

Lógica de Programação

Teoria e Prática com Linguagem C

INTELLECTUALE TECNOLOGIA E TREINAMENTO

Prof. Eduardo Casavella

O que é Lógica de Programação?

- Lógica de Programação é a técnica de criar sequências lógicas de ações para realizar uma determinada tarefa que será executada por um programa de computador.

Algoritmo

- Algoritmo é uma sequência finita de passos que visa atingir um determinado objetivo.
- Tipos de algoritmos:
 - Descrição narrativa
 - Diagrama de blocos
 - Pseudocódigo

Instruções

- Instruções são um conjunto de regras ou normas definidas para a realização ou emprego de algo.
- No jargão da informática, uma instrução é o que indica a um computador uma ação elementar a executar.

Sequência Lógica

- “Sequência Lógica são passos executados até atingir um objetivo ou solução de um problema.”
- É necessário escrever um conjunto de instruções na sequência lógica, ou seja na ordem adequada para solucionar o problema.

Regras para a construção de algoritmos

- Usar somente um verbo por frase;
- Ser objetivo;
- Não utilizar palavras que possam ter duplo sentido.
- Sempre que possível usar frases curtas.

Descrição Narrativa:

- Consiste em analisar o problema ou tarefa e escrever usando a língua portuguesa os passos a serem seguidos para resolver o problema, ou realizar a tarefa.
 - Vantagem: A língua é bem conhecida.
 - Desvantagem: A língua portuguesa permite várias interpretações, o que pode dificultar a transcrição do algoritmo para o programa.

Exemplo de Descrição narrativa: Ir para a escola

- Acordar às 6 horas.
- Tomar banho.
- Secar-se.
- Escolher uma roupa no armário.
- Vestir-se.
- Tomar café.
- Sair de casa.
- Pegar onibus.
- Descer próximo à escola.
- Entrar na escola.

Exercício – Descrição Narrativa

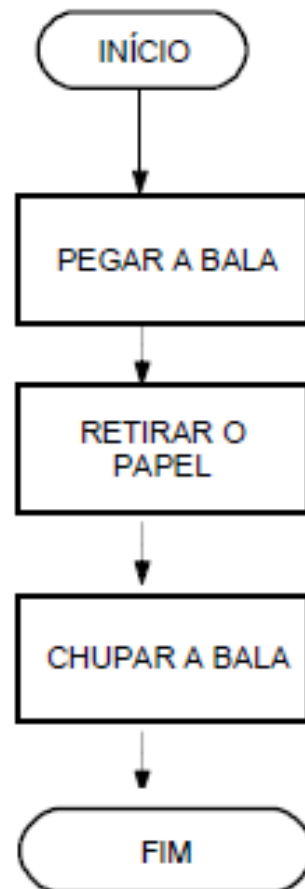
- Elaborar a descrição narrativa para trocar uma lâmpada

Diagrama de blocos

- Consiste em analisar o problema e utilizar elementos gráficos previamente definidos para representar os passos a serem executados para alcançar a solução.
 - **Vantagens:** Os elementos gráficos são de fácil entendimento.
 - **Desvantagem:** é necessário aprender a simbologia utilizada, e o diagrama não representa muitos detalhes, o que dificulta a transcrição para o programa.

Exemplo de diagrama de blocos

- Diagrama de blocos para executar a tarefa de “chupar uma bala.”



Pseudocódigo

- Consiste em escrever por meio de regras previamente definidas, os passos a serem seguidos para resolver um problema.
 - **Vantagem:** O algoritmo em pseudocódigo é bem próximo da linguagem de programação, o que facilita a transcrição para o programa.
 - **Desvantagem:** É necessário aprender as regras do pseudocódigo.

Do algoritmo ao programa

- Seqüência Lógica (Diagrama de Blocos)
- Algoritmo (Pseudocódigo)
- Linguagem de Programação
- Programa Executável.

Solucionando Problemas

- Devemos ter em mente que um algoritmo não é a única solução de um problema.
- Diversos algoritmos podem levar a solução de um mesmo problema.
- Ao identificar a melhor maneira de resolver o problema estaremos otimizando o algoritmo.

Formas de representação de algoritmos

- Existem diversas formas para representar um algoritmo, por exemplo:
 - Usando uma língua (português, inglês);
 - Usando uma linguagem de programação;
 - Utilizando formato gráfico (Diagrama de Blocos);
 - Pseudocódigo (Português Estruturado ou Portugol).

Pseudocódigo

É uma representação de algoritmo feita utilizando comandos em língua portuguesa.

Programa de computador

- É um algoritmo que será executado por um computador.
- A particularidade do programa de computador é que as instruções executadas são específicas para o computador.

Linguagem de Programação

- A linguagem de programação é o conjunto de instruções que o processador do computador entende.
- As instruções de uma linguagem de programação também são chamadas de comandos.
- São exemplos de linguagem de programação: Cobol, C, C++, C#, Pascal, Object Pascal (Delphi), Java, Visual Basic, etc.

Como construir um algoritmo?

- Primeiramente devemos identificar claramente qual é o problema a ser resolvido.
- Para resolver o problema e construir o algoritmo precisamos responder três perguntas básicas:
 - O que eu preciso?
 - Como chegar ao resultado?
 - Qual é o resultado desejado?

Entrada, processamento e saída



Entrada: São os dados necessários para iniciar a resolução do problema.



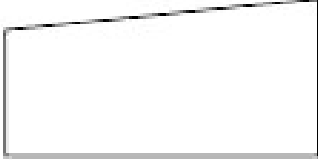

Processamento:
São os procedimentos para chegar ao objetivo.

Saída: São os dados processados, ou seja, o resultado ou os resultados.

O processo de programação

- Vejamos como ocorre o processo de programação:
 - Criamos a lógica do programa em pseudocódigo ou em diagrama de blocos.
 - Transcrevemos a lógica do programa em uma linguagem de programação.
 - Após a digitação do código em linguagem de programação, executamos a compilação do programa.
 - Se a codificação estiver correta o programa será compilado com sucesso e convertido em linguagem de máquina (0 e 1), ou seja linguagem que o processador entende. Nesse momento é gerado um programa executável (.exe).

Simbologia básica para fluxogramas

Simbolo	Função
 TERMINAL	Indica o INÍCIO ou FIM de um processamento Exemplo: Início do algoritmo
 PROCESSAMENTO	Processamento em geral Exemplo: Calculo de dois números
 ENTRADA DE DADO MANUAL	Indica entrada de dados através do Teclado Exemplo: Digite a nota da prova 1
 EXIBIR	Mostra informações ou resultados Exemplo: Mostre o resultado do calculo

Exercício

- Elaborar o pseudocódigo para calcular a média aritmética de um aluno que realizou duas provas.

Linguagem de Máquina

- Um computador transforma cada instrução da linguagem empregada na construção de um programa em sequências de códigos que serão executadas pelo microprocessador.
- Essas sequências de códigos estão em formato binário, ou seja: sequência de dígitos zero e um. A combinação desses códigos determina qual conjunto de ações (instruções) serão executadas pela máquina.

Exemplo de código binário

- Configuração binária para armazenar o número 7995:

1111100111011

Hexadecimais

- Os primeiros computadores exigiam a codificação em código de máquina, isto tornava a tarefa de programação extremamente trabalhosa.
- A fim de facilitar o processo de programação essa sequência de zeros e uns da codificação binária pode ser convertida para códigos hexadecimais. Isto facilita um pouco, porém não muito. Vejamos um exemplo:
 - Para convertermos 7995 para hexadecimal teríamos o valor: 1F3B.

Linguagem Assembly

- Foi a primeira linguagem de programação. Para cada instrução em Assembly tem uma correspondência em código de máquina. É uma linguagem de difícil utilização, porém extremamente rápida.
- Exemplo de código em Assembly:
MOV DX 10C0
INT 21
MOV BX, [002C]
CALL 01AC

Tipos de Linguagens

- Podemos classificar as linguagens de programação em dois tipos:
 1. Linguagens de alto nível.
 - São mais próximas da linguagem humana. Possuem comandos de fácil assimilação. Ex: Pascal, C, C++, C#, Visual Basic, Java.
 2. Linguagens de baixo nível.
 - Aproximam-se do código de máquina. Ex: Assembly.

Variáveis

- Variáveis são locais na memória reservados para armazenar informações.
- Uma variável é composta por dois elementos:
 1. Conteúdo: valor da variável.
 2. Identificador: nome dado à variável para possibilitar sua manipulação.

Regras para criação de identificadores

Os identificadores, como o próprio nome diz, servem para identificar variáveis.

O primeiro caracter de um identificador deve ser obrigatoriamente uma letra, os demais caracteres podem ser letras ou números.

Não é permitido o uso de símbolos ou de sinais de pontuação, exceto o caracter _.

Tipos de variáveis

Numéricas: basicamente dividem-se em:

- Inteiro: armazena números inteiros
- Reais: armazena números com casas decimais (ponto flutuante);

Caracter: São variáveis alfanuméricas ou seja, conjuntos de caracteres (letras, dígitos ou símbolos) que devem ser colocados entre aspas.

Declaração de variáveis

Ao declarar uma variável, reservamos um espaço na memória que armazenará o valor da variável de acordo com o tipo da mesma. Esse espaço é referenciado por um identificador.

Exemplo:

Var

Total:Real;

Comando de atribuição

Atribuição significa armazenar um valor em uma variável.

Exemplo:

Atribuindo o valor 10 para a variável X.

$X \leftarrow 10;$

Em **linguagem C** usamos o operador = para atribuir.
 $X=10;$

Declaração de variáveis em linguagem C

Exemplos

Em C devemos listar primeiro o tipo, depois o nome da variável.

Sintaxe:

<tipo> <nome_da_variável>;

Declarando uma variável inteira:

```
int Quant;
```

Declarando uma variável Real:

```
float Media;
```

Declarando uma variável do tipo caracter:

```
char letra;
```

Constantes

São valores que são armazenados na memória do computador e permanecem os mesmos do início ao fim do programa.

Entrada de dados

Os dados entram no computador por meio dos periféricos de entrada (como teclado, leitor de código de barras, telas sensíveis ao toque, etc).

Na maioria dos nossos exemplos e exercícios, vamos supor que o usuário vai entrar com os dados utilizando um teclado.

Exemplo entrada de dados

Pseudocódigo:

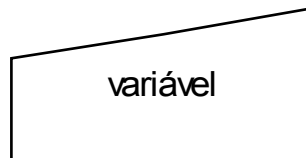
Para armazenar um valor de entrada de dados feito via teclado usamos o comando:

Leia(variável);

ou

Receba(variável);

Representação no **diagrama de blocos:**



Saída de dados

Os dados “saem” do computador utilizando periféricos de saída de dados (monitor, impressora, etc).

Na maioria de nossos exemplos e exercícios, vamos supor que os dados estejam saindo para o monitor, e certas vezes, para a impressora ou sendo gravados em um meio externo de armazenamento.

Exemplo saída de dados

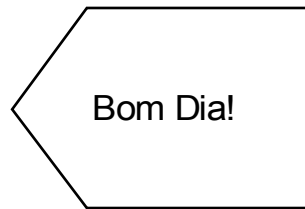
Pseudocódigo:

O comando escreva realiza a saída de dados, mostrando a mensagem no vídeo.

Exemplo:

Escreva("Bom Dia!");

Representação no diagrama de blocos:



Expressões aritméticas

São as expressões que utilizam operadores aritméticos ou funções que envolvem o uso de variáveis numéricas ou constantes.

Operadores aritméticos

SÍMBOLOS	OPERADORES
*	Multiplicação
/	Divisão
%	Módulo
+	Adição
-	Subtração

Prioridade entre operações aritméticas

1. Potenciação, radiciação
2. Multiplicação, divisão
3. Adição, subtração

Expressões relacionais

São expressões compostas por expressões ou variáveis com operadores relacionais.

As expressões relacionais retornam valores lógicos, ou seja: Verdadeiro ou Falso.

Operadores Relacionais

SÍMBOLOS	SIGNIFICADO
>	Maior que
<	Menor que
>=	Maior ou igual
<=	Menor ou igual
==	Igualdade
!=	Diferença
=	Atribuição simples

Operador relacional de igualdade



Não confunda o operador relacional **==** que representa o valor de igualdade de comparação entre dois elementos com o símbolo de **atribuição =**.

Expressões lógicas

Denominamos expressões lógicas às expressões compostas por expressões relacionais com operadores lógicos.

Expressões relacionais retornam valores lógicos.

Operadores Lógicos

OPERADOR	FUNÇÃO
&&	E lógico
	Ou lógico
!	Não lógico

Tabela da verdade - Operadores Lógicos

p	q	p E q	p OU q
verdadeiro	verdadeiro	verdadeiro	verdadeiro
verdadeiro	falso	falso	verdadeiro
falso	verdadeiro	falso	verdadeiro
falso	falso	falso	falso

Introdução à Programação com Linguagem C

Estrutura de um programa em C

```
#include <stdio.h> //biblioteca padrão
```

```
main ( ) //esta é função principal de qualquer programa C
```

```
{ // inicia o corpo da função (BEGIN)
```

```
    comandos ou corpo da função
```

```
} // termina a função (END)
```

Observe que a função `main ()` sempre vai existir em um programa escrito em C.

Usando printf para escrever na tela

```
#include <stdio.h>
```

```
main ( )  
{ // Início
```

```
    // escrevendo uma mensagem na tela  
    printf ("Olá Mundo!");
```

```
} // Final
```

Arquivo stdio.h

A linha **#include <stdio.h>** é uma diretiva que informa ao compilador que ele deve incluir o arquivo (também chamado de biblioteca padrão) stdio.h (Standard Input Output), pois nele existem definições de funções de I/O (entrada e saída padrão).

O arquivo stdio.h possui as funções:

- printf() usada para a saída de dados
- scanf() usada para entrada de dados

Padrão ANSI C

Vejamos um programa escrito de acordo com a padronização ANSI C:

```
#include <stdio.h>
```

```
int main(void) // função main como int e retorno void  
{  
    printf ("Olá Mundo!");
```

```
    return(0); // retornando um valor zero para main  
}
```

Vantagem de seguir padrão ANSI C

Observe que main é definida como uma função inteira, então é ideal incluir a linha com o return 0 para a função antes de fechar chaves e terminar o programa.

Fazendo isso evitamos receber mensagens do compilador dizendo que a função tem que ter algum valor de retorno ao compilar o programa.

Embora não seja errado escrever fora do padrão ANSI C ou seja, o programa vai compilar e funcionar, alguns compiladores informam mensagens de advertência caso você não siga o padrão.

Usando printf para um character e para string

```
#include <stdio.h>
```

```
int main (void)
```

```
{
```

```
    printf ("O character %c ", 'A');
```

```
    printf ("a palavra %s.", "anta");
```

```
    return 0;
```

```
}
```

O código de formatação para representar apenas um character é **%c**.

Para representar uma string, ou seja, uma sequência de caracteres usamos **%s**.

Diferença entre “ e ’

Observe no programa anterior que 'A' é apenas um caracter e está representado entre apóstrofes e a palavra "anta" representa uma cadeia de caracteres e está entre aspas.

Isto é feito para diferenciar um caracter de uma string, ou seja, uma cadeia de caracteres.

A função printf()

Forma geral:

printf (string_de_controle, lista_de_argumentos);

A string de controle é uma descrição de tudo que a função vai colocar na tela.

A string de controle mostra não apenas os caracteres que devem ser colocados na tela, mas também quais as variáveis e suas respectivas posições. Isto é feito usando-se os códigos de controle, que usam a notação %.

Na string de controle indicamos quais, de qual tipo e em que posição estão as variáveis a serem apresentadas.

É muito importante que, para cada código de controle, tenhamos um argumento na lista de argumentos.

Códigos para impressão com printf

Principais códigos para impressão formatada da função `printf ()` e o que eles realmente expressam.

Código	Significado
<code>%d</code>	Inteiro (número inteiro decimal)
<code>%f</code>	Float (número reais)
<code>%c</code>	Caractere
<code>%s</code>	String (sequencia de caracteres)
<code>%e</code>	Número em notação científica
<code>%g</code>	Efetua a escrita de <code>%e</code> ou <code>%f</code> , no formato mais curto.

Função scanf()

Permite realizar entrada de dados através do teclado. Requer a utilização dos códigos de formatação de tipos de variáveis.

```
scanf (string_de_controle, lista_de_variáveis);
```

```
printf (“Digite um número: ”);
```

```
scanf (“%f ”, &N1);
```

Na função **scanf()** utiliza-se o **& (e comercial)** antes de cada variável para indicar o endereço de memória no qual o conteúdo da variável estará armazenado. Este é de uso obrigatório quando tratamos variáveis do tipo numérica (inteira ou real).

