785,00	1540,00	1400,00	546,00	0,00
(MARIA) **LINHA 3**	1658,00	1245,00	1410,00	245,00	1546,00

A representação desta tabela em forma de matriz, seria:

VendasDiarias : matriz [3, 5] de real;

Veja como ficaria o algoritmo para ler esses valores:

```
Algoritmo LeVendasDiarias;
Inicio
      real: VendasDiarias[3,5];
      inteiro: ndLinha, indColuna, i ;


   //Variando o número de linhas - Vendedoras
   Para indLinha ← 1 até 3 faça
         escrever ('Vendedora :', indLinha);

      //Variando o número de colunas – Dias da Semana
      Para indColuna ← 1 até 5 faça
            escreva ('Faturamento do Dia : ', indColuna);
            leia (VendasDiarias[indLinha, indColuna]);
         Fim-para;
   Fim-para;

Fim
```

Poderíamos melhorar o algoritmo acima, trabalhando com um vetor que contivesse os nomes dos dias da semana e das vendedoras. Assim, a comunicação do programa com o usuário ficaria mais clara. Veja:

```
Algoritmo LeVendasDiariasVersao2;

Inicio
      real: VendasDiarias[3,5];
      caracter: Vendedoras[3];
      caracter: DiasSemana[5];
      inteiro: indLinha, indColuna;

      Vendedoras[1] ← 'Sandra';
      Vendedoras[2] ← 'Vera';
      Vendedoras[3] ← 'Maria';
      DiasSemana['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta');

      //Variando o número de linhas - Vendedoras
      Para indLinha ← 1 até 3 faça
            escreva('Vendedora : ', Vendedoras[indLinha]);
            //Variando o número de colunas – Dias da Semana
            Para indColuna ← 1 até 5 faça
                  escreva('Fatur.do Dia:', DiasSemana[indColuna]);
                  leia(VendasDiarias[indLinha, indColuna]);
            Fim-para;
      Fim-para;
Fim
```

Um algoritmo que apenas lê e nada faz com esses resultados, não serve para grande coisa, certo ?! Por isso, vamos melhorar esse algoritmo e apresentar como resultado o faturamento diário de todas as vendedoras.

```
Algoritmo LeVendasDiariasVersao3;
Início
      real: VendasDiarias[3,5];
      caracter: Vendedoras [3], DiasSemana[5];
      inteiro: indLinha, indColuna;
      real: FaturaDia;

      Vendedoras[1] ← 'Sandra';
      Vendedoras[2] ← 'Vera';
      Vendedoras[3] ← 'Maria';

      DiasSemana[1] ← 'Segunda';
      DiasSemana[2] ← 'Terça';
      DiasSemana[3] ← 'Quarta';
      DiasSemana[4] ← 'Quinta';
      DiasSemana[5] ← 'Sexta';

      //Variando o número de linhas – Vendedoras }
      para indLinha ← 1 até 3 faça
            escreva('Vendedora : ', Vendedoras[indLinha]);
            //Variando o número de colunas – Dias da Semana
            para indColuna ← 1 até 5 faça
                  escreva('Fatur.do Dia : ', DiasSemana[indColuna]);
                  leia(VendasDiarias[indLinha, indColuna]);
            fim-para;
      fim-para;

      //Vamos começar variando a coluna, para poder obter o //faturamento
      de cada dia da semana
      Para indColuna ← 1 até 5 faça
            //A cada novo dia a variável que recebe faturamento é
            // zerada
            FaturaDia ← 0;

            //Vamos variar a linha, para obter os valores
            //faturados de cada vendedora
            para indLinha ← 1 até 3 faça
                  FaturaDia←FaturaDia + _
                              VendasDiarias[indLinha,indColuna];
            fim-para

            escreva("Faturamento de : ", DiasSemana[indColuna]);
            escreva(FaturaDia);

      fim-para;

fim
```

