

# Lógica de Programação



## Sumário

<b>LÓGICA DE PROGRAMAÇÃO</b>	<b>8</b>
INTRODUÇÃO À LÓGICA DE PROGRAMAÇÃO	8
LÓGICA	8
SEQUÊNCIA LÓGICA	8
INSTRUÇÕES	8
ALGORITMO	9
PROGRAMAS	10
EXERCÍCIOS	10
DESENVOLVENDO ALGORITMO	12
PSEUDOCÓDIGO	12
REGRAS PARA CONSTRUÇÃO DO ALGORITMO	13
FASES	13
EXEMPLO DE ALGORITMO	14
TESTE DE MESA	15
EXERCÍCIOS	15
DIAGRAMA DE BLOCO	17
O QUE É UM DIAGRAMA DE BLOCO?	17
SIMBOLOGIA	17
EXERCÍCIOS	19
CONSTANTES, VARIÁVEIS E TIPOS DE DADOS	21
CONSTANTES	21
VARIÁVEIS	21
TIPOS DE VARIÁVEIS	22
DECLARAÇÃO DE VARIÁVEIS	22
EXERCÍCIOS	22
OPERADORES	25
OPERADORES ARITMÉTICOS	25
OPERADORES RELACIONAIS	26
OPERADORES LÓGICOS	27
EXERCÍCIOS	29
OPERAÇÕES LÓGICAS	29
EXERCÍCIOS	31
ESTRUTURAS DE DECISÃO E REPETIÇÃO	34
COMANDOS DE DECISÃO	35
SE ... ENTÃO	35
SE... ENTÃO... SENÃO	36
CASO SELECIONE / SELECT... CASE	38
EXERCÍCIOS	39
COMANDOS DE REPETIÇÃO	42
ENQUANTO X, PROCESSAR	42
ATÉ QUE X, PROCESSAR	43

PROCESSAR..., ENQUANTO X.....	44
PROCESSAR..., ATÉ QUE X.....	45
PARA DE... ATÉ... FAÇA.....	46
EXERCÍCIOS.....	47
ARQUIVOS DE DADOS .....	49
CONCEITOS BÁSICOS.....	49
ABERTURA DE ARQUIVOS.....	50
FECHAMENTO DE ARQUIVOS.....	51
LEITURA DE ARQUIVOS .....	51
MOVIMENTAÇÃO DE REGISTROS.....	52
GRAVAÇÃO DE ARQUIVOS.....	53
MACRO FLUXO.....	53
EXERCÍCIOS.....	54
<b>Linguagem C.....</b>	<b>57</b>
INTRODUÇÃO.....	57
PRIMEIROS PASSOS .....	58
EXERCÍCIOS.....	60
INTRODUÇÃO BÁSICA ÀS ENTRADAS E SAÍDAS .....	60
CARACTERES.....	60
STRINGS.....	62
COMANDO printf .....	63
COMANDO scanf .....	64
EXERCÍCIOS.....	65
COMENTÁRIOS.....	65
PALAVRAS RESERVADAS DO C .....	65
VARIÁVEIS, CONSTANTES, OPERADORES E EXPRESSÕES .....	66
NOMES DAS VARIÁVEIS .....	66
OS TIPOS DO C.....	66
Declaração e Inicialização de Variáveis.....	67
EXERCÍCIOS.....	70
Constantes.....	70
Constantes dos tipos básicos.....	70
Constantes hexadecimais e octais .....	70
Constantes strings.....	71
Constantes de barra invertida.....	71
Operadores Aritméticos e de Atribuição .....	72
EXERCÍCIOS.....	73
OPERADORES RELACIONAIS E LÓGICOS .....	74
Expressões .....	76
Conversão de tipos em expressões .....	76
Expressões que Podem ser Abreviadas.....	77
Encadeando expressões: o operador “;” .....	77