Regras para identificadores de variáveis

- O nome de uma variável deve descrever o que ela armazena.
- Caso o nome seja composto utilize o _ (underscore) para ligar as palavras ou a diferença entre maiúsculas e minúsculas para facilitar a leitura.
- O primeiro caracter de uma variável jamais pode ser um número.
- Não é permitido usar palavras reservadas da linguagem como nome de variáveis. Por exemplo não podemos chamar uma variável de if.

Exercício

Elabore um programa para somar dois números e exibir o resultado na tela.

Programa para adicionar dois números em C

```
#include <stdio.h>
int main(void)
{
    int A, B, Soma; // declaração de variáveis inteiras

    printf("Digite um numero inteiro: ");
    scanf("%d", &A); // recebe um inteiro decimal e armazena na variavel A
    printf("Digite um numero inteiro: ");
    scanf("%d", &B); // recebe um inteiro decimal e armazena na variavel B

    Soma = A + B; // Efetua adição de A com B e armazena na var. Soma

    printf("O valor e = %d", Soma); // Mostra mensagem com o resultado
    printf("\n"); // desloca o cursor para próxima linha;

    system("PAUSE"); // parada de tela
    return(0);
}
```

Exercício

Calcular o valor do salário líquido de um profissional horista, já calculados os descontos. Suponha que o usuário vai digitar um valor percentual para definir o desconto.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    float HorasTrab, ValorHora, PercDesc, TotalDesc, SalBruto, SalLiq;
    // Entrada: Recebendo variáveis de entrada
    printf("Quantas horas de trabalho? ");
    scanf("%f", &HorasTrab);
    printf("Qual o valor da hora? ");
    scanf("%f", &ValorHora);
    printf("Qual o percentual de desconto? ");
    scanf("%f", &PercDesc);

    // Processamento: Efetuando cálculo do salário
    SalBruto = HorasTrab * ValorHora;
    TotalDesc = (PercDesc/100) * SalBruto;
    SalLiq = SalBruto - TotalDesc;

    //Saída: Mostrando resultados na tela
    printf("Salario Bruto ....: %f\n", SalBruto);
    printf("Desconto .....: %f\n", TotalDesc);
    printf("Salario liquido ..: %f\n", SalLiq);
    system("PAUSE");
    return(0);
}
```


Formatando a precisão de valores com ponto flutuante

Para formatar um float, colocamos antes do f na expressão de controle o total de casas a serem apresentadas e o total de casas decimais.

Assim temos por exemplo:

`%8.2f`

que significa mostrar valores com um total de 8 dígitos, sendo que destes oito, dois deles são decimais. O ponto também ocupa um dígito. Neste caso temos 5 casas para a parte inteira, e duas casas decimais.

```
printf("Salario Bruto : %8.2f \n", SalBruto);  
printf("Desconto : %8.2f \n", TotalDesc);  
printf("Salario Liquido ...: %8.2f \n", SalLiq);
```

Estruturas de decisão

ou

Estruturas condicionais

Estrutura Condicional SE simples

Pseudocódigo

Sintaxe:

Se (condição) **então**
 {Instrução1};

Neste exemplo **somente uma instrução** será executada quando a condição for verdadeira, por isso não há necessidade de colocar início e fim.

Se (condição1) **então**
início

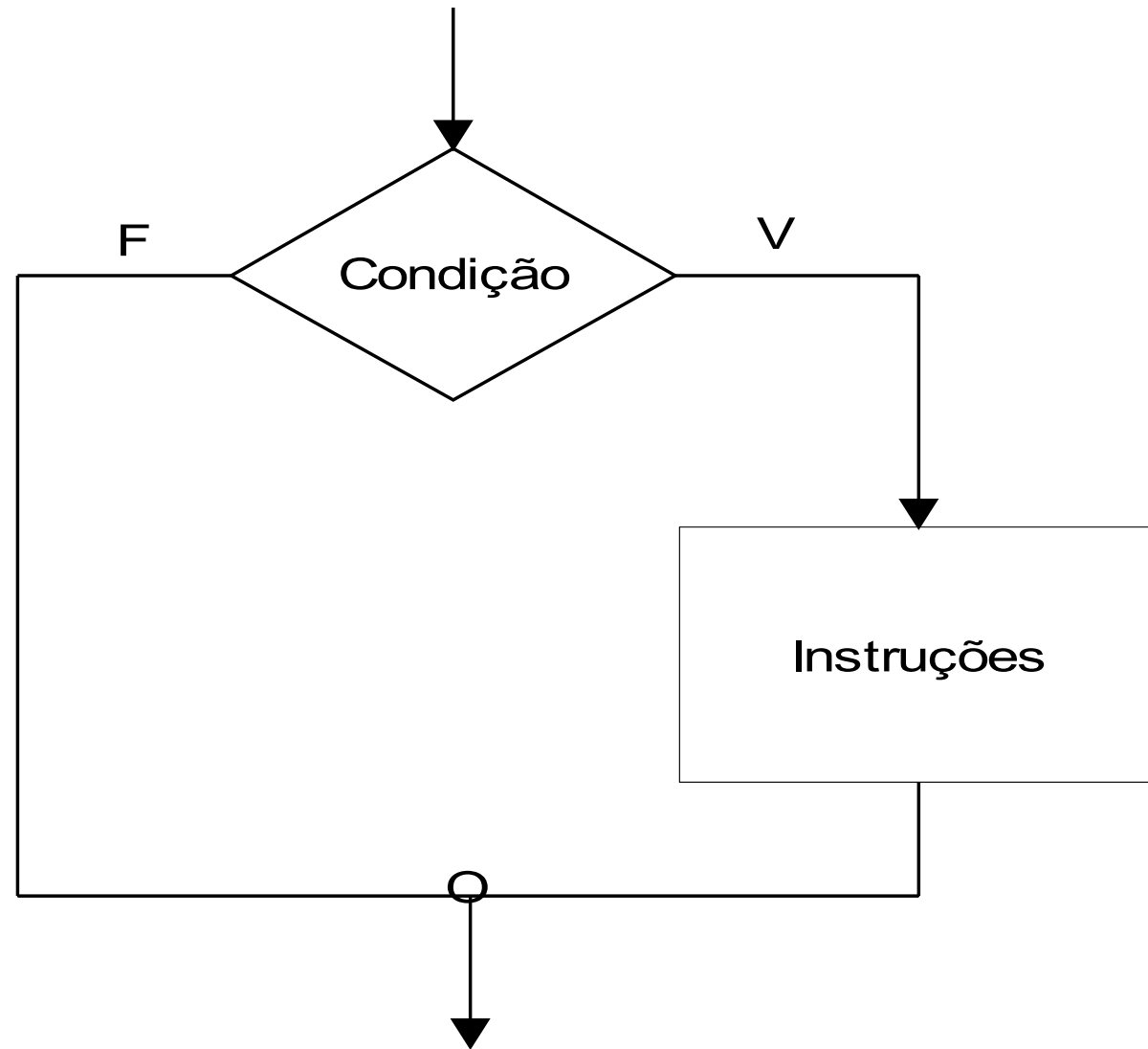
 {InstruçãoA};
 {InstruçãoB};

Fim;

Quando for necessário executar mais que uma instrução, devemos obrigatoriamente escrever **início** no começo do bloco de instruções que será executado e **fim;** ao terminar o bloco.

Estrutura Condicional SE Simples

Diagrama de Blocos



Estrutura Condicional SE simples

Exemplo:

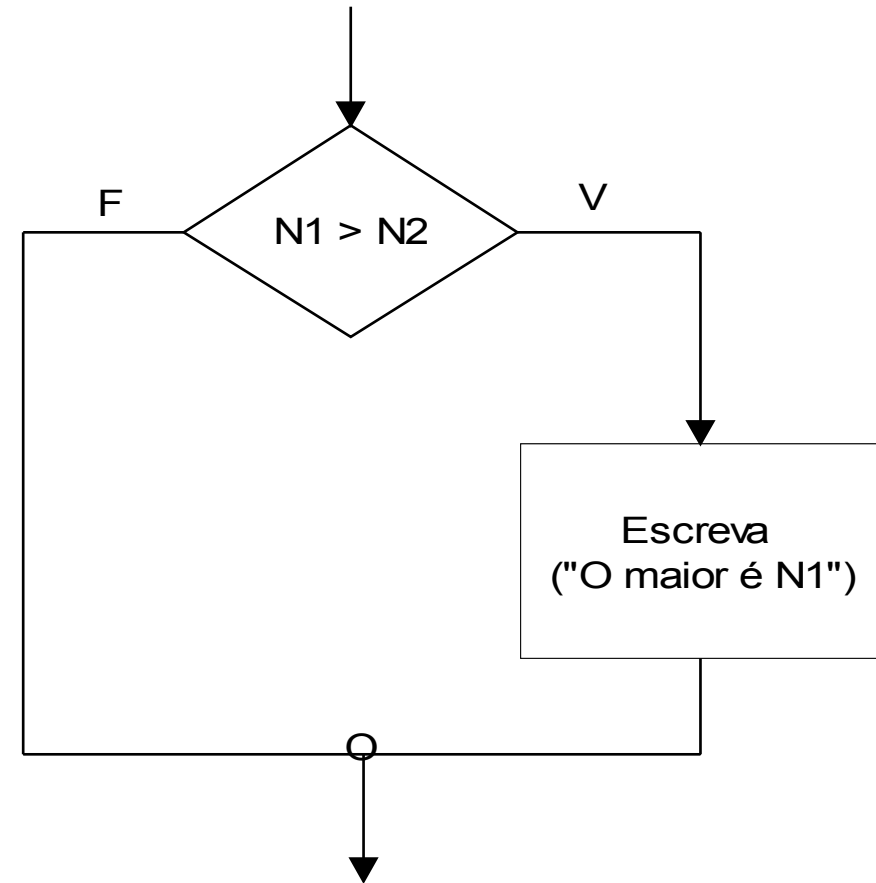
Se ($N1 > N2$)

Início

Escreva ("N1 é o maior");

Fim;

Observe que o comando só será executado quando a condição for Verdadeira, caso contrário, nada será feito, e o programa segue seu fluxo para a próxima instrução.



Estrutura de decisão simples

Linguagem C - if

Em Português Estruturado temos:

Se (condição) então

Início

instrução ou instruções para condição verdadeira;

Fim;

Em linguagem C o equivalente é:

if (condição)

{

instrução ou instruções para condição verdadeira;

}

Exercício – estrutura de decisão if

Elabore o diagrama de blocos, o algoritmo e o programa que receba dois valores e depois mostre-os na tela em ordem decrescente.

Estrutura condicional SE composta

Pseudocódigo

Sintaxe:

Se (condição) **então**

Início

{instruções}

Fim

Senão

Início

{instruções}

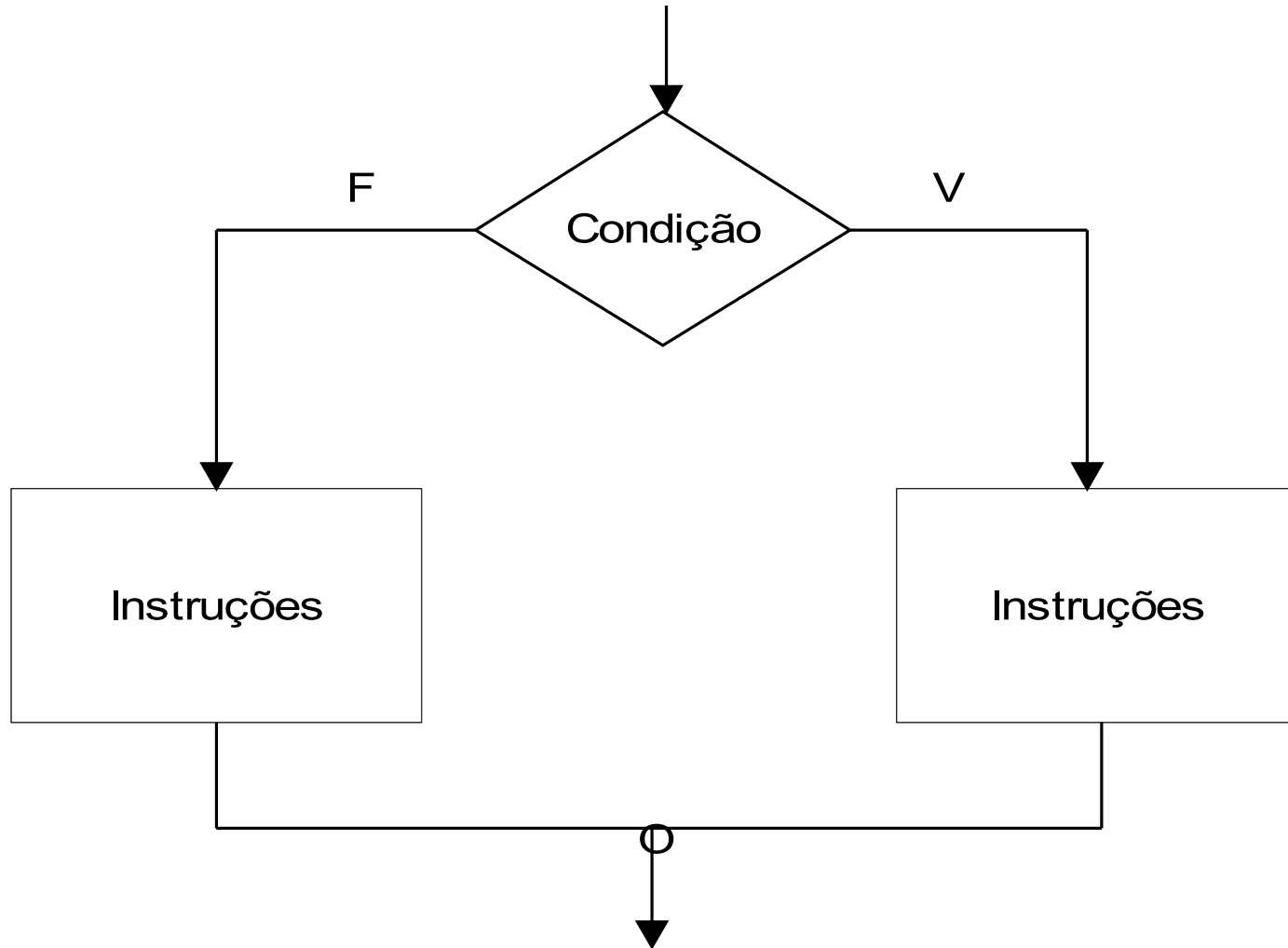
Fim;

- Na estrutura SE composta, temos instruções a serem executadas quando a condição for Verdadeira e também quando for falsa.

- A comando **Senão** determina o que será feito quando a condição for Falsa.

Estrutura Condicional SE composta

Diagrama de Blocos



Estrutura Condicional SE composta

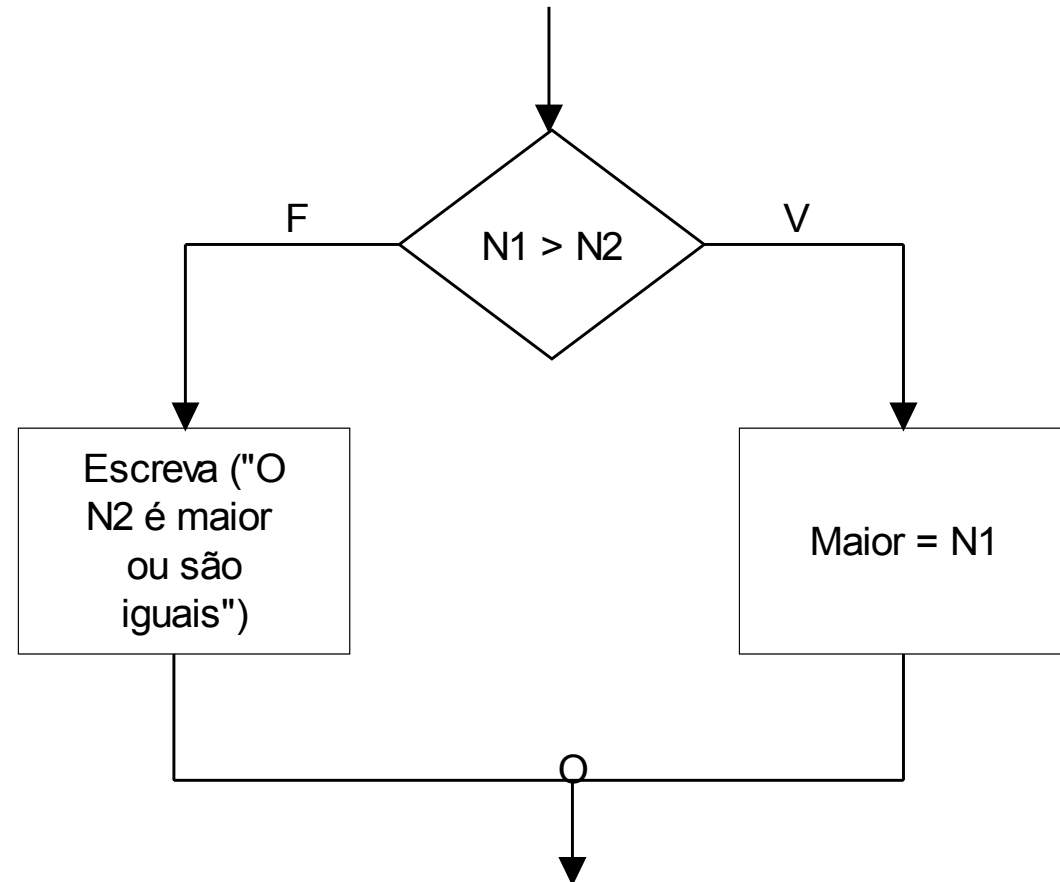
Exemplo:

Se ($N1 > N2$) **então**

$\text{Maior} \leftarrow N1$;

senão

 Escreva ("N2 é o maior ou
 são iguais.");



Estrutura de decisão composta

Linguagem C - if ... else

Em Português Estruturado temos:

Se (condição) então

Início

instrução ou instruções para condição **Verdadeira**;

Fim

Senão

Instrução ou instruções para condição **Falsa**;

Em linguagem C o equivalente é:

if (condição)

{

instrução ou instruções para condição **Verdadeira**;

}

else

instrução ou instruções para condição **Falsa**;

Exercício - if...else

Elabore o diagrama de blocos, o algoritmo e um programa que receba duas notas e calcule a média do aluno. Caso a média seja igual ou superior a 7,0 mostrar mensagem: “Aluno Aprovado”, senão mostrar mensagem “Aluno Reprovado”.

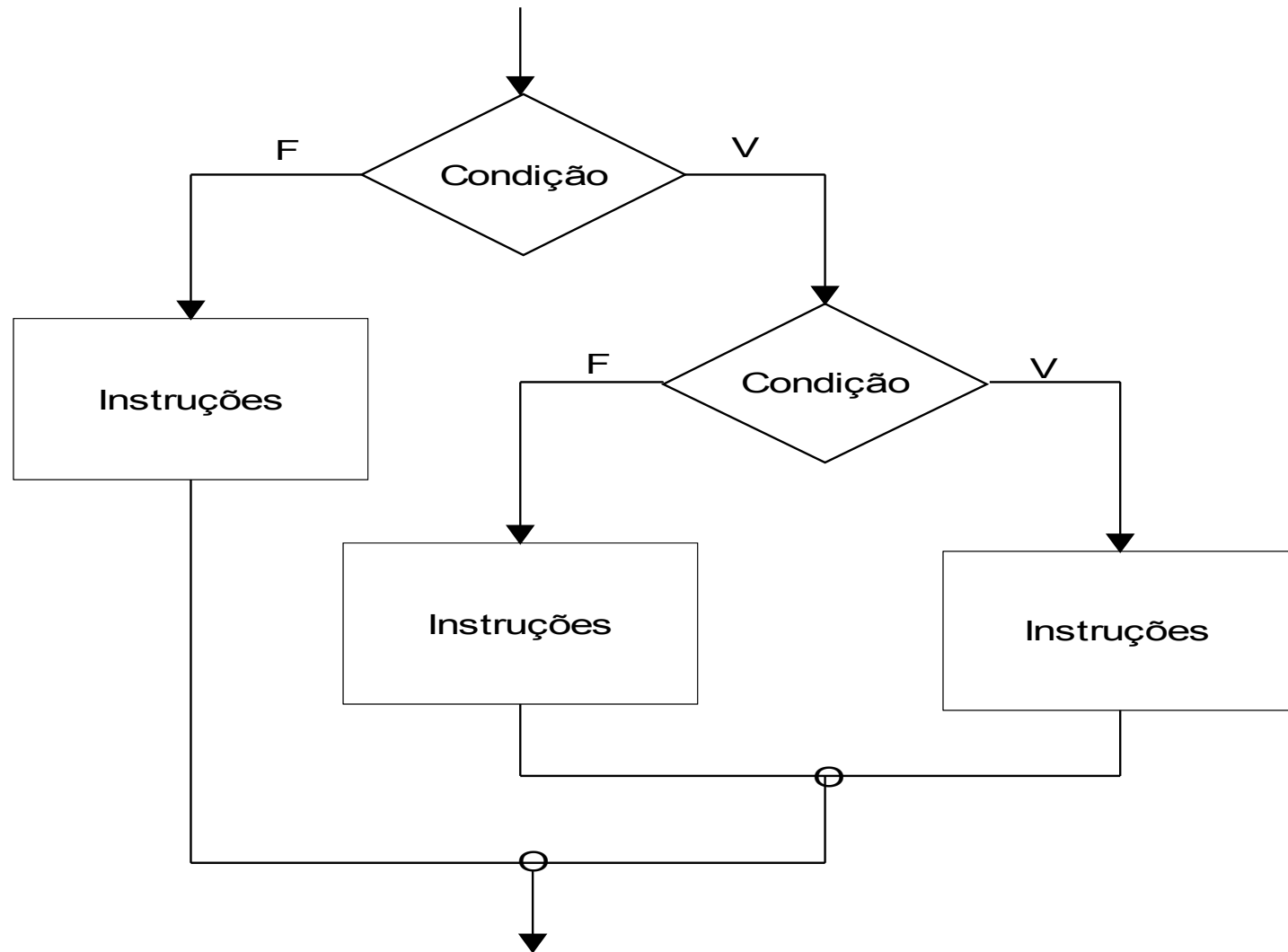
Estrutura Condicional SE Encadeada

Existem situações onde precisamos testar várias condições interdependentes a fim de poder executar determinadas instruções. Para ,tal, podemos encadear comandos SE.