Tabela ASCII

Caracteres normais

Binário	Decimal	Hex	Gráfico
0010 0000	32	20	(vazio) (sp)
0010 0001	33	21	!
0010 0010	34	22	"
0010 0011	35	23	#
0010 0100	36	24	$
0010 0101	37	25	%
0010 0110	38	26	&
0010 0111	39	27	'
0010 1000	40	28	(
0010 1001	41	29	)
0010 1010	42	2A	*
0010 1011	43	2B	+
0010 1100	44	2C	,
0010 1101	45	2D	-
0010 1110	46	2E	.
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1
0011 0010	50	32	2
0011 0011	51	33	3
0011 0100	52	34	4
0011 0101	53	35	5
0011 0110	54	36	6
0011 0111	55	37	7
0011 1000	56	38	8
0011 1001	57	39	9
0011 1010	58	3A	:
0011 1011	59	3B	;
0011 1100	60	3C	<
0011 1101	61	3D	=
0011 1110	62	3E	>
0011 1111	63	3F	?

Binário	Decimal	Hex	Gráfico
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79	4F	O
0101 0000	80	50	P
0101 0001	81	51	Q
0101 0010	82	52	R
0101 0011	83	53	S
0101 0100	84	54	T
0101 0101	85	55	U
0101 0110	86	56	V
0101 0111	87	57	W
0101 1000	88	58	X
0101 1001	89	59	Y
0101 1010	90	5A	Z
0101 1011	91	5B	[
0101 1100	92	5C	\
0101 1101	93	5D	]
0101 1110	94	5E	^
0101 1111	95	5F	_

Binário	Decimal	Hex	Gráfico
0110 0000	96	60	`
0110 0001	97	61	a
0110 0010	98	62	b
0110 0011	99	63	c
0110 0100	100	64	d
0110 0101	101	65	e
0110 0110	102	66	f
0110 0111	103	67	g
0110 1000	104	68	h
0110 1001	105	69	i
0110 1010	106	6A	j
0110 1011	107	6B	k
0110 1100	108	6C	l
0110 1101	109	6D	m
0110 1110	110	6E	n
0110 1111	111	6F	o
0111 0000	112	70	p
0111 0001	113	71	q
0111 0010	114	72	r
0111 0011	115	73	s
0111 0100	116	74	t
0111 0101	117	75	u
0111 0110	118	76	v
0111 0111	119	77	w
0111 1000	120	78	x
0111 1001	121	79	y
0111 1010	122	7A	z
0111 1011	123	7B	{
0111 1100	124	7C	|
0111 1101	125	7D	}
0111 1110	126	7E	~
0111 1111	127	7F	Delete

Referências Bibliográficas

SEBESTA, Robert W. Conceitos de linguagem de programação, 4ª edição. Editora Bookman, ano 2000.

FARRER, Harry, BECKER, Christiano G., FARIA, Eduardo C., MATOS, Helton Fábio de, SANTOS, Marcos Augusto dos, MAIA, Miriam Lourenço. Algoritmos Estruturados. Editora Guanabara, 1989.

GUIMARÃES, Angelo de Moura, LAGES, Newton A de Castilho. Algoritmos e estruturas de dados. Rio de Janeiro. LTC – Livros Técnicos e Científicos Editora, 1985.

SALVETTI, Dirceu Douglas, BARBOSA, Lisbete Madsen. Algoritmos. Editora Makron Books, 1998.

SILVA, Joselias Santos da. Concursos Públicos – Raciocínio Lógico. R&A Editora Cursos e Materiais Didáticos, 1999.

WIRTH, Niklaus. Algoritmos e Estruturas de Dados. Editora Prentice-Hall do Brasil, 1986.

SILVA, Cristiano Vieira. Apostila de Algoritmos – Introdução a Lógica de Programação, Colégio Universitas – Cursos Técnico em Informática. Edição particular, publicação ©2004 – A utilização deste material será somente sob autorização e a fonte deverá ser citada.