Tabela de Precedências do C.....	78
Modeladores (Casts) .....	79
<b>ESTRUTURAS DE CONTROLE DE FLUXO.....</b>	<b>79</b>
<b>TOMADA DE DECISÃO .....</b>	<b>80</b>
<b>COMANDO if.....</b>	<b>80</b>
<b>COMANDO else.....</b>	<b>81</b>
<b>COMANDO if-else-if .....</b>	<b>82</b>
<b>A expressão condicional.....</b>	<b>83</b>
<b>COMANDO ifs aninhados .....</b>	<b>84</b>
<b>COMANDO Operador “?” .....</b>	<b>85</b>
<b>EXERCÍCIOS.....</b>	<b>86</b>
<b>COMANDO switch .....</b>	<b>86</b>
<b>EXERCÍCIOS.....</b>	<b>88</b>
<b>LAÇOS DE REPETIÇÃO.....</b>	<b>88</b>
<b>COMANDO for .....</b>	<b>88</b>
<b>O loop infinito.....</b>	<b>90</b>
<b>O loop sem conteúdo .....</b>	<b>90</b>
<b>O loop para strings .....</b>	<b>91</b>
<b>EXERCÍCIOS.....</b>	<b>92</b>
<b>O comando While .....</b>	<b>92</b>
<b>O comando While .....</b>	<b>94</b>
<b>EXERCÍCIOS.....</b>	<b>95</b>
<b>COMANDO break.....</b>	<b>95</b>
<b>COMANDO continue .....</b>	<b>96</b>
<b>O Comando goto .....</b>	<b>97</b>
<b>EXERCÍCIOS.....</b>	<b>99</b>
<b>MATRIZES E STRINGS.....</b>	<b>100</b>
<b>Vetores.....</b>	<b>100</b>
<b>EXERCÍCIOS.....</b>	<b>101</b>
<b>Strings.....</b>	<b>102</b>
<b>COMANDO gets .....</b>	<b>103</b>
<b>COMANDO strcpy.....</b>	<b>103</b>
<b>COMANDO strcat.....</b>	<b>104</b>
<b>COMANDO strlen.....</b>	<b>105</b>
<b>COMANDO strcmp .....</b>	<b>105</b>
<b>EXERCÍCIOS.....</b>	<b>106</b>
<b>Matrizes.....</b>	<b>106</b>
<b>Matrizes bidimensionais .....</b>	<b>106</b>
<b>Matrizes de strings.....</b>	<b>107</b>
<b>Matrizes multidimensionais .....</b>	<b>108</b>
<b>Inicialização .....</b>	<b>108</b>
<b>Inicialização sem especificação de tamanho .....</b>	<b>109</b>

<b>EXERCÍCIOS.....</b>	<b>109</b>
<b>PONTEIROS .....</b>	<b>110</b>
<b>Como Funcionam os Ponteiros .....</b>	<b>110</b>
<b>Declarando e Utilizando Ponteiros.....</b>	<b>111</b>
<b>EXERCÍCIOS.....</b>	<b>114</b>
<b>Ponteiros e Vetores .....</b>	<b>115</b>
<b>Vetores como ponteiros .....</b>	<b>115</b>
<b>Ponteiros como vetores .....</b>	<b>117</b>
<b>Vetores de ponteiros .....</b>	<b>118</b>
<b>Inicializando Ponteiros.....</b>	<b>118</b>
<b>Ponteiros para Ponteiros .....</b>	<b>119</b>
<b>EXERCÍCIOS.....</b>	<b>119</b>
<b>Cuidados a Serem Tomados ao se Usar Ponteiros .....</b>	<b>120</b>
<b>FUNÇÕES.....</b>	<b>121</b>
<b>A Função .....</b>	<b>121</b>
<b>Argumentos .....</b>	<b>122</b>
<b>O Comando return.....</b>	<b>123</b>
<b>EXERCÍCIOS.....</b>	<b>126</b>
<b>Protótipos de Funções.....</b>	<b>127</b>
<b>Arquivos-Cabeçalhos.....</b>	<b>128</b>
<b>EXERCÍCIOS.....</b>	<b>129</b>
<b>Escopo de Variáveis .....</b>	<b>130</b>
<b>Variáveis locais.....</b>	<b>130</b>
<b>Parâmetros formais .....</b>	<b>131</b>
<b>Variáveis globais .....</b>	<b>131</b>
<b>ENTRADAS E SAÍDAS PADRONIZADAS.....</b>	<b>132</b>
<b>Introdução.....</b>	<b>132</b>
<b>Lendo e Escrevendo Caracteres .....</b>	<b>133</b>
<b>Comando getch e getche.....</b>	<b>133</b>
<b>Comando putchar.....</b>	<b>134</b>
<b>Lendo e Escrevendo Strings .....</b>	<b>134</b>
<b>Comando gets.....</b>	<b>134</b>
<b>Comando puts.....</b>	<b>135</b>
<b>ENTRADA E SAÍDA FORMATADA.....</b>	<b>135</b>
<b>Comando printf.....</b>	<b>135</b>
<b>Comando scanf.....</b>	<b>137</b>
<b>Comando sprintf e sscanf .....</b>	<b>138</b>
<b>EXERCÍCIOS.....</b>	<b>139</b>
<b>Abrindo e Fechando um Arquivo .....</b>	<b>139</b>
<b>Comando fopen .....</b>	<b>139</b>
<b>Comando exit .....</b>	<b>141</b>
<b>Comando fclose.....</b>	<b>141</b>