```
Se (condição 1) então
    Se (condição2) então
        Instrução A;
    Senão
        Instrução B;
Senão
    Instrução C;
```

Estrutura Condicional Se Encadeada

Diagrama de Blocos



Estrutura Condicional SE Encadeada

Exemplo

Se (N1 < > N2) **então**

Se (N1 > N2) **então**

 Maior \leftarrow N1;

Senão

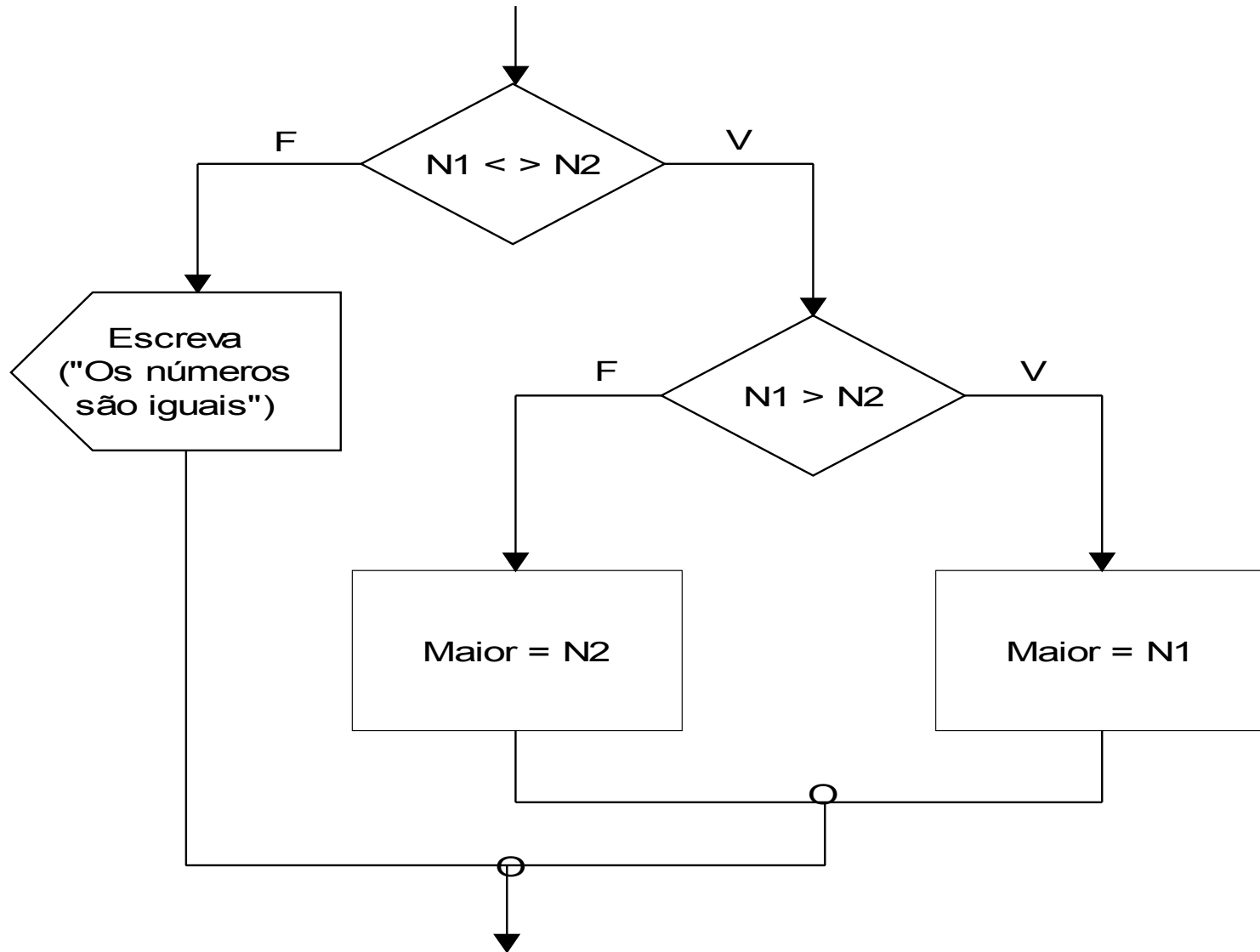
 Maior \leftarrow N2;

Senão

 Escreva (“Os números são iguais”);

Estrutura Condicional SE Encadeada

Exemplo



if encadeado – linguagem C

```
if (condição)
{
    comandos;
}
else if (condição)
{
    comandos;
}
else if (condição)
{
    comandos;
}
else
{
    comandos;
}
```

Múltipla Escolha com SE

Em certos casos, uma variável assume somente um valor para cada estrutura SE, sendo que as instruções a serem executadas dependem do valor da variável. Nesse caso, podemos usar várias estruturas SE, uma após a outra.

Sintaxe:

Se (condição1) **então**
 {Instrução};

Se (condição2) **então**
 {Instrução};

Se (condição3) **então**
 {Instrução};

Se (condição1) **então**
início
 {InstruçãoA};
 {InstruçãoB};
fim;

Se (condição1) **então**
início
 {InstruçãoC};
 {InstruçãoD};
fim;

Se (condição3) **então**
início
 {InstruçãoE};
 {InstruçãoF};
fim;

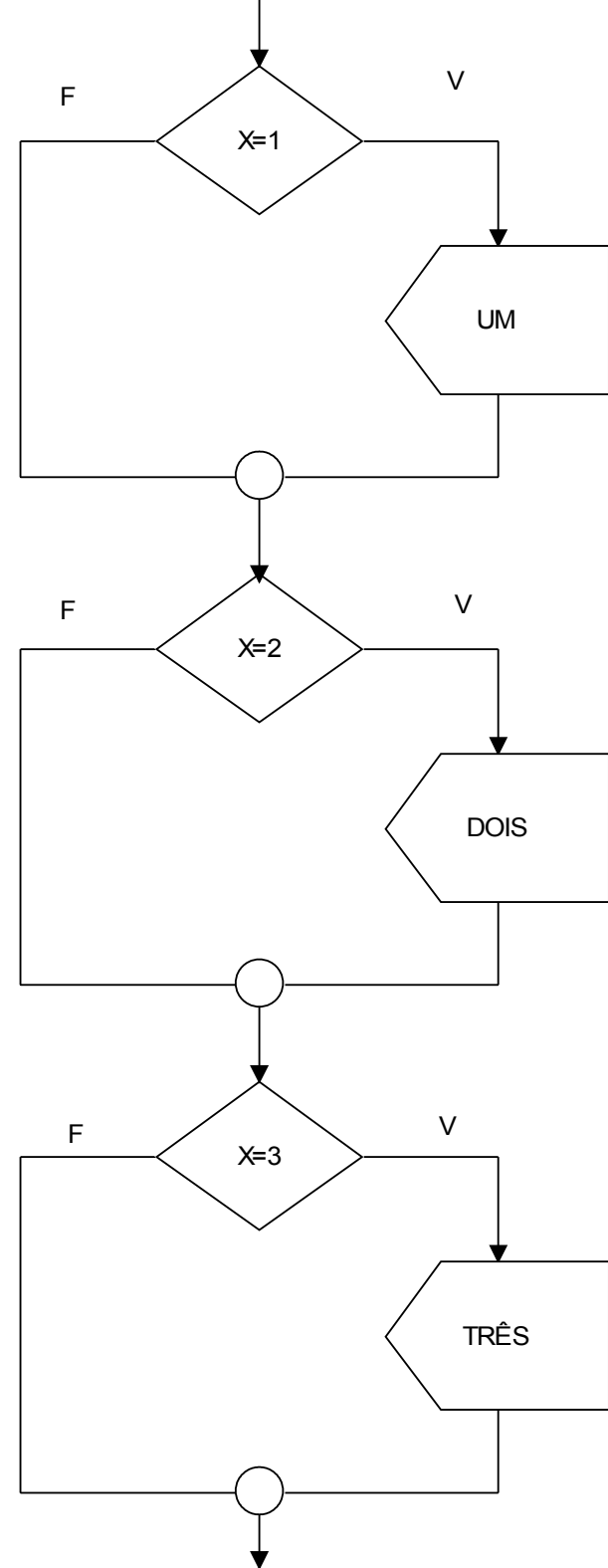
Múltipla escolha com SE

Execução da instrução

Se ($X = 1$) **então**
 Escreva (“Um”);

Se ($X = 2$) **então**
 Escreva (“Dois”);

Se ($X = 3$) **então**
 Escreva (“três”);



Estrutura Condicional de Múltipla Escolha

ESCOLHA/CASO

A estrutura ESCOLHA...CASO é uma variação simplificada da estrutura SE, muito utilizada na construção de menus.

A estrutura compara o conteúdo de uma variável com um valor constante, e caso a comparação resulte verdadeira, uma determinada ação é realizada.

Não são aceitas expressões condicionais, somente valores constantes.

Estrutura Condicional ESCOLHA/CASO

Sintaxe:

Escolha (Variável)

Inicio

Caso (Valor1): Instruções;

Caso (Valor2): Instruções;

Caso (ValorN): Instruções;

Fim;

Estrutura Condicional Escolha/Caso

Exemplo de execução da instrução:

Escolha (A)

Início

Caso (1): Escreva (“Um”);

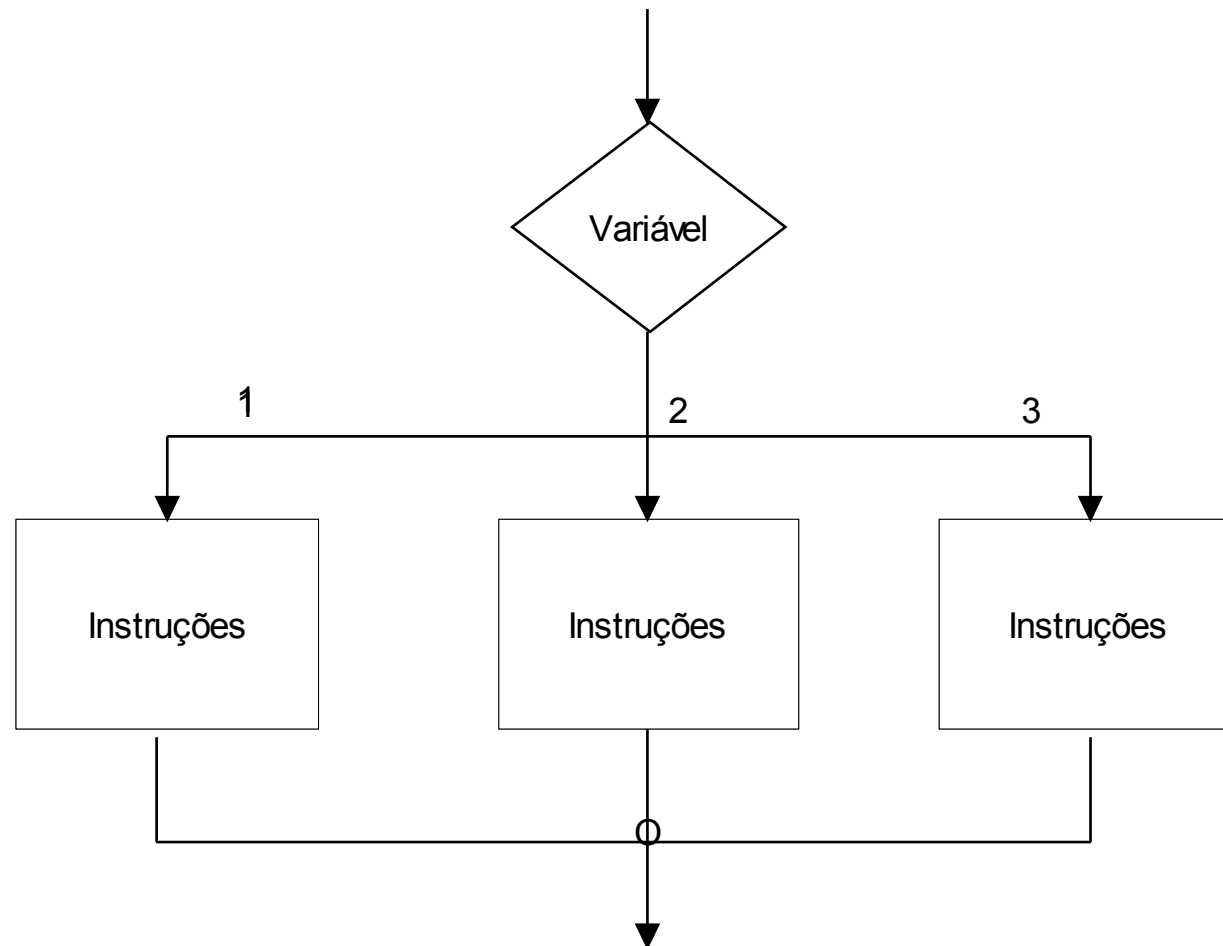
Caso (2): Escreva (“Dois”);

Caso (3): Escreva (“Três”);

Fim;

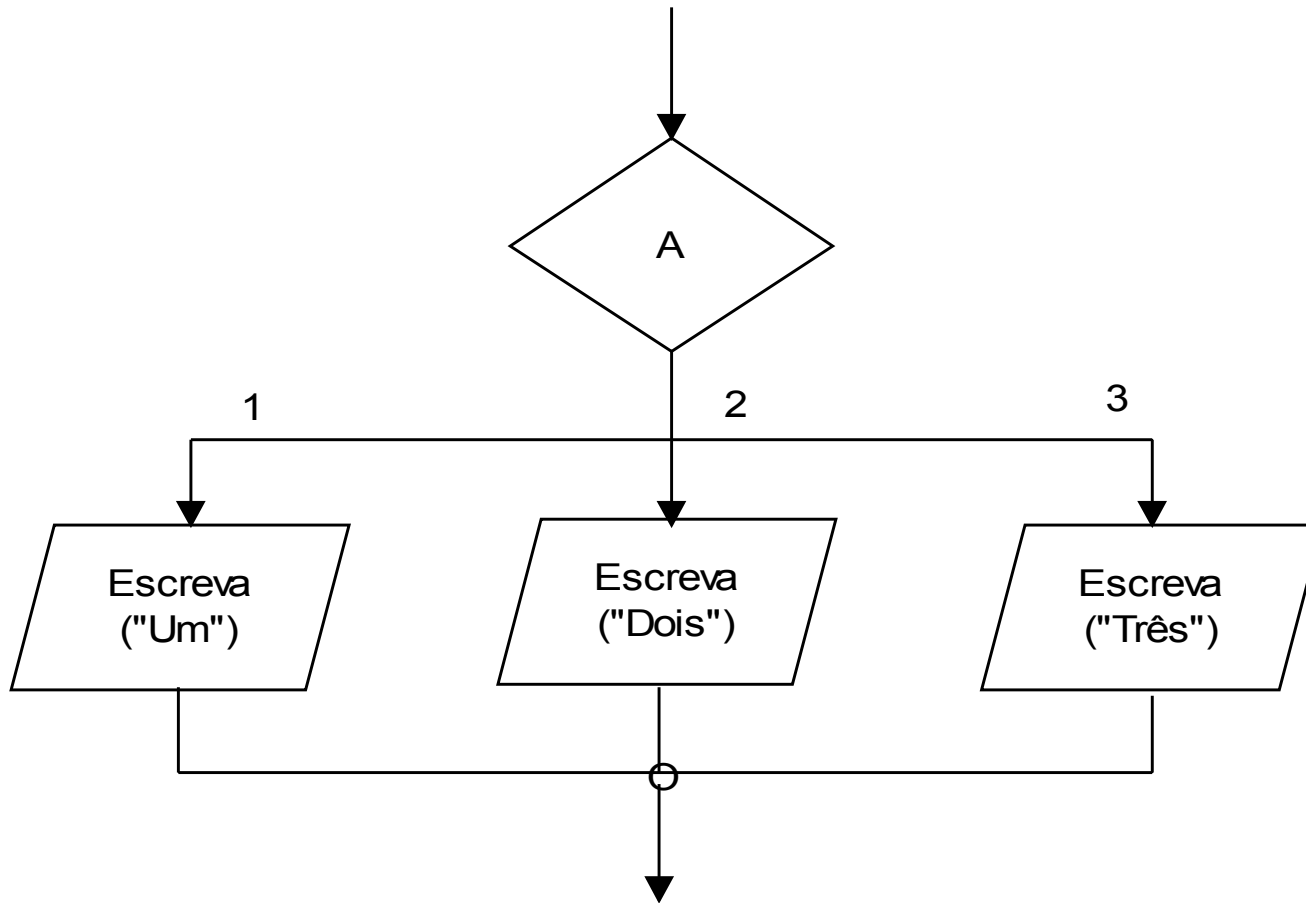
Estrutura Condicional Escolha/Caso

Diagrama de Blocos



Estrutura Condicional Escolha/Caso

Diagrama de Blocos



Linguagem C

Estrutura switch ...case

```
switch (variável)
{
    case constante1:

        comandos;

        break;

    case constante2:

        comandos;

        break;

    default

        comandos;
}
```

Exemplo switch...case

```
#include <stdio.h>
#include <stdlib.h>
int main (void )
{
    char opcao;
    puts ("Entre com uma letra:");
    opcao = getchar( );
    switch ( opcao )
    {
        case 'A' :
            printf ("Letra A\n");
            break;
        case 'B' :
            printf ("Letra B\n");
            break;
        case 'C' :
            printf ("Letra C\n");
            break;
        default :
            printf ("Não e A, B, nem C\n");
    }
    system("PAUSE");
    return 0;
}
```

Estruturas de Repetição

Estruturas de Repetição

Uma estrutura de repetição nos auxilia quando é preciso repetir certos comandos de um algoritmo um determinado número de vezes.

Estudaremos as estruturas de repetição:

PARA

ENQUANTO

FAÇA...ENQUANTO

Como determinar qual é a estrutura mais adequada para determinado programa?

→ Para determinar a estrutura mais adequada para um caso específico, é preciso saber o número de vezes que o trecho do programa será repetido ou então conhecer a condição para que a repetição ocorra.

Controlando o número de repetições

Quando conhecemos previamente quantas vezes o conjunto de instruções será repetido, podemos usar uma variável **contador**, a fim de controlar a estrutura.

O contador controla a repetição da estrutura até que o seu valor atinja o limite estipulado na condição da estrutura de repetição. Depois que o limite é ultrapassado, ele deixa de executar as instruções seguindo o fluxo do programa.

Contador

É uma variável cujo conteúdo é alterado pelo seu próprio valor adicionado ou subtraído de uma determinada constante (normalmente o valor da constante é 1).

O contador é muito utilizado para contar a quantidade de vezes que o programa executa as instruções de uma estrutura de repetição.

Exemplo:

$\text{Cont} \leftarrow \text{Cont} + 1;$

Acumulador

É a variável cujo valor é alterado pelo seu próprio valor adicionado do valor de outra variável.

Para evitar erros nos cálculos, o acumulador dever ser iniciado com um valor zero.

Sintaxe:

NomeAcumulador \leftarrow NomeAcumulador + Variável;

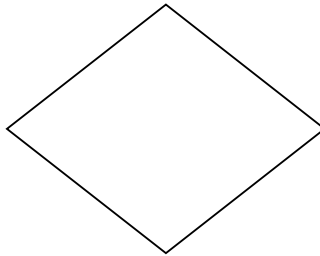
Exemplo:

Acum \leftarrow Acum + Valor;

É muito comum a utilização de acumuladores dentro de laços, a fim de acumular um valor para fazer um teste, ou um cálculo depois de sair do laço.

Simbologia para estruturas de repetição

O símbolo define onde a condição que controla o loop será testada.



Estrutura de Repetição PARA

(Repetição controlada por contador)

Estrutura de repetição que utiliza variável para controlar a contagem do loop, que ocorre de forma automática na própria estrutura.

É bem “enxuta”, pois a própria estrutura se encarrega de iniciar, incrementar e encerrar a variável que controla o laço de repetição.

É a estrutura mais indicada quando se conhece previamente o número de vezes que a repetição será executada.

Estrutura de Repetição PARA

Para (Variável = Valor Inicial **até** Valor Final **passo** + n) **faça**
Início

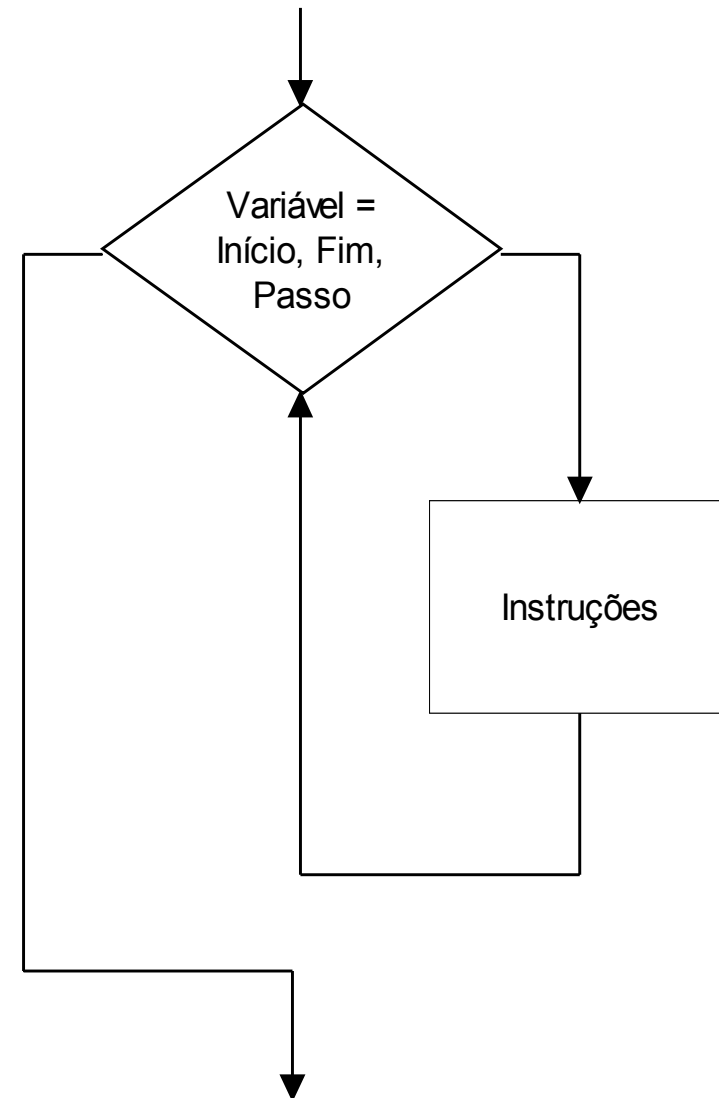
{instruções}

Fim;

Observações:

O passo é o valor que será incrementado, ou decrementado a cada repetição no contador que controla o loop.

Normalmente, quando o passo é + 1, não precisa ser declarado, e o incremento +1 será assumido automaticamente.



Estrutura de Repetição Para

Exemplo de execução da instrução:

Para (I = 1 **até** 10) **faça**

Início

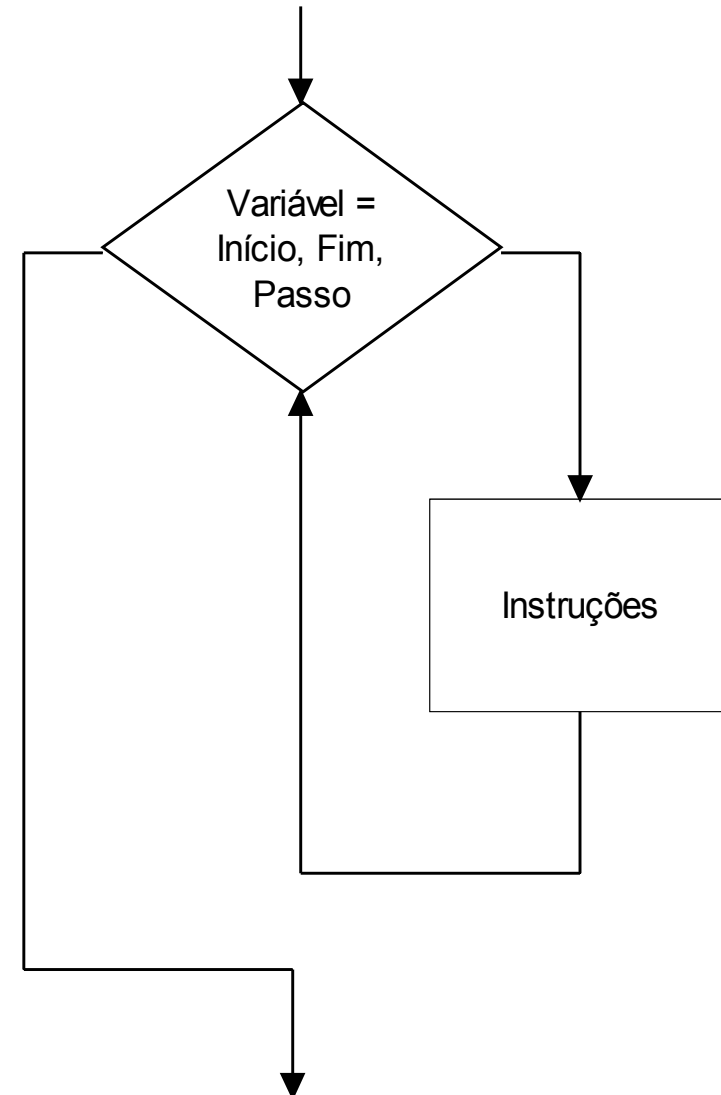
X = Y;

Z = X;

Fim;

Neste exemplo, o passo é +1 e portanto foi omitido.

O valor da variável de controle I é incrementado cada vez que ocorre uma repetição, logo esta estrutura efetuará 10 repetições.



Estrutura de repetição FOR

Linguagem C

Executa um conjunto de instruções, um determinado número de vezes, enquanto uma variável de controle é incrementada ou decrementada a cada passagem pelo loop.

```
for(valor_inicial; condição_final; valor_incremento)  
  
    comando;
```

Exercício – usando for

Faça um programa em linguagem C que calcule a tabuada de multiplicação para um número digitado pelo usuário.