Lendo e Escrevendo Caracteres em Arquivos.....	142
Comando putc.....	142
Comando getc.....	143
Comando feof ou EOF.....	143
EXERCÍCIOS.....	145
Outros Comandos de Acesso a Arquivos.....	145
Comando fgetc.....	146
Comando fputc .....	147
Comando ferror e perror.....	147
Comando fread .....	148
Comando fwrite .....	148
Comando fseek.....	150
Comando rewind .....	150
Comando remove.....	150
Fluxos Padrão .....	152
Comando fprintf.....	152
Comando fscanf .....	152
EXERCÍCIOS.....	153
TIPOS DE DADOS DEFINIDOS PELO USUÁRIO .....	153
Estruturas - Primeira parte.....	153
Criar a estrutura.....	154
Usar a estrutura .....	155
Matrizes de estruturas.....	156
EXERCÍCIOS.....	156
Estruturas - Segunda parte.....	157
Atribuindo .....	157
Passando para funções .....	157
EXERCÍCIOS.....	158
Declaração Union .....	159
Enumerações .....	160
O Comando sizeof.....	162
<b>Introdução a Banco de Dados MYSQL.....</b>	<b>164</b>
MODELAGEM DE DADOS .....	164
MODELO CONCEITUAL .....	164
ENTIDADE/RELACIONAMENTOS.....	164
MODELO ENTIDADE RELACIONAMENTO (MER).....	165
ENTIDADE .....	166
ATRIBUTO.....	166
Tipos de atributos .....	166
RELACIONAMENTO.....	167
DIAGRAMA ENTIDADE-RELACIONAMENTO (DER).....	168
MODELO LÓGICO .....	169

<b>SQL 170</b>	
<b>DML – LINGUAGEM DE MANIPULAÇÃO DE DADOS .....</b>	<b>170</b>
<b>DDL – LINGUAGEM DE DEFINIÇÃO DE DADOS .....</b>	<b>171</b>
<b>DCL – LINGUAGEM DE CONTROLE DE DADOS .....</b>	<b>172</b>
<b>DTL – LINGUAGEM DE TRANSAÇÃO DE DADOS .....</b>	<b>173</b>
<b>DQL – LINGUAGEM DE CONSULTA DE DADOS .....</b>	<b>173</b>
<b>MYSQL.....</b>	<b>174</b>
<b>CRIANDO, INSERINDO E EXIBINDO – PHP+MYSQL .....</b>	<b>174</b>
<b>O que é um banco de dados?.....</b>	<b>175</b>
<b>Sistema de banco de Dados .....</b>	<b>175</b>
<b>PHPMYADMIN.....</b>	<b>175</b>
<b>CRIANDO UM BANCO DE DADOS UTILIZANDO O PHPMYADMIN .....</b>	<b>176</b>
<b>C# Visual Studio .....</b>	<b>182</b>
<b>METODOLOGIA X MÉTODOS .....</b>	<b>182</b>
<b>METODOLOGIA .....</b>	<b>182</b>
<b>METODO .....</b>	<b>186</b>
<b>INTRODUÇÃO .....</b>	<b>186</b>
<b>A LINGUAGEM.....</b>	<b>186</b>
<b>O .NET FRAMEWORK.....</b>	<b>186</b>
<b>DEFINIÇÃO DE VARIÁVEIS .....</b>	<b>187</b>
<b>OPERADORES ARITMÉTICOS.....</b>	<b>188</b>
<b>OPERADORES ARITMÉTICOS DE ATRIBUIÇÃO REDUZIDA .....</b>	<b>189</b>
<b>Operadores incrementais e decrementais:.....</b>	<b>189</b>
<b>INCREMENTAL (++).....</b>	<b>190</b>
<b>PRE – INCREMENTAL.....</b>	<b>190</b>
<b>PÓS-INCREMENTAL.....</b>	<b>190</b>
<b>DECREMENTAL (-) .....</b>	<b>191</b>
<b>PRÉ – DECREMENTAL .....</b>	<b>191</b>
<b>PÓS – DECREMENTAL.....</b>	<b>191</b>
<b>OPERADORES RELACIONAIS.....</b>	<b>192</b>
<b>OPERADORES LÓGICOS.....</b>	<b>192</b>
<b>COMANDOS CONDICIONAIS .....</b>	<b>194</b>
<b>COMANDOS DE DECISÃO.....</b>	<b>194</b>
<b>REPETIÇÃO FOR .....</b>	<b>200</b>
<b>BREAK E CONTINUE .....</b>	<b>201</b>
<b>BREAK.....</b>	<b>201</b>
<b>CONTINUE .....</b>	<b>202</b>
<b>NAMESPACE.....</b>	<b>203</b>
<b>CLASSES .....</b>	<b>204</b>
<b>FORMS.....</b>	<b>205</b>
<b>BUTTON .....</b>	<b>209</b>
<b>LABEL.....</b>	<b>219</b>

<b>COMBOBOX.....</b>	<b>221</b>
<b>LISTBOX .....</b>	<b>225</b>
<b>TOLLTIP .....</b>	<b>229</b>
<b>GROUPBOX .....</b>	<b>232</b>
<b>CRIANDO MINI NAVEGADOR.....</b>	<b>235</b>
<b>O QUE É FRAMEWORK?.....</b>	<b>240</b>



# LÓGICA DE PROGRAMAÇÃO

## INTRODUÇÃO À LÓGICA DE PROGRAMAÇÃO

### LÓGICA

A lógica de programação é necessária para pessoas que desejam trabalhar com desenvolvimento de sistemas e programas, ela permite definir a sequência lógica para o desenvolvimento. Então o que é lógica?

**“Lógica de programação é a técnica de encadear pensamentos para atingir determinado objetivo.”**

### SEQUÊNCIA LÓGICA

Estes pensamentos podem ser descritos como uma sequência de instruções, que devem ser seguidas para se cumprir uma determinada tarefa.

**“Sequência Lógica são passos executados até atingir um objetivo ou solução de um problema.”**

### INSTRUÇÕES

Na linguagem comum, entende-se por *instruções* “um conjunto de regras ou normas definidas para a realização ou emprego de algo”.

Em informática, porém, instrução é a informação que indica a um computador uma ação elementar a executar. Convém ressaltar que uma ordem isolada não permite realizar o processo completo, para isso é necessário um conjunto de instruções colocadas em ordem sequencial lógica.

Por exemplo, se quisermos fazer uma omelete de batatas, precisaremos colocar em prática uma série de instruções: descascar as batatas, bater os ovos, fritar as batatas, etc. É evidente que essas instruções têm que ser executadas em uma ordem adequada – não se pode descascar as batatas depois de fritá-las.

Dessa maneira, uma instrução tomada em separado não tem muito sentido; para obtermos o resultado, precisamos colocar em prática o conjunto de todas as instruções, na ordem correta.

**“Instruções são um conjunto de regras ou normas definidas para a realização ou emprego de algo. Em informática, é o que indica a um computador uma ação elementar a executar”.**

## ALGORITMO

Um algoritmo é formalmente uma sequência finita de passos que levam a execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma sequência de instruções que dão cabo de uma meta específica. Estas tarefas não podem ser redundantes nem subjetivas na sua definição, devem ser claras e precisas.

Como exemplos de algoritmos podemos citar os algoritmos das operações básicas (adição, multiplicação, divisão e subtração) de números reais decimais. Outros exemplos seriam os manuais de aparelhos eletrônicos, como um videocassete, que explicam passo-a-passo como, por exemplo, gravar um evento.

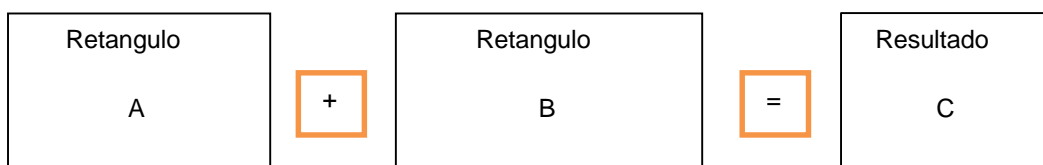
Até mesmo as coisas mais simples, podem ser descritas por sequências lógicas. Por exemplo:

“Chupar uma bala”.

- Pegar a bala
- Retirar o papel
- Chupar a bala
- Jogar o papel no lixo

“Somar dois números quaisquer”.

- Escreva o primeiro número no retângulo A
- Escreva o segundo número no retângulo B
- Some o número do retângulo A com número do retângulo B e coloque o resultado no retângulo C.





[illegible]

This image shows a full page of white paper with horizontal blue or grey ruling lines. The lines are evenly spaced and run across the width of the page, providing a template for handwriting practice. There are no margins, text, or other markings on the paper.



O algoritmo deve ser fácil de interpretar e fácil de codificar. Ou seja, ele deve ser o intermediário entre a linguagem falada e a linguagem de programação.

## REGRAS PARA CONSTRUÇÃO DO ALGORITMO

Para escrever um algoritmo precisamos descrever a sequência de instruções, de maneira simples e objetiva. Para isso utilizaremos algumas técnicas:

- Usar somente um verbo por frase
- Imaginar que você está desenvolvendo um algoritmo para pessoas que não trabalham com informática
- Usar frases curtas e simples
- Ser objetivo
- Procurar usar palavras que não tenham sentido dúbio

## FASES

No capítulo anterior vimos que ALGORITMO é uma sequência lógica de instruções que podem ser executadas.

É importante ressaltar que qualquer tarefa que siga determinado padrão pode ser descrita por um algoritmo, como por exemplo:

***COMO FAZER ARROZ DOCE***

Ou então

***CALCULAR O SALDO FINANCEIRO DE UM ESTOQUE***

Entretanto ao montar um algoritmo, precisamos primeiro dividir o problema apresentado em três fases fundamentais.