```
/* Tabuada For */
#include <stdio.h>
#include <conio.h>
int main(void)
{
    int I, Numero, Res;

    printf("Digite o valor para calcular a tabuada ");
    scanf("%d",&Numero);

    for (I = 1; I <= 10; I++)
    {
        Res = Numero * I;
        printf("%d X %d = %d ",Numero, I, Res);
        printf("\n");
    }

    getch();
    return(0);
}
```

Conceito de Flag

Nem sempre é possível usar um contador para finalizar uma repetição. Existem casos em que não sabemos de antemão quantas vezes a repetição será feita, pois o usuário é quem vai decidir isso. Nesses casos utilizamos um flag.

Chamamos de flag ao evento que deve ocorrer com a variável de controle para indicar a finalização de uma estrutura de repetição.

Por exemplo, podemos fazer uma repetição, que será realizada enquanto o valor da variável de controle for diferente de 'S'.

Exemplo de flag

Neste trecho de algoritmo, o usuário decide quando deve sair ao teclar o caracter S para a variável Resp. Quando este evento ocorre, a repetição é finalizada.

Faça
Início

```
Escreva("Digite o nome do cliente: " );  
Receba(Cliente);  
Escreva("Tecle S para sair ou C para novo cadastro: " );  
Receba(Resp);
```

Fim

Enquanto (Resp <> 'S');

Estrutura de repetição ENQUANTO

(Teste lógico no início do laço)

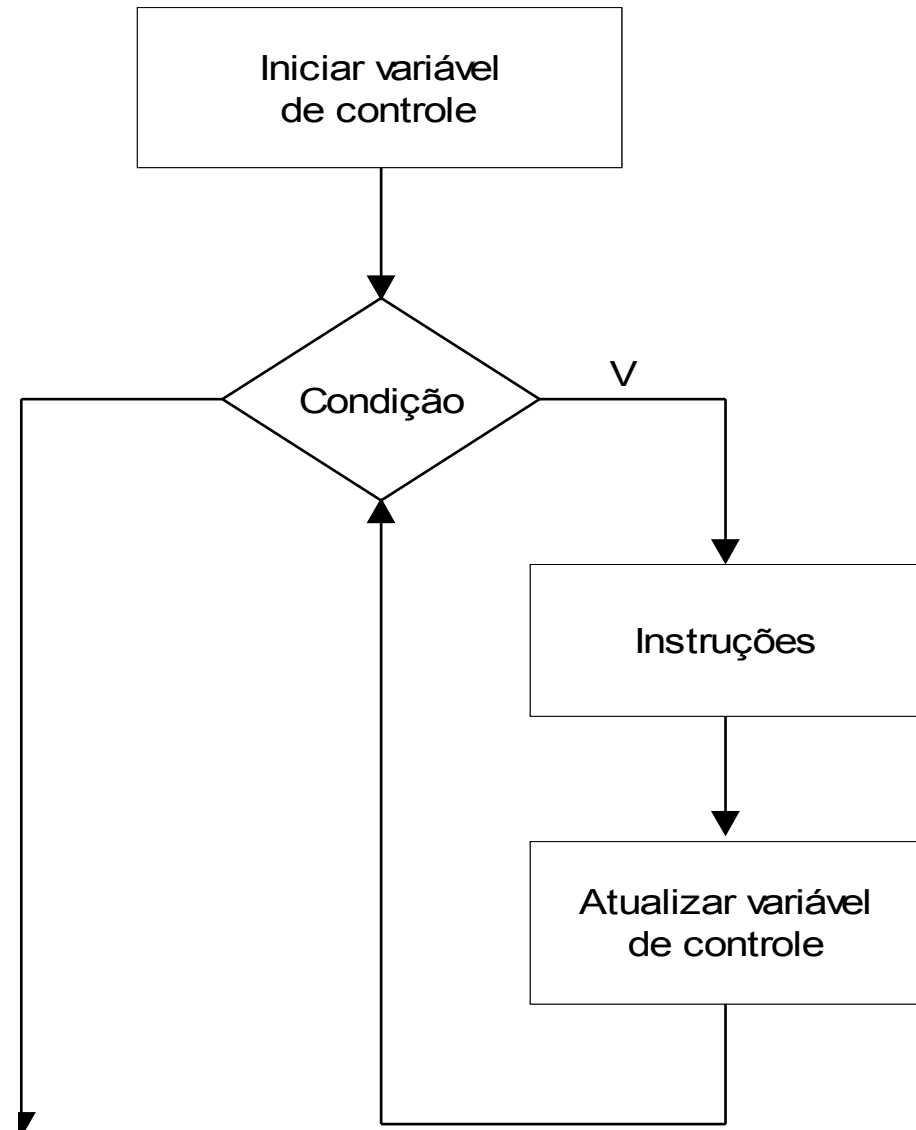
Esta estrutura testa uma condição no início do laço e pode executar um determinado número de instruções, enquanto a condição verificada para a variável de controle permanecer verdadeira. Quando a condição se tornar falsa, o processamento é desviado para fora do laço.

Caso a condição seja falsa logo no início, as instruções do laço são ignoradas.

Estrutura de Repetição ENQUANTO

Sintaxe:

Enquanto {Iniciar a variável de controle}
(condição for verdadeira)**faça**
Início
 {instruções}
 {Atualizar a variável de controle}
Fim;



Estrutura de Repetição ENQUANTO

Exemplo:

$X \leftarrow 1$; //Iniciando a variável que controla o loop

Enquanto ($X < 10$) **faça**
Início

Escreva("O valor de X é = ",X);

$X = X + 1$; //atualizando a variável de controle

Fim;

Loop while

Linguagem C

```
while (condição)
```

```
{
```

```
    /*comandos*/
```

```
}
```

Exemplo While

```
#include <stdio.h>
#include <conio.h>
int main(void)
{

    int contador = 1; // Inicializa a variável de controle

    while (contador <= 100) // Testa a variável de controle
    { // Início do loop

        printf("%d ", contador); // Executa os comandos

        contador++; // incrementa a variável de controle

    } // Fim do loop

    getch();
    return 0;
}
```

Estrutura de Repetição Faça/Enquanto (Teste lógico no final do laço)

Na estrutura Faça...Enquanto seu funcionamento é controlado por decisão, porém as instruções são executadas no mínimo uma vez, antes de testar a condição da variável que controla o laço no final do comando.

Faça...Enquanto

Sintaxe:

{Iniciar a variável de controle}

Faça

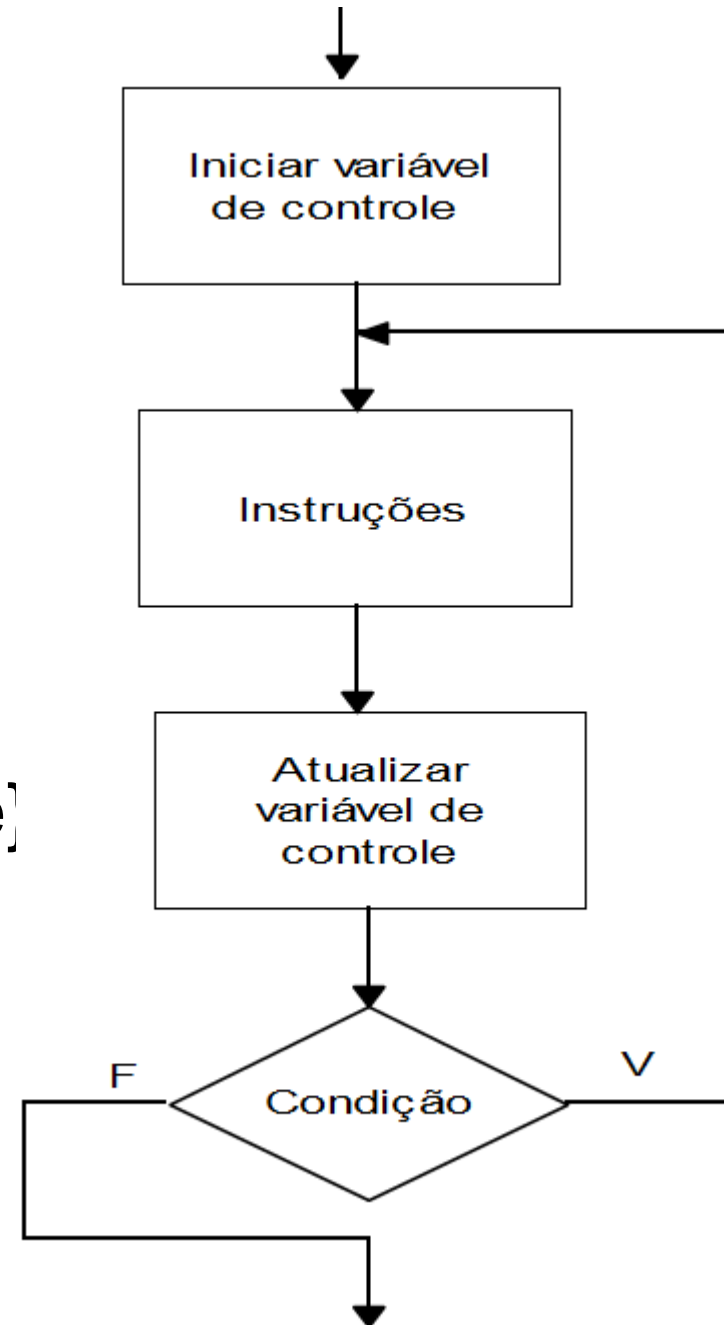
Início

{Instruções}

{Atualizar a variável de controle}

Fim

Enquanto (condição for verdadeira);



Faça...Enquanto - Execução

Execução da instrução:

$X \leftarrow 1$; // Inicializando a variável de controle

Faça
Início

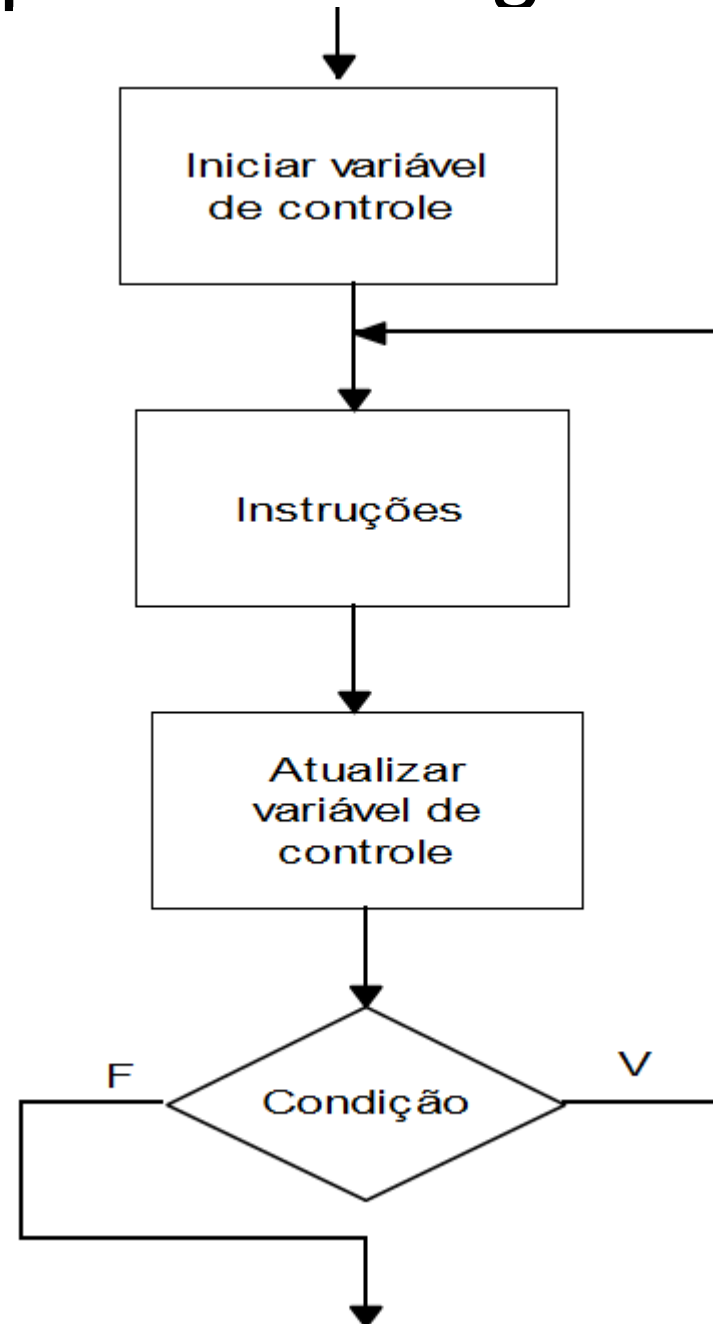
Escreva("O valor de X é = ",X);

$X = X + 1$; //atualizando a variável de controle

Fim

Enquanto ($X \leq 10$);

Faça...Enquanto - Diagrama de Blocos



LOOP DO ... WHILE – Linguagem C

No comando do ... while a avaliação da expressão condicional ocorre sempre após a execução do comando.

Isto faz com que o comando sempre execute pelo menos uma vez antes de realizar o teste da condição. Vejamos o exemplo:

```
#include <stdio.h>
#include <conio.h>
int main (void)
{
    int i = 1;    // inicializando variavel de controle
    do
        printf ("%d \n", i);
    while ( ++i <= 10 ); // condição

    getch();
    return(0);
}
```

Exercício do...while

Modifique o programa da tabuada para que proporcione ao usuário uma opção para calcular outra tabuada novamente. Caso o usuário queira calcular outra tabuada ele deve digitar 1, caso queira encerrar o programa, deve digitar 2.

```
#include <stdio.h>
#include <stdlib.h> // biblioteca que permite usar o comando system("cls")
int main(void)
{
    int l, Numero, Res;
    int Continuar;

    do
    {
        system("cls"); // limpa a tela - somente para ambiente windows
        printf("Digite o valor para calcular a tabuada ");
        scanf("%d",&Numero);

        for (l = 1; l <= 10; l++)
        {
            Res = Numero * l;
            printf("%d X %d = %d ",Numero, l, Res);
            printf("\n\n");
        }

        printf("Tecle 1 para fazer outra tabuada ou 2 para sair ");
        scanf("%d",&Continuar);

    }
    while(Continuar==1);

    return(0);
}
```

Exercício Extra

Uma classe tem 20 alunos.

Em uma determinada disciplina, os alunos fazem um total de 3 provas, valendo 100 pontos cada uma.

A nota final de um aluno é a média dos dois melhores resultados, dentre as três notas recebidas.

Deseja-se construir um programa para ler as notas atribuídas a três provas de cada aluno e calcular sua nota final.

Calcular também a nota média da sala.

Funções de entrada de caracteres

Linguagem C

Função gets

Processa tudo que foi digitado até que a tecla ENTER seja pressionada.

Suponha que o usuário vai digitar seu nome completo. Nesse caso para armazenarmos o valor é interessante usar gets para armazenar o nome inteiro inclusive com o sobrenome.

Exemplo:

```
printf ("Digite seu nome:");  
gets(nome);
```

FUNÇÃO `getchar()`

Funciona de forma semelhante a `gets`, porém é usada para receber apenas um caracter, enquanto que `gets` lê uma string toda.

A leitura do caracter digitado ocorre quando o usuário pressionar a tecla <Enter>.

FUNÇÃO getch()

A função getch() edita um caracter do teclado e permite que ele seja mostrado na tela do computador sem que seja necessário o pressionamento da tecla ENTER por parte do usuário. Isso já ocorre de forma automática.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int main (void )
{
    char ch;
    printf ("Digite uma tecla:");
    ch = getch( );
    printf ("\n A tecla que você pressionou foi %c. \n", ch);

    system("PAUSE");
    return 0;
}
```

FUNÇÃO getch()

Permite que o usuário forneça um caracter através do teclado. Este caracter não será mostrado na tela do computador.

Não é necessário pressionar ENTER, pois a passagem para a próxima ocorre automaticamente.

Também usamos esta função como recurso de parada temporária da execução do programa, semelhante ao comando Readkey da linguagem Pascal.

Funções de Saída de Dados Linguagem C

FUNÇÃO puts()

Representa a saída de uma única STRING por vez, seguida do caracter de nova linha, ou seja, a função muda de linha automaticamente sem que haja a necessidade do pressionamento da tecla de entrada de dados, ENTER.

Exemplos:

```
puts ("O rato");  
puts ("roeu a roupa");  
puts ("do rei de Roma.");
```

FUNÇÃO putchar ()

Representa a saída de apenas um caracter na tela do computador e não acrescenta uma nova linha automaticamente como a função puts().

As instruções seguintes são equivalentes:
putchar ('c');

printf ("%c", 'c');

Vetores

Variáveis indexadas unidimensionais

Vetor

Um vetor é uma estrutura de dados indexada, que pode armazenar múltiplos valores do mesmo tipo.

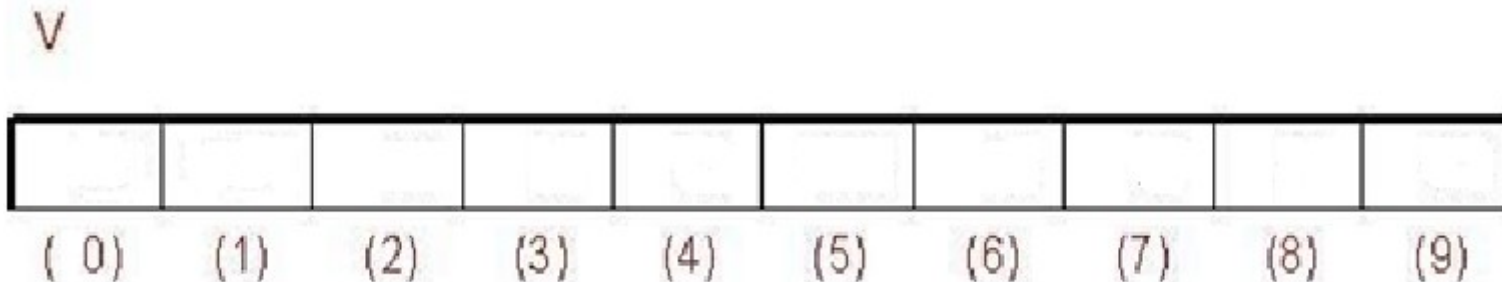
Facilita a manipulação de um grande conjunto de dados do mesmo tipo, declarando apenas uma variável.

Pseudocódigo:

NomeDoVetor: Vetor[inicio..final] de tipodevariavel

Exemplo:

V: vetor[0..9] de Real;



Índices do vetor

V

8.5	10.0								
(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)

Os endereços de armazenamento (índices) de um vetor são numerados.

Neste exemplo o vetor parte do índice zero e sua posição final é o índice 9. Sendo assim temos um total de 10 posições para armazenamento.

Atribuindo valores a um vetor

Para atribuir valor a um vetor, devemos sempre informar o nome do vetor e o índice.

Por exemplo:

$V[0] \leftarrow 8.5;$

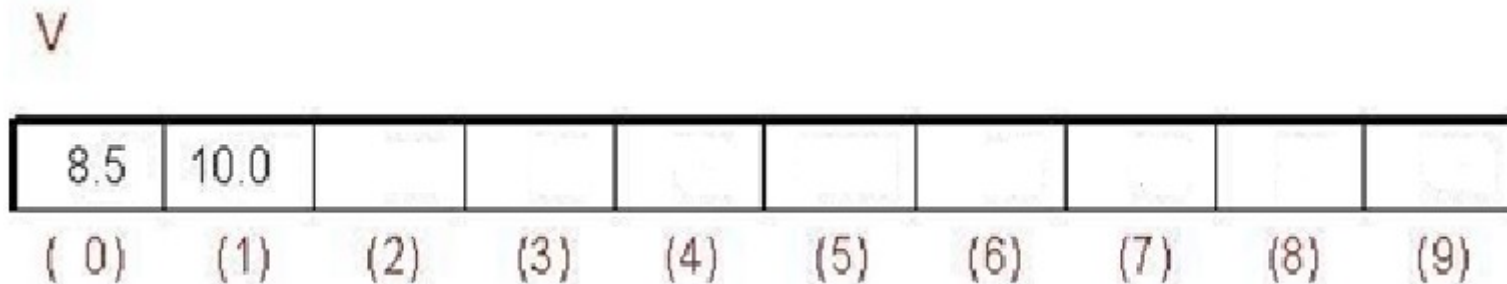
$V[1] \leftarrow 10.0;$

Para acessar os valores, devemos colocar o nome do vetor seguido da posição.

Vetor em C

Em linguagem C um vetor é indexado a partir da posição zero.

Declaração do vetor: `float V[10];`



Exemplo 1: Vetor em C

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int notas[5] = {18, 12, 95, 99, 22}; // declarando e inicializando o vetor

    printf("Valores do Vetor \n");

    printf("notas[0] %d\n", notas[0]);
    printf("notas[1] %d\n", notas[1]);
    printf("notas[2] %d\n", notas[2]);
    printf("notas[3] %d\n", notas[3]);
    printf("notas[4] %d\n", notas[4]);
    system("PAUSE");
}
```

Exemplo 2: Vetor em C

```
/* Calculo de Media da classe para 4 alunos */
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    float Media[4];
    float MediaSala, SomaNota = 0;
    int I;
    printf("\nCalculo de Media\n\n");
    for (I = 0; I <= 3; I++)
    {
        printf("Informe a %da. nota: ", I+1);
        scanf("%f", &Media[I]);
        SomaNota = SomaNota+Media[I];
    }
    MediaSala = SomaNota / 4;
    printf("\nA media da classe equivale a: %5.2f\n\n", MediaSala);
    system("PAUSE");
    return(0);
}
```

Exercício - Vetores

Elaborar um programa que some as posições correspondentes de dois vetores com 4 elementos cada um. Considere que os valores serão inseridos pelo usuário.

```

#define MAX 4
#include <stdio.h>
#include <conio.h>
int main (void)
{
    int i, vetor1[MAX], vetor2[MAX];
    printf("Somando dois vetores\n\n");
    printf("\n\nDigite os dados para o vetor 1 \n\n");
    for ( i=0; i<MAX; i++ )
    {
        printf ("vetor1[%d]=", i);
        scanf ("%d", &vetor1[ i ]);
    }
    printf("\n\nDigite os dados para o vetor 2 \n\n");
    for ( i=0; i<MAX; i++ )
    {
        printf ("vetor2[%d]=", i);
        scanf ("%d", &vetor2[ i ]);
    }
    printf("\n\n*****Somatoria*****\n\n");
    for ( i=0; i<MAX; i++ )
        printf ("vetor1[%d] + vetor2[%d] = %d\n", i, i, vetor1[ i ] + vetor2[ i ]);
    getch();
    return(0);
}

```

Exercício Extra - Vetor

Utilize uma variável do tipo vetor para resolver o problema:

Uma classe tem 20 alunos. Em uma determinada disciplina, os alunos fazem um total de 3 provas, valendo 100 pontos cada uma. A nota final de um aluno é a média dos dois melhores resultados, dentre as três notas recebidas.

- Deseja-se construir um programa para ler as notas atribuídas para as três provas de cada aluno e calcular sua media final. Caso o aluno tenha media final maior que 70 pontos o programa deve avisar que o aluno está aprovado, caso contrário emitir mensagem dizendo que o aluno está reprovado na disciplina.
- Calcular a nota média da classe, a maior média e também a menor média obtida pelos alunos .

Matrizes

Variáveis indexadas multidimensionais

Matriz

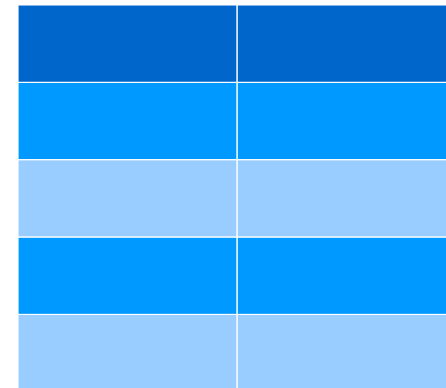
Chamamos de matriz a uma estrutura de dados homogênea com duas ou mais dimensões. Na prática, as matrizes formam tabelas na memória.

Pseudocódigo:

M:Matriz[inicioL...finalL,inicioC..finalC] de tipodedado;

Exemplo:

Representação Gráfica



Matriz em C

Chamamos de matriz a uma estrutura de dados homogênea com duas ou mais dimensões. Na prática, as matrizes formam tabelas na memória.

Exemplo de declaração de matriz com 2 dimensões:

```
float Media[5][2];
```

O valor 5 representa a quantidade de linhas.

O valor 2 representa a quantidade de colunas.

Essa matriz é do tipo 5 X 2 e tem capacidade para armazenar até 10 elementos do tipo float.

/* EXEMPLO DE MATRIZ:

Armazenamento de notas para 5 alunos com 2 notas por aluno */

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    float Media[5][2], N;
    int I, J;
    printf("\nLeitura e apresentacao de notas armazenadas na matriz\n");
    /*Leitura das notas dos alunos */
    for (I = 0; I <= 4; I++)
    {
        printf("\nInforme as notas do %do. aluno: \n\n", I+1);
        for (J = 0; J <= 1; J++)
        {
            printf("Nota %d: ", J+1);
            scanf("%f", &N);
            Media[I][J] = N;
        }
    }

    printf("\n\n***** Saída de Notas ***** \n\n");

    for (I = 0; I <= 4; I++)
    {
        printf("\nAs notas do aluno %d sao : \n\n", I+1);
        for (J = 0; J <= 1; J++)
            printf("Nota %d: %5.2f\n", J+1, Media[I][J]);
    }
    system("PAUSE");
    return(0);
}
```

/* Faça o fluxograma e o teste de mesa para o programa da MATRIZ abaixo */

```
#include <stdio.h>
#include <conio.h>
int main (void )
{
    int matriz[2][2],i, j;
    printf ("\nDigite valor para os elementos da matriz\n\n");

    for ( i=0; i<3; i++ )
        for ( j=0; j<3; j++ )
        {
            printf ("\nElemento[%d][%d]", i, j);
            scanf ("%d", &matriz[ i ][ j ]);
        }

    for ( i=0; i<3; i++ )
        for ( j=0; j<3; j++ )
            if ( i == j )
                printf ("\n%d", matriz[ i ][ j ]);
    getch();
    return(0);
}
```

Registros - Structs

Matrizes heterogêneas

Estrutura -struct

Para resumir, struct é uma estrutura de dados em formato de tabela que tem capacidade de trabalhar com tipos de dados diferentes. Desta forma, podemos dizer que uma struct é uma estrutura de dados heterogênea.

As structs da linguagem C são o equivalente ao que se denomina registros em outras linguagens. Os membros das structs, por sua vez, são o que se denomina campos em em outras linguagens.

Na realidade, embora a nomenclatura seja diferente, os conceitos são exatamente os mesmos entre struct e registros.

Sintaxe - struct

```
struct<identificador>
{
    <listagem dos tipos e membros>;
}
struct <identificador> <variavel>;
```

Exemplo:

```
struct CadastroAluno
{
    char nome[50];
    float nota1;
    float nota2;
};
struct CadastroAluno ALUNO;
```

/* Exemplo do uso de uma struct */

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    struct CadastroAluno
    {
        char Nome[40];
        float Nota1;
        float Nota2;
    };
    struct CadastroAluno Aluno;

    printf("\nCadastro de Aluno\n\n");
    printf("Informe o Nome .....: ");
    fflush(stdin); fgets(Aluno.Nome, 40, stdin);
    printf("Informe a 1a. nota ...: ");
    scanf("%f", &Aluno.Nota1);
    printf("Informe a 2a. nota ...: ");
    scanf("%f", &Aluno.Nota2);
    printf("\n");
    printf("\nLendo os dados da struct");
    printf("Nome .....: %s\n", Aluno.Nome);
    printf("Nota 1 ...: %6.2f\n", Aluno.Nota1);
    printf("Nota 2 ...: %6.2f\n", Aluno.Nota2);
    printf("\n\n\n");
    system("PAUSE");
    return(0);
}
```


Arquivos

Arquivos de dados

Um arquivo é um conjunto de registros armazenados em uma mídia.

Vantagens da utilização de arquivos:

- As informações são armazenadas na mídia de forma permanente, diferentemente de informações armazenadas na memória RAM que são perdidas quando o programa termina de executar.
- As informações podem ser consultadas a qualquer momento.
- A quantidade de informações armazenadas é maior do que aquela que poderia ser armazenada em uma tabela na memória, limitada apenas ao tamanho meio físico usado para gravação.

Arquivos

Conceitos Importantes

- ✓ Arquivo é um conjunto de registros.
- ✓ Registro é um conjunto de campos.
- ✓ Cada campo contém um tipo de dado
- ✓ Arquivos são manipulados usando-se um ponteiro de registro.
- ✓ O ponteiro “aponta” para a localização do registro.
- ✓ Chamamos de “registro corrente”, ao registro que está sendo manipulado no momento.

Campo

Denominamos campo a um espaço reservado na memória para receber informações.

Exemplo: Campo Nome

Representação do campo na memória:

NOME

José da Silva

Registro

Um conjunto de campos formam um de registro.

Exemplo: Registro de clientes, sendo um conjunto de campos formado pelos campos codigo, nome, endereço e telefone de um determinado cliente.

CODIGO	NOME	ENDEREÇO	TELEFONE
00001	MARIA DA SILVA	RUA DAS MARIAS, 1986	9765-9899

Arquivo

É um conjunto de registros armazenados de forma permanente em uma mídia digital.

Exemplo: O arquivo de Clientes da Empresa.

ARQCLIENTES

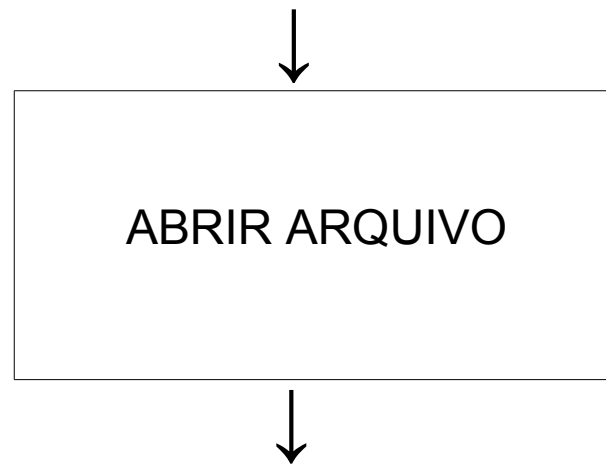


Abertura de Arquivos

Para trabalhar com um arquivo, primeiramente precisamos ABRIR o arquivo.

O processo de abertura de arquivos aloca o periférico em que o arquivo se encontra, prepara-o para leitura/gravação.

Símbolo de abertura de arquivo:

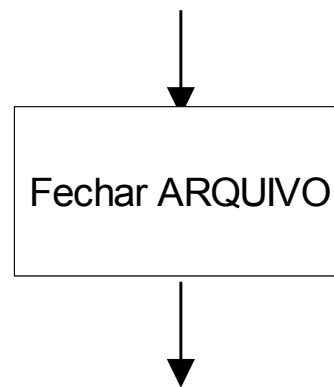


www.intellectuale.com.br

Fechamento de Arquivos

O processo de fechamento de um arquivo libera o periférico que estava sendo usado.

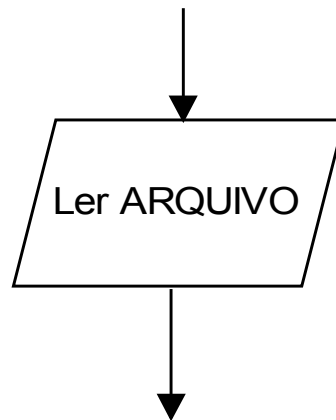
Símbolo de fechamento de arquivo



Leitura de Arquivos

Depois de abrir o arquivo, é necessário fazer a leitura dos dados que estão armazenados na mídia e transferi-los para memória.

Representação de leitura de arquivo:



BOF, EOF e Ponteiro

Ao abrir um arquivo, o “ponteiro” é posicionado automaticamente no primeiro registro, ou seja, no início do arquivo.

A área de início de arquivo é denominada BOF – (Begin of File).

O final do arquivo é denominado EOF - End Of File.

Representação gráfica da leitura de um arquivo

BOF (Início
do Arquivo)

→
PONTEIRO

COD-CLI	NOME	ENDERECO	TELEFONE
00001	JOSE DA SILVA	RUA CLELIA 95	6534-3000
00002	MARIA CLARA	RUA MIRANDA 77	6543-8561
00003	MÔNICA	RUA IGUATEMI 142	2288-8907
*			

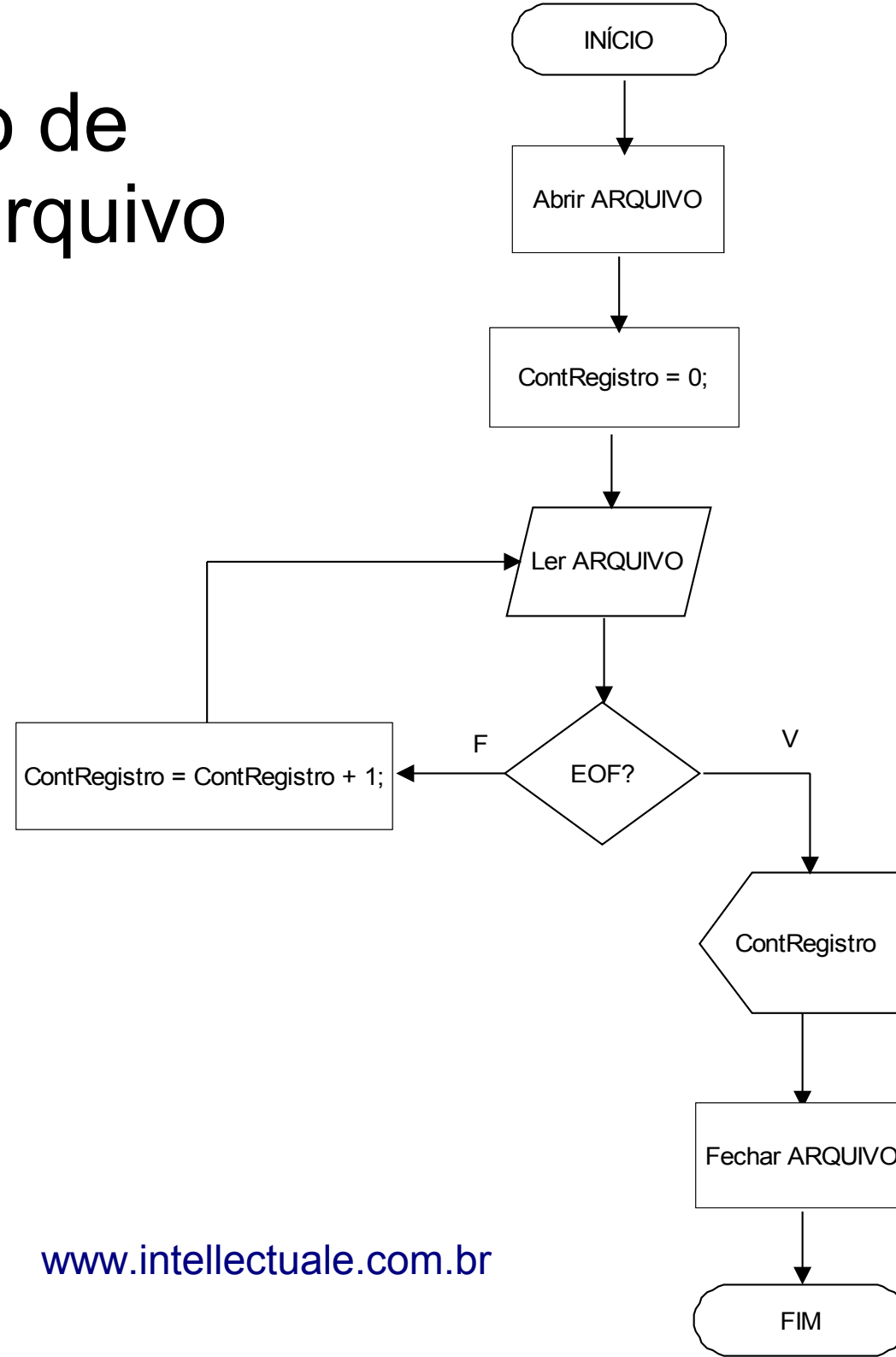
Registro 1

Registro 2

EOF (Final
do Arquivo)

Após a leitura de um registro, o ponteiro desloca-se para o seguinte, a fim de efetuar a próxima leitura.

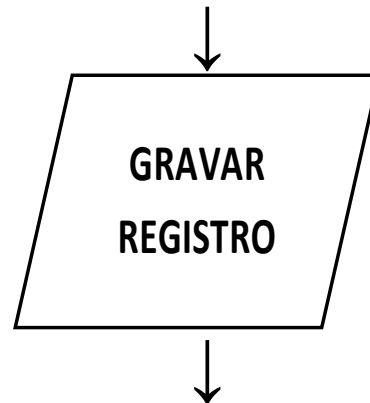
Processo de leitura de arquivo



Gravação de Arquivos

A gravação consiste na transferência de um registro da memória, para uma mídia de armazenamento digital.

Representação da gravação de arquivos:



Formas de acesso de arquivos

Os arquivos podem ser acessados de duas formas:

- 1 – Acesso sequencial;
- 2 - Acesso direto ou indexado;

Arquivo de Acesso Sequencial

No arquivo de acesso sequencial, os registros são gravados em formato sequencial, ou seja, um após o outro. Para acessar um determinado registro, é necessário passar por todos os anteriores.

Arquivo de Acesso direto

Neste tipo de arquivo, os registros são gravados com uma referência de localização, ou seja, utiliza-se um campo-chave. Para acessar um registro específico, torna-se necessária a indicação dessa chave.

Formas de acesso de arquivos

Os arquivos podem ser acessados de duas formas:

- 1 – Acesso sequencial;
- 2 - Acesso aleatório;

Linguagem C

Operações com arquivos

Operações com arquivo

Na linguagem C um arquivo é do tipo **FILE**, ou seja, é uma estrutura que é constituída de elementos do mesmo tipo organizados de forma sequencial.

O arquivo comunica-se com a memória RAM e com os meios de armazenamento por meio do sistema operacional.

Sintaxe:

FILE<*ponteiro>

A estrutura do arquivo está presente na biblioteca stdio.h

Leitura e escrita em arquivos

Para realizar operações de leitura ou de escrita em arquivo, primeiramente devemos abrir o arquivo e assim que realizarmos as operações de leitura ou escrita devemos fechar o arquivo.

Sintaxe de abertura de arquivo:

```
<ponteiro> = fopen("nome do arquivo", "tipo de abertura");
```

Sintaxe de fechamento de arquivo

```
fclose<ponteiro>;
```

Sendo que o ponteiro é a mesma variável ponteiro associada ao comando de abertura de arquivo.

Tipos de abertura de arquivos

Tipo de abertura	Descrição
r	Permissão de abertura somente para leitura. É necessário que o arquivo já esteja presente no disco.
w	<p>Permite abrir um arquivo texto para escrita (gravação). Este código cria o arquivo caso ele não exista, e caso o mesmo exista ele recria o arquivo novamente fazendo com que o conteúdo seja perdido.</p> <p>Portanto esse código deve ser usado com cautela e moderação.</p>
a	Permissão para abrir um arquivo texto para escrita(gravação), permite acrescentar novos dados ao final do arquivo. Caso não exista, ele será criado.

Criação de arquivo texto

```
#include <stdio.h>
int main(void)
{
    FILE *PontArq; // ponteiro para o arquivo

    PontArq = fopen("ARQUIVO1.TXT", "a"); //abrindo arquivo

    fclose(PontArq); // fechando arquivo

    printf("Arquivo criado com sucesso!");
    return(0);
}

/* Este programa cria o arquivo texto no mesmo diretório do
   executável do programa*/
```

Abrindo, gravando e fechando arquivo

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
FILE *PontArq; // cria variável ponteiro para o arquivo
```

```
char testeStr[15];
```

```
PontArq = fopen("ARQUIVO2.TXT", "w"); //abre arquivo
```

```
printf("\n\nEscreva uma string para testar gravacao de arquivo: ");
```

```
scanf("%s", testeStr);
```

```
fprintf(PontArq, "%s", testeStr); // grava a informação no arquivo
```

```
fclose(PontArq); // fecha arquivo
```

```
printf("Dados gravados com sucesso!");
```

```
return(0);
```

```
}
```

Manipulação de arquivo – Gravando vários strings

```
#include <stdio.h>
int main(void)
{
    FILE *PontArq;
    char RES, texto[51];
    RES = 'S';
    PontArq = fopen("TextoArquivo.txt", "a");
    while(RES == 'S' || RES == 's')
    {
        printf("\n\nDigite um texto qualquer\n\n");
        fflush(stdin);
        fgets(texto, 51, stdin);
        fputs(texto, PontArq);
        printf("\nDeseja continuar (S/N)? ");
        fflush(stdin);
        scanf("%c", &RES);
    }
    fclose(PontArq);
    return(0);
}
```


Leitura de arquivos

```
#include <stdio.h>
```

```
int main(void)
{
    FILE *PontArq;
    char TextoStr[51];
    PontArq = fopen("ARQUIVO2.TXT", "r");
    while(fgets(TextoStr, 50, PontArq) != NULL)
        printf("%s", TextoStr);
    fclose(PontArq);
    getch();
    return(0);
}
```

Exercício - Arquivos

Elabore um programa que leia, grave e exiba a apresentação de uma listagem dos nomes armazenados em arquivo texto.

O programa deve conter um menu para as rotinas de criação de arquivo, cadastro e exibição.

Programação Estruturada

Módulos

Técnicas de Programação Estruturada

A programação estruturada visa desenvolver programas confiáveis, flexíveis e de fácil manutenção utilizando as seguintes técnicas:

- 1 – Desenvolvimento de algoritmos por etapas;
- 2 – Número limitado de estruturas de controle de fluxo de dados;
- 3 – Refinamento do algoritmo em módulos.

Modularização

É a técnica que utiliza o conceito de refinamentos sucessivos para desenvolver algoritmos. Os refinamentos sucessivos, visam dividir um problema maior em partes menores, sendo que cada uma dessas partes tem uma tarefa bem definida.

Módulo é um conjunto de comandos que constitui uma parte de um algoritmo principal, e que possui uma tarefa bem definida.

Execução de módulos

Existe um módulo especial, denominado **módulo principal** que existe sempre em qualquer algoritmo (programa) que inicia automaticamente sua própria execução.

Ele é o único que inicia automaticamente, os outros módulos para serem executados precisam ser chamados (ativados) por algum módulo.

A execução começa sempre pelo módulo principal, sendo desviada para qualquer outro módulo que seja chamado pelo principal.

Cada módulo é executado até o seu fim, depois o fluxo de execução retorna ao ponto em que o módulo foi chamado e segue a execução do programa (algoritmo).

Módulos e subrotinas

Os módulos, também são chamados de subrotinas ou subprogramas.

Os módulos devem ser o mais independente possível entre si. Devido a isso eles podem possuir suas próprias variáveis locais e a dependência entre eles fica restrita ao cabeçalho (interface) do módulo.

Módulo: Retorno de valor

Retorno de valor é a saída por meio da qual o módulo dá uma resposta ao módulo que o chamou, ou seja que o ativou.

Essa resposta, pode ser o resultado de uma operação, ou uma sinalização de que a operação foi realizada com sucesso. De qualquer forma a resposta de retorno é um valor pertencente a um tipo de variável.

O retorno precisa ser recebido dentro do módulo chamador, ou seja, ativador, dentro do qual será tratado.

Comando de retorno:

```
retorne(valor);
```


Retorno de valores e diferença entre procedimento e função

Os módulos podem (ou não) retornar valores. Caso um módulo não retorne valores, dizemos que seu retorno é neutro.

Em algumas linguagens de programação (por exemplo: Pascal...) um módulo que não retorna valor é chamado de procedimento, enquanto que um módulo que retorna valor é chamado de função.

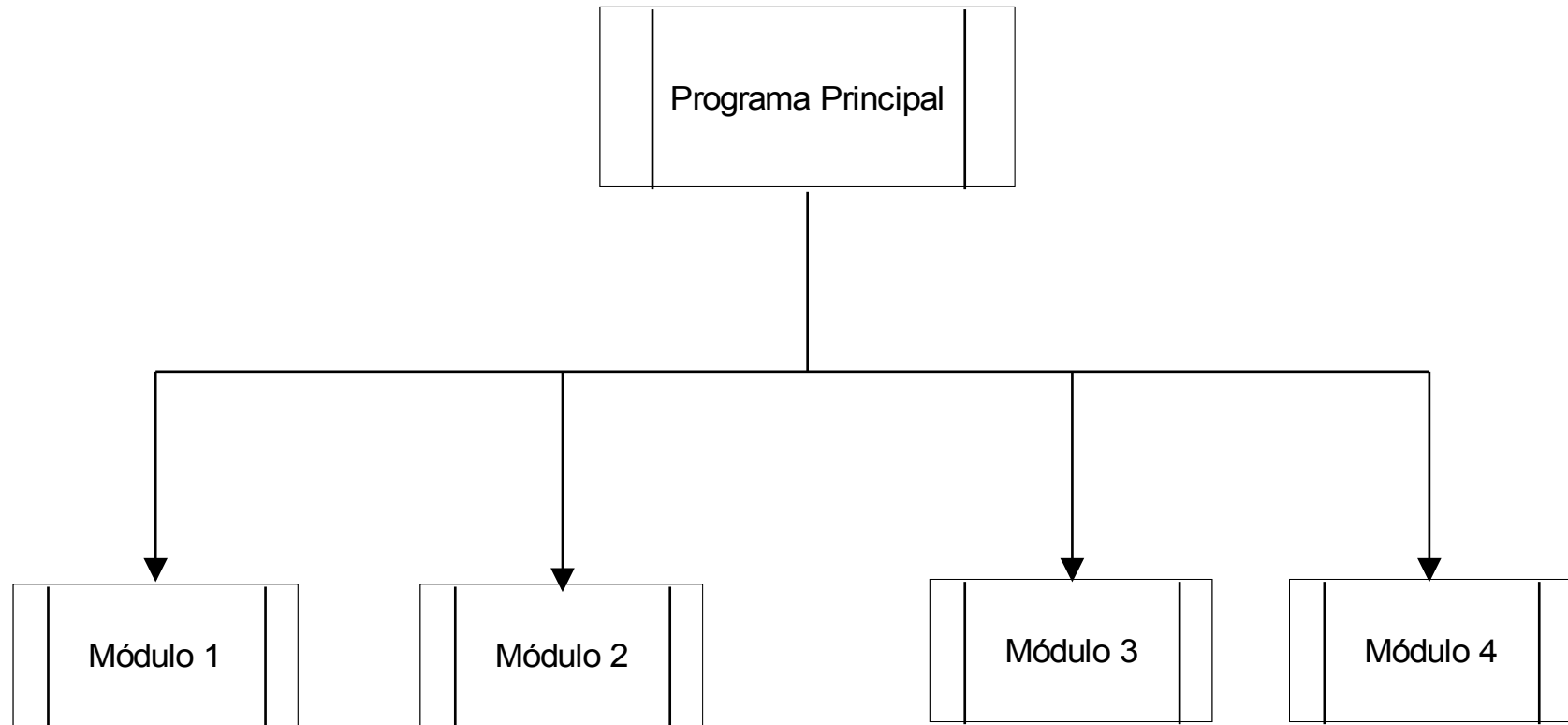
Na linguagem C, não existe esta distinção entre procedimento e função, logo independentemente de retornar ou não valores, em C todos os módulos são denominados funções.

Módulos: Procedimento e função

De forma geral, podemos dizer que procedimentos e funções são comandos utilizados para gerar módulos em um determinado algoritmo ou programa.

Ambos estão subordinados a um algoritmo principal denominado módulo principal ou programa principal.

Hierarquia dos módulos



Definição de Procedimento

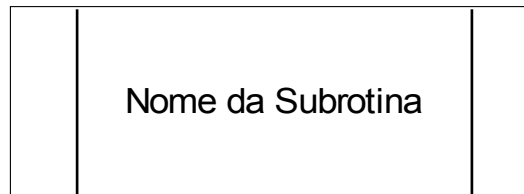
É um conjunto de comandos que realiza uma tarefa específica e é referenciado pelo programa principal por um nome.

Quando o programa, ou módulo principal referencia o procedimento, dizemos que está realizando a chamada do procedimento.

Ao ser chamado o procedimento é executado e após sua execução o programa principal reassume o controle do fluxo.

Simbologia de Procedimento

Diagrama de blocos



Para representar a um procedimento usamos o símbolo de subrotina (processo pré-definido).

Definição de Função

É um conjunto de comandos que realiza uma tarefa específica e é referenciado pelo programa principal por um nome, e retornando obrigatoriamente um valor no próprio nome da função.

Escopo de variáveis

- ✓ O chamado escopo de uma variável está relacionado com a abrangência da variável, ou seja, em quais partes do algoritmos a variável é “visível” e pode ser utilizada.
- ✓ **Variável local** só pode ser utilizada dentro do módulo onde foi declarada. É declarada no início do módulo a qual pertence.
- ✓ **Variável global** pode ser utilizada por todos os módulos de um programa. É declarada antes de todos os módulos.

Utilização de procedimento Pseudocódigo

Algoritmo Teste; //Algoritmo Principal

Var

N1, N2; // Variáveis globais

Procedimento Soma

Var

Resultado: Real; // Variável local ao procedimento

Início

Resultado \leftarrow N1 + N2;

Escreva("O resultado da soma é: ",Resultado);

Fim;

Início

Escreva("Digite os valores: ");

Receba(N1,N2);

Soma; //Chamada do procedimento

Fim.

Parâmetros

Os parâmetros permitem que os módulos se comuniquem a fim de realizar determinada tarefa.

Os parâmetros classificam-se em:

- Parâmetro formal: declarado com o módulo.
- Parâmetro real: substitui o parâmetro formal no programa principal.

Procedimento com passagem de parâmetros

Algoritmo Ex; //Algoritmo Principal

Procedimento Soma(N1,N2:REAL) //N1 e N2 parâmetros formais

Var

Resultado: Real;

Inicio

Resultado \leftarrow N1 + N2;

Escreva("O resultado da soma é: ",Resultado);

Fim;

Var

V1, V2;

Inicio

Escreva("Digite os valores: ");

Receba(V1,V2);

Soma(V1,V2); //chamada do procedimento

// e passagem dos parâmetros **REAIS** V1 e V2

Fim.

Procedimento - Exemplo Linguagem C

```
#include<stdio.h>
#include<stdlib.h>
Soma(float N1, float N2)  /* N1 e N2 parâmetros formais*/
{
    float resultado;
    resultado = N1 + N2;
    printf("Resultado = %8.2f\n",resultado);
}

int main(void)
{
    float V1, V2;
    printf("Digite o primeiro valor:");
    scanf("%f", &V1);
    printf("Digite o segundo valor:");
    scanf("%f", &V2);
    Soma(V1,V2);  /*chamada da função e passagem de parâmetros reais*/
    system("pause");
    return 0;
}
```

Procedimento – Exemplo em Pascal

```
Program ExProcedimento;
```

```
Procedure Multiplicar(A, B : integer);
```

```
Var
```

```
    Resultado : integer;
```

```
Begin
```

```
    Resultado := A * B;
```

```
    WriteLn('Resultado = ', Resultado);
```

```
End;
```

```
Var
```

```
    X, Y : integer;
```

```
Begin
```

```
    Write('Digite um valor: ');
```

```
    ReadLn(X);
```

```
    Write('Digite outro valor: ');
```

```
    ReadLn(Y);
```

```
    Multiplicar(X, Y);
```

```
    ReadLn();
```

```
End.
```

Passagem de parâmetros

- É o processo de substituição do parâmetro formal pelo parâmetro real.
- Dá-se de duas formas:
- Por valor
- Por referência

Passagem de parâmetros por valor

Nesse processo, uma cópia da variável é feita durante a ativação da subrotina (módulo), sendo os argumentos atribuídos aos parâmetros de entrada da subrotina.