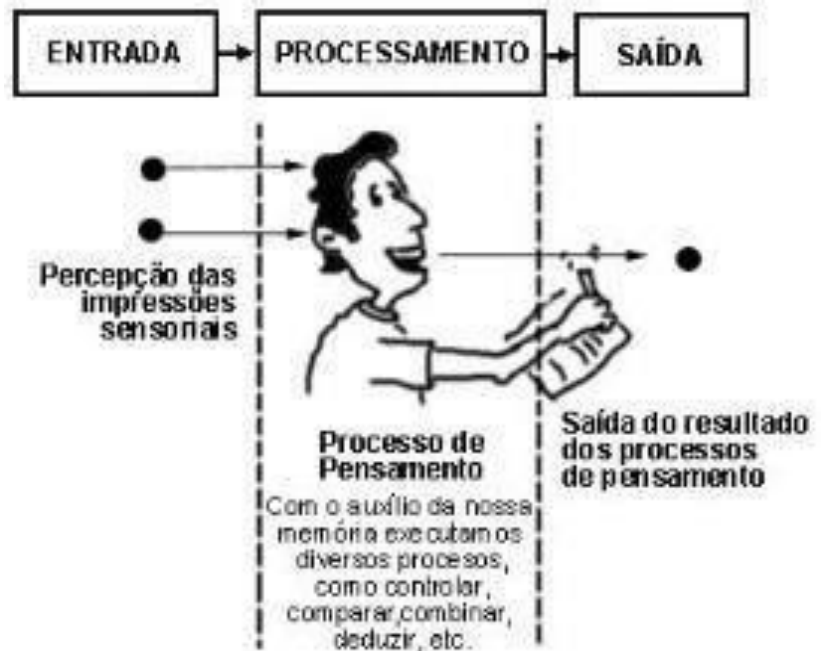
Onde temos:

**ENTRADA:** São os dados de entrada do algoritmo

**PROCESSAMENTO:** São os procedimentos utilizados para chegar ao resultado final

**SAÍDA:** São os dados já processados

Analogia com o homem



## EXEMPLO DE ALGORITMO

Imagine o seguinte problema: Calcular a média final dos alunos da 3ª Série.

Os alunos realizarão quatro provas: P1, P2, P3 e P4.

Onde:

$$\text{Média Final} = \frac{P1 + P2 + P3 + P4}{4}$$

Para montar o algoritmo proposto, faremos três perguntas:

a) Quais são os dados de entrada? R: Os dados de entrada são P1, P2, P3 e P4

b) Qual será o processamento a ser utilizado?

R: O procedimento será somar todos os dados de entrada e dividi-los por 4 (quatro); **(P1 + P2 + P3 + P4)/4**

c) Quais serão os dados de saída?

R: O dado de saída será a média final

### Algoritmo

Receba a nota da prova1

Receba a nota de prova2

Receba a nota de prova3

Receba a nota da prova4

Some todas as notas e divida o resultado por 4

Mostre o resultado da divisão

## TESTE DE MESA

Após desenvolver um algoritmo ele deverá sempre ser testado. Este teste é chamado de **TESTE DE MESA**, que significa, seguir as instruções do algoritmo de maneira precisa para verificar se o procedimento utilizado está correto ou não.

Veja o exemplo:

Nota da Prova 1

Nota da Prova 2

Nota da Prova 3

Nota da Prova 4

Utilize a tabela ao lado:

P1	P2	P3	P4	Média

## EXERCÍCIOS

1) Identifique os dados de entrada, processamento e saída no algoritmo abaixo.

- Receba código da peça
- Receba valor da peça
- Receba Quantidade de peças



- Calcule o valor total da peça (Quantidade \* Valor da peça)
- Mostre o código da peça e seu valor total.

**2)** Faça um algoritmo para “Calcular o estoque médio de uma peça”, sendo que.

$$\text{ESTOQUE M\u00c9DIO} = (\text{QUANTIDADE M\u00cdNIMA} + \text{QUANTIDADE M\u00c1XIMA}) / 2$$

**3)** Teste o algoritmo anterior com dados definidos por voc\u00ea.

# DIAGRAMA DE BLOCO

## O QUE É UM DIAGRAMA DE BLOCO?





O diagrama de blocos é uma forma padronizada e eficaz para representar os passos lógicos de um determinado processamento.

Com o diagrama podemos definir uma sequência de símbolos, com significado bem definido, portanto, sua principal função é a de facilitar a visualização dos passos de um processamento.

## SIMBOLOGIA

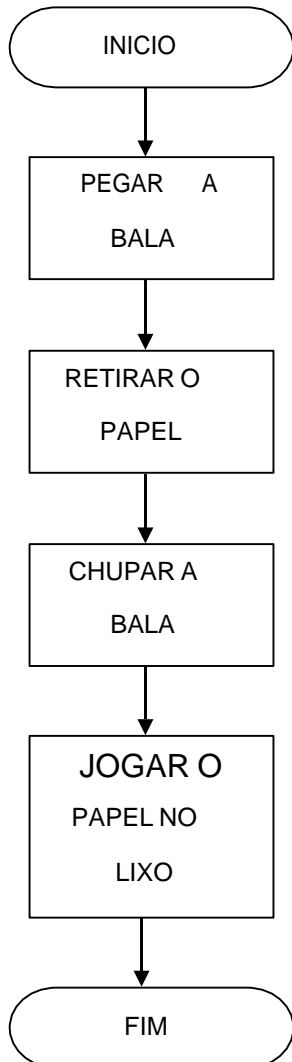
Existem diversos símbolos em um diagrama de bloco. No decorrer do curso apresentaremos os mais utilizados.

Veja no quadro abaixo alguns dos símbolos que iremos utilizar:

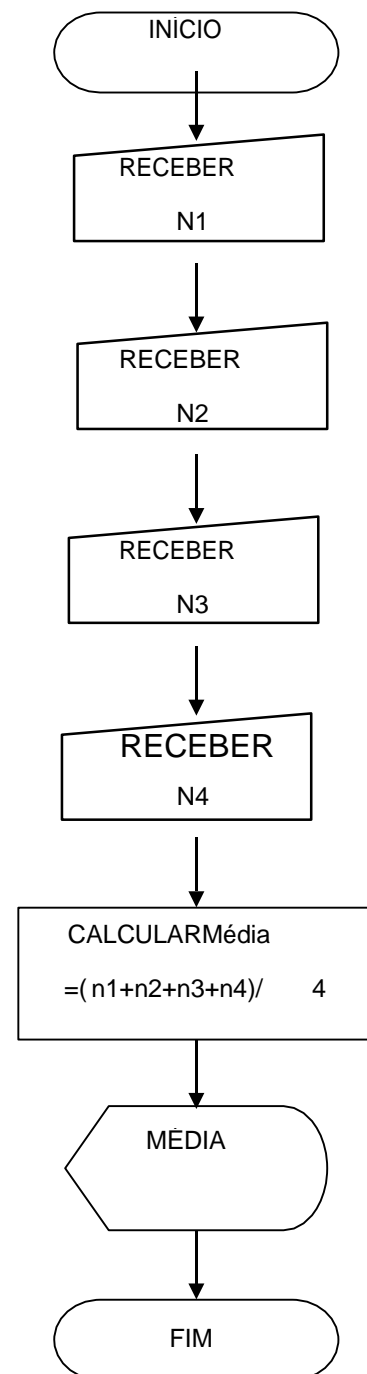
Símbolo	Função
<b>TERMINAL</b> 	Indica o INÍCIO ou FIM de um processamento Exemplo: Início do algoritmo
<b>PROCESSAMENTO</b> 	Processamento em geral Exemplo: Calculo de dois números
<b>ENTRADA DE DADO MANUAL</b> 	Indica entrada de dados através do Teclado Exemplo: Digite a nota da prova 1
<b>EXIBIR</b>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

Dentro do símbolo sempre terá algo escrito, pois somente os símbolos não nos dizem nada. Veja no exemplo a seguir:

Exemplos de Diagrama de Bloco  
“CHUPAR UMA BALA”



“CALCULAR A MÉDIA DE 4 NOTAS



Veja que no exemplo da bala seguimos uma sequência lógica somente com informações diretas, já no segundo exemplo da média utilizamos cálculo e exibimos o resultado do mesmo.

# EXERCÍCIOS

1) Construa um diagrama de blocos que:

- Leia a cotação do dólar
- Leia um valor em dólares
- Converta esse valor para Real
- Mostre o resultado

2) Desenvolva um diagrama que:

- Leia 4 (quatro) números
- Calcule o quadrado para cada um
- Somem todos e
- Mostre o resultado

3) Construa um algoritmo para pagamento de comissão de vendedores de peças, levando-se em consideração que sua comissão será de 5% do total da venda e que você tem os seguintes dados:

- Identificação do vendedor
- Código da peça
- Preço unitário da peça
- Quantidade vendida

E depois construa o diagrama de blocos do algoritmo desenvolvido, e por fim faça um teste de mesa.

# CONSTANTES, VARIÁVEIS E TIPOS DE DADOS

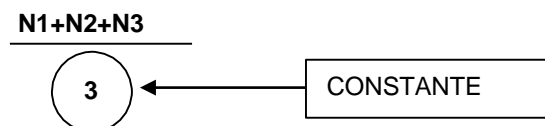
Variáveis e constantes são os elementos básicos que um programa manipula. Uma variável é um espaço reservado na memória do computador para armazenar um tipo de dado determinado.

Variáveis devem receber nomes para poderem ser referenciadas e modificadas quando necessário. Um programa deve conter declarações que especificam de que tipo são as variáveis que ele utilizará e às vezes um valor inicial. Tipos podem ser por exemplo: inteiros, reais, caracteres, etc. As expressões combinam variáveis e constantes para calcular novos valores.

## CONSTANTES

Constante é um determinado valor fixo que não se modifica ao longo do tempo, durante a execução de um programa. Conforme o seu tipo, a constante é classificada como sendo numérica, lógica e literal.

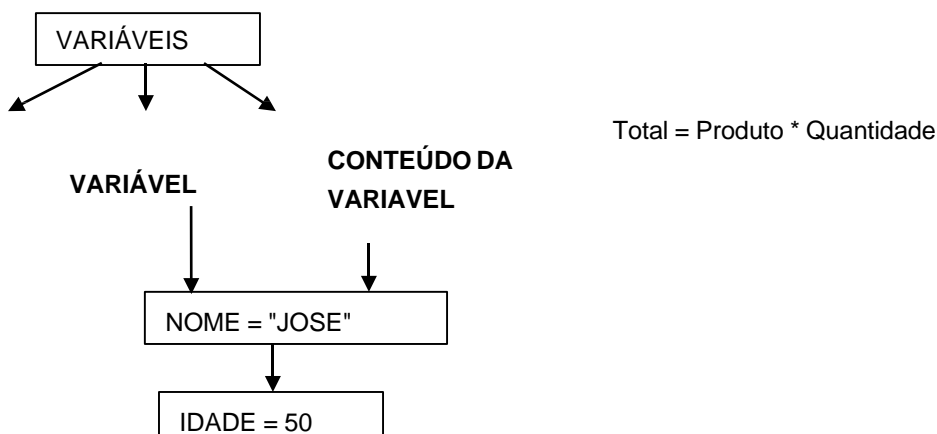
Exemplo de constantes:



## VARIÁVEIS

Variável é a representação simbólica dos elementos de um certo conjunto. Cada variável corresponde a uma posição de memória, cujo conteúdo pode se alterar ao longo do tempo durante a execução de um programa. Embora uma variável possa assumir diferentes valores, ela só pode armazenar um valor a cada instante

Exemplos de variáveis:



## TIPOS DE VARIÁVEIS

As variáveis e as constantes podem ser basicamente de quatro tipos:

Numéricas, caracteres, Alfanuméricas ou lógicas.

<b>Numéricas</b>	Específicas para armazenamento de números, que posteriormente poderão ser utilizados para cálculos. Podem ser ainda classificadas como Inteiras ou Reais. As variáveis do tipo inteiro são para armazenamento de números inteiros e as Reais são para o armazenamento de números que possuam casas decimais.
<b>Caracteres</b>	Específicas para armazenamento de conjunto de caracteres que não contenham números (literais). Ex: nomes.
<b>Alfanuméricas</b>	Específicas para dados que contenham letras e/ou números. Pode em determinados momentos conter somente dados numéricos ou somente literais. Se usado somente para armazenamento de números, não poderá ser utilizada para operações matemáticas.
<b>Lógicas</b>	Armazenam somente dados lógicos que podem ser Verdadeiro ou Falso.

## DECLARAÇÃO DE VARIÁVEIS

As variáveis só podem armazenar valores de um mesmo tipo, de maneira que também são classificadas como sendo numéricas, lógicas e literais.

## EXERCÍCIOS

1) O que é uma constante? Dê dois exemplos.

---



---



---



---

2) O que é uma variável? Dê dois exemplos.

---



---



---



---

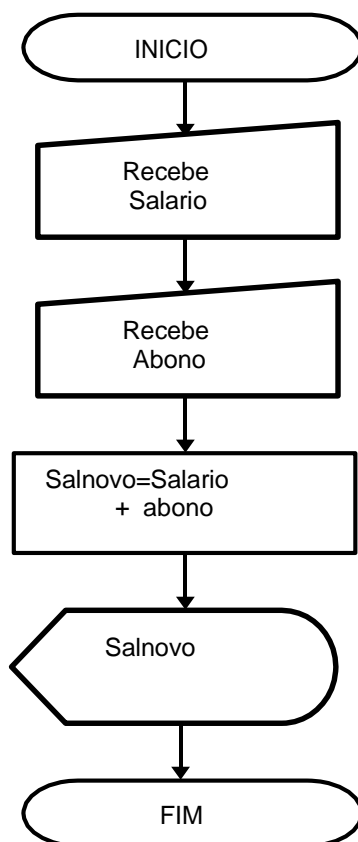


---



---

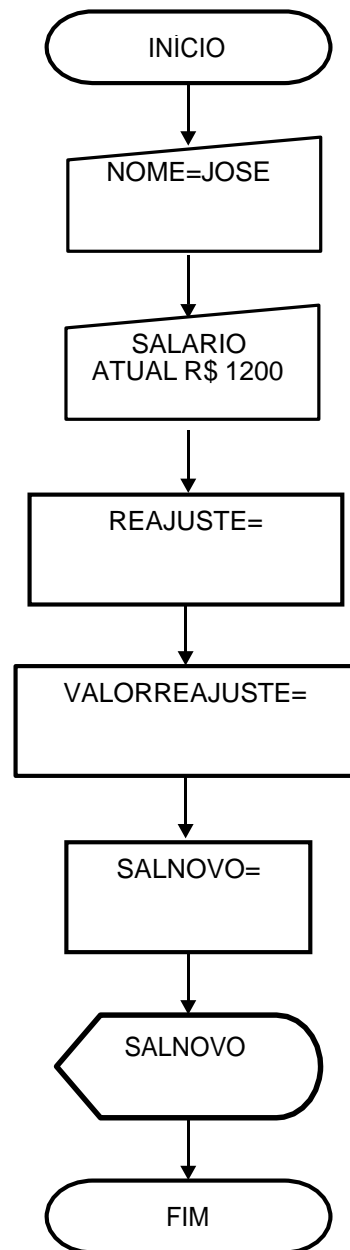
3) Faça um teste de mesa no diagrama de bloco abaixo e preencha a tabela ao lado com os dados do teste:



Salário	Abono	Salnovo
600,00	60,00	
350,00		



4) Sabendo-se que José tem direito a 15% de reajuste de salário, complete o diagrama abaixo:



# OPERADORES

Os operadores são meios pelo qual incrementamos, decrementamos, comparamos e avaliamos dados dentro do computador. Temos três tipos de operadores:

- Operadores Aritméticos
- Operadores Relacionais
- Operadores Lógicos

## OPERADORES ARITMÉTICOS

Os operadores aritméticos são os utilizados para obter resultados numéricos. Além da adição, subtração, multiplicação e divisão, podem utilizar também o operador para exponenciação.

Os símbolos para os operadores aritméticos são:

OPERAÇÃO	SÍMBOLO
Adição	+
Subtração	—
Multiplicação	*
Divisão	/
Exponenciação	**

### Hierarquia das Operações Aritméticas:

- 1º ( ) Parênteses
- 2º Exponenciação
- 3º Multiplicação, divisão (o que aparecer primeiro)
- 4º + ou — (o que aparecer primeiro)

### Exemplo:

**TOTAL = PRECO \* QUANTIDADE**

$$1 + 7 * 2 ** 2 - 1 = 28$$

$$3 * (1 - 2) + 4 * 2 = 5$$

## OPERADORES RELACIONAIS

Os operadores relacionais são utilizados para comparar **String** de caracteres e números. Os valores a serem comparados podem ser caracteres ou variáveis.

Estes operadores sempre retornam valores lógicos (verdadeiro ou falso/ True ou False)

Para estabelecer prioridades no que diz respeito a qual operação executar primeiro, utilize os parênteses.

Os operadores relacionais são:

Descrição	Símbolo
Igual a	=
Diferente de	<> ou #
Maior que	>
Menor que	<
Maior ou igual a	>=
Menor ou igual a	<=

Exemplo:

Tendo duas variáveis A = 5 e B = 3

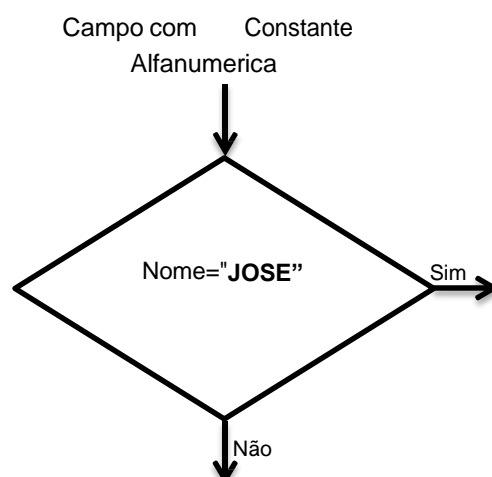
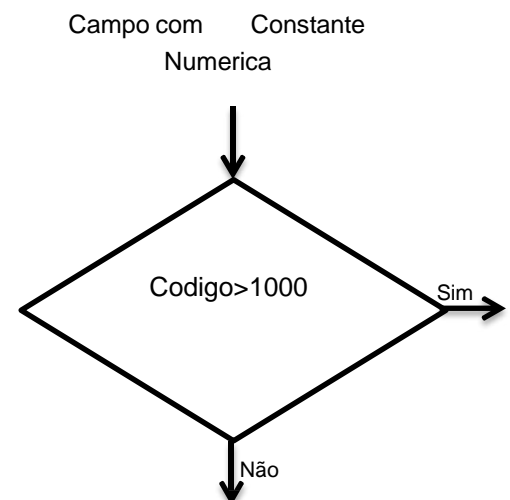