Desta forma, as variáveis locais do módulo ativado ficam completamente independentes das variáveis do módulo chamador, ocorre um isolamento, ou seja mesmo que seus valores sejam alterados, essas alterações não são refletidas no módulo chamador.

Passagem de Parâmetros por referência

O endereço da variável é passado durante a ativação do módulo.

Neste processo, embora os identificadores das variáveis sejam diferentes, os valores manipulados no módulo ativado e do módulo ativador são exatamente a mesma variável. Deste modo, toda alteração feita é refletida no módulo ativador.

Função

Função é um conjunto de ações para realizar uma tarefa específica que é chamada pelo programa principal, assim como o procedimento, porém a diferença básica é que a função sempre retorna um valor.

Sintaxe:

função<nome>(lista de parâmetros): tipodevariavel;

Utilizando função Pseudocódigo

Algoritmo Principal;

Var

N1,N2,R:inteiro;

Função Mult(V1,V2:Inteiro):Inteiro;

Inicio

Mult =V1 * V2;

Fim;

Inicio

Escreva("Digite dois valores para multiplicar: ");

Receba(N1,N2);

R = Mult(N1,N2); // chamada da função mult

// o retorno da função é armazenado em R, então agora mostre

Escreva(R);

Fim.

Função – Exemplo em Pascal

```
PROGRAM EX_Funcao;  
USES CRT;  
VAR  
    N1,N2,R: INTEGER;
```

```
Function Mult(V1,V2:Integer):Integer;  
BEGIN  
    Mult:=V1*V2;  
END;
```

```
BEGIN // programa principal  
CLRSCR;  
    WRITE('Digite dois valores para serem multiplicados: ');  
    READLN(N1,N2);  
    R:=Mult(N1,N2);  
    Writeln('Resultado = ',R);  
    Readln;  
END.
```

Função – Linguagem C

```
#include<stdio.h>
#include<stdlib.h>
int Mult(int N1, int N2) //função que recebe N1,N2 e retorna um int
{
    int resultado;
    resultado = N1 * N2;
    return(resultado);    //retornando o valor para main
}
```

```
int main(void)
{
    int V1, V2, resultado;
    printf("Digite o primeiro valor:");
    scanf("%d", &V1);
    printf("Digite o segundo valor:");
    scanf("%d", &V2);
    resultado = Mult(V1,V2); //chama a função e recebe o retorno
    printf("Resultado = %d\n", resultado);
    system("pause");
    return 0;
}
```

Protótipo de função em C

O protótipo de uma função é basicamente, uma declaração da interface da função, ou seja, deve especificar:

- Tipo da função;
- Nome da função;
- Lista de parâmetros que a função necessita;

Exemplo:

```
int Mult(int N1, int N2)
```

Exercício - função

Elabore um programa que utilize uma função para calcular a média entre duas notas.

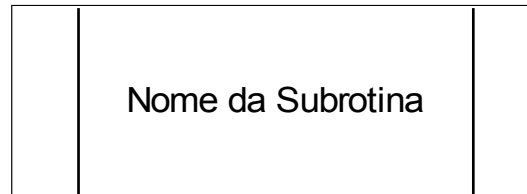
Cálculo de média usando função

```
#include<stdio.h>
#include<stdlib.h>
float Media(float N1, float N2); //protótipo da função
int main(void)
{
    float V1, V2, M;
    printf("Digite o primeiro valor:");
    scanf("%f", &V1);
    printf("Digite o segundo valor:");
    scanf("%f", &V2);
    M = Media(V1,V2); //chama a função e recebe o retorno
    printf("Media = %5.2f\n", M);
    system("pause");
    return 0;
}
float Media(float N1, float N2) //função recebe N1,N2 e retorna um float
{
    float resultado;
    resultado = (N1 + N2)/2;
    return(resultado); //retornando o valor para main
}
```

www.intellectuale.com.br

Simbologia da função

Diagrama de blocos



Para representar a função usamos o símbolo de subrotina (processo pré-definido).

Conceitos de Programação Orientada a Objetos

Programação Orientada a Objetos

Embora a programação orientada a objetos tenha sido mais divulgada a partir da década de 90, suas origens datam da década de 60.

Atualmente existem diversas linguagens de programação orientadas a objetos por exemplo C++, Delphi, Java, C# e outras.

O conceito de orientação a objetos, é suportado por quatro pilares mestres, que são eles: classe, objeto, atributo e método.

Conceitos de Programação Orientada a Objetos

Atributo

Características específicas de uma classe ou objeto.

Classe

Conjunto de objetos que possuem uma ou mais características comuns.

Estado

Situação do comportamento de um determinado objeto em um determinado momento.

Conceitos de Programação Orientada a Objetos

Encapsulamento

Definição de como implementar atributos e métodos de uma classe

Herança

Capacidade de uma classe filho (subclasse) herdar um ou mais classes (classe pai).

Instância de classe

É uma ocorrência específica de uma determinada classe, ou seja um objeto.

Conceitos de Programação Orientada a Objetos

Método

É a característica que possibilita alterar a funcionalidade de um determinado atributo de um objeto.

Objeto

Representação de um elemento do mundo real, Instância de uma classe que pode ser persistente ou transitória.

Poliformismo

Capacidade de interagir atributos de um objeto sem a necessidade de conhecer seu tipo. Capacidade que um objeto possui de mudar sua forma para outra forma.

Exemplo de classe com C++

```
class TAluno: // Definição da classe TAluno
{
    public:

        char Nome[50]; // atributo NOME
        float Notas[2]; // atributo NOTA
        float Media; // atributo MEDIA
        float CalcMed(void); // método CalcMed
} ALUNO; // nomeando o objeto ALUNO
```

Conceitos Básicos de Banco de Dados

Banco de dados

Banco de dados é um objeto capaz de armazenar informações complexas e estruturadas e recuperá-las rapidamente.

No entanto, para que isso seja possível, você precisa projetar e implementar corretamente seu banco de dados a fim de ele possa ser consultado com rapidez.

Onde podemos usar um banco de dados?

O banco de dados é usado para tomar decisões de negócios.
O interessante é encontrar respostas para perguntas como:

Qual foi o produto mais vendido na região Sudeste?
Em qual estado a maior venda ocorreu? Para qual cliente?
Qual é o mês de maior média de vendas do produto X?

Normalmente para dar essas respostas, somos obrigados a consultar dados de várias tabelas diferentes.

Banco de dados relacional

Os bancos de dados relacionais, tem por base relacionamentos entre seus dados.

As tabelas representam entidades do mundo real, e é dentro delas que os dados relacionados com as entidades ficam armazenados.

O ideal, é manter tabelas pequenas e facilmente gerenciáveis.

Tabelas, campos e registros

As informações de um banco de dados relacional são armazenadas em Tabelas.

Uma tabela é constituída por um conjunto de registros.

Os registros por sua vez, são um conjunto de campos.

Por exemplo, suponha uma tabela denominada "Clientes", onde seriam armazenadas informações sobre os diversos clientes. Podemos armazenar informações em diversos campos como: Nome, RG, CPF, Endereço, Data de Nascimento e outras.

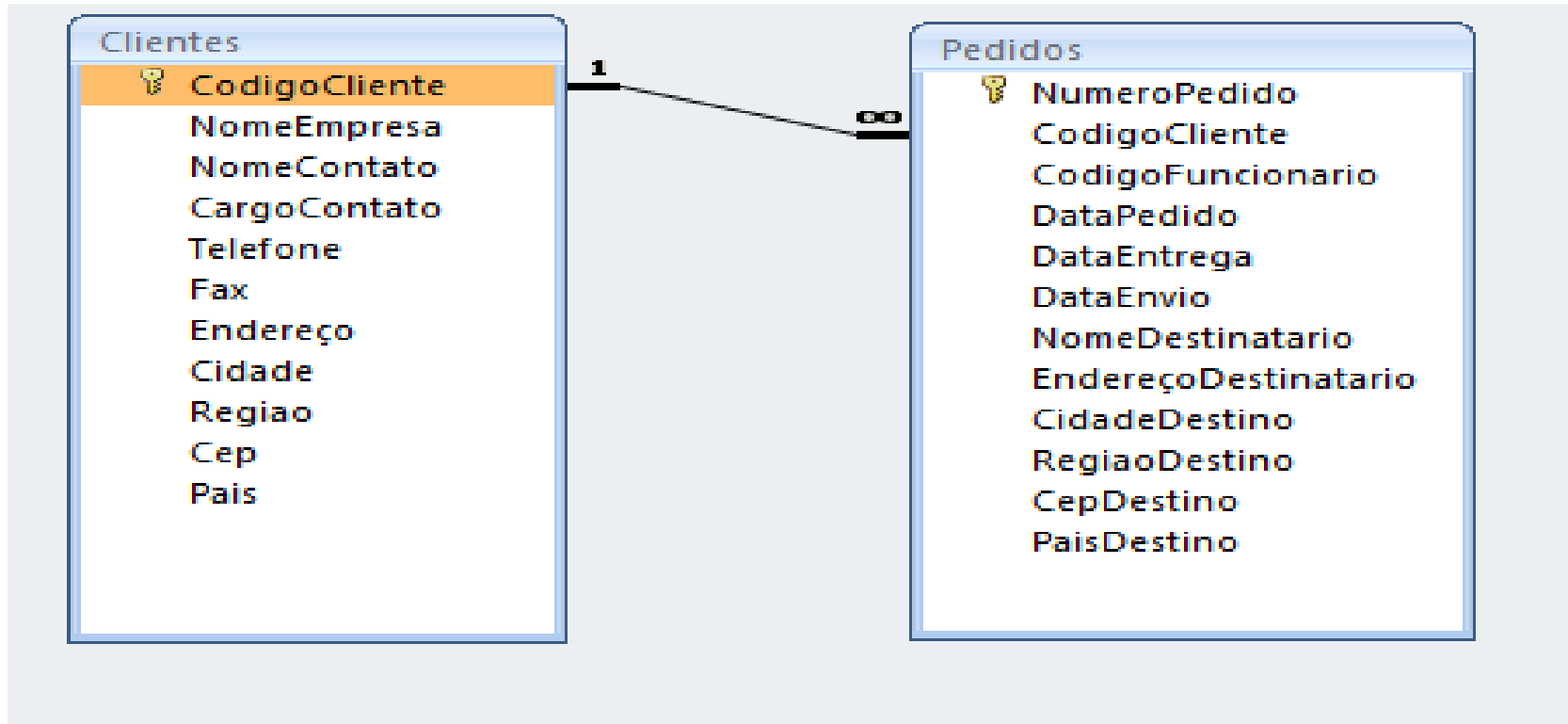
Um conjunto de campos relativo a um cliente forma o registro desse cliente.

Chave Primária

Chave primária é um campo que identifica de maneira única cada registro de uma tabela.

Exemplo prático: se o campo código da tabela clientes é a chave primária, isto significa que não podem ser cadastrados dois clientes com o mesmo código.

Relacionamento entre a tabela clientes e a tabela pedidos



Relacionamento um para vários entre Clientes e Pedidos.

Cada Cliente é cadastrado uma única vez na tabela de Clientes, portanto a tabela Clientes será o lado um do relacionamento. Porém um cliente pode fazer vários pedidos, sendo assim o mesmo código de cliente poderá aparecer várias vezes na tabela de Pedidos: tantas vezes quantos forem os pedidos que o cliente tiver feito.

Índice

Um índice é um mecanismo para agilizar buscas. Para isto é usada uma técnica chamada de indexação, que exige que os dados sejam mantidos em uma determinada ordem.

Um SGBD pode manter vários índices para uma tabela. Suponhamos a tabela de clientes. Você pode querer pesquisar por nome, CEP, cidade, país, por exemplo. O ideal então para acelerar essas pesquisas, é que exista um índice para cada campo que deseje pesquisar.

Linguagem SQL

Structured Query Language

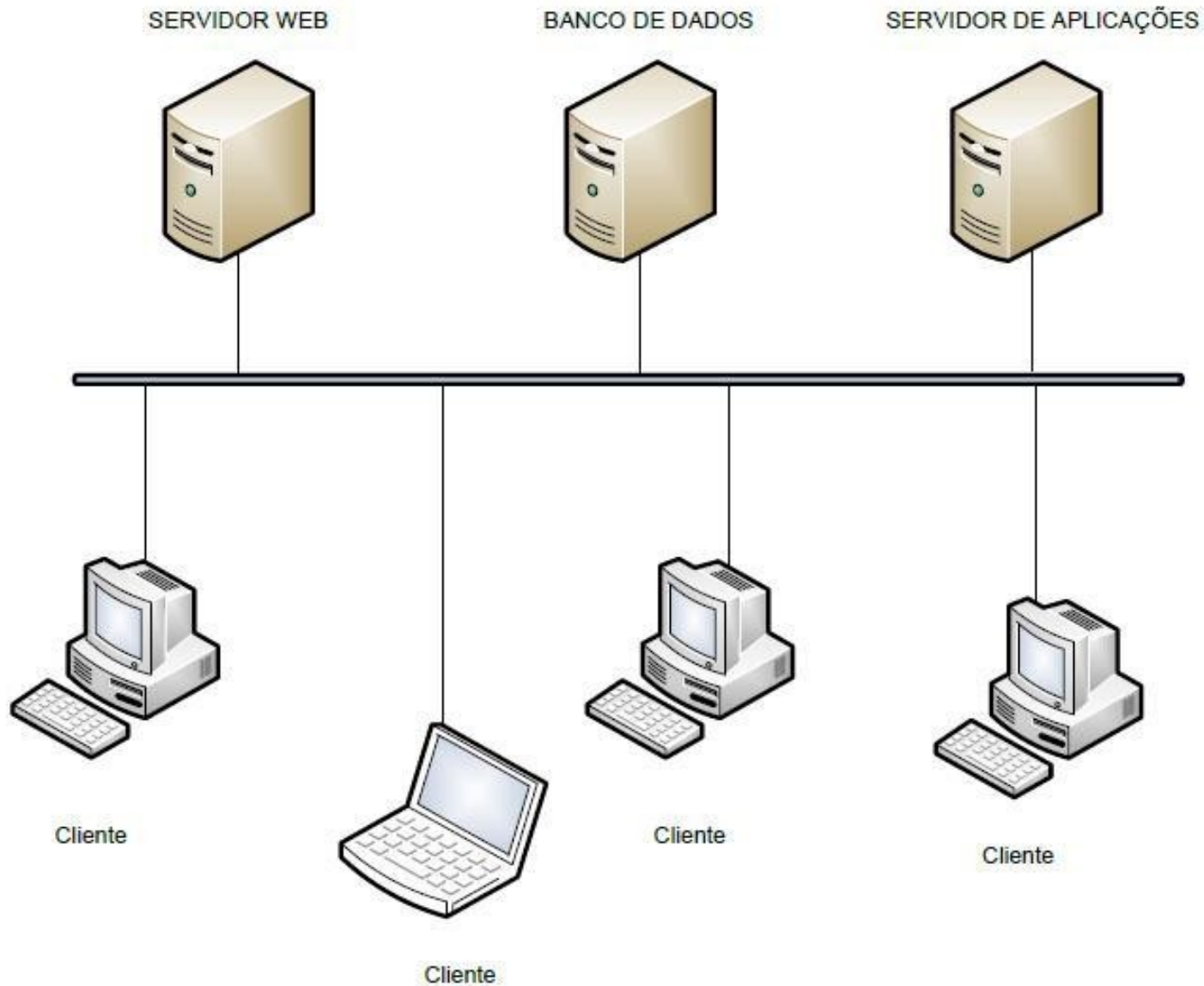
Linguagem padrão para banco de dados.

Possui três objetivos principais:

- Criar banco de dados e definir sua estrutura;
- Consultar o banco de dados a fim de obter dados necessários para responder as questões;
- Controlar a segurança do banco de dados;

Arquitetura Cliente/Servidor

4 camadas



Referências Bibliográficas

AVILLANO, ISRAEL DE CAMPOS : Algoritmos e Pascal – Manual de Apoio. Ed. Ciência Moderna, 2/2006

ZIVIANI, NIVIO :Projeto de Algoritmos com Implementação em Pascal e C. Ed. Thomson, 2/2004

GUIMARÃES/LAGES : Algoritmos e Estruturas de Dados. Ed. Livros Técnicos e Científicos

BOENTE, ALFREDO : Aprendendo a Programar em Linguagem C. Ed. Brasport

MANZANO, J.A.; Estudo Dirigido em Linguagem C. Editora Erica.

ASCENCIO, ANA FERNANDA G.. Lógica de Programação com Pascal. Ed. Makron Books. São Paulo, 1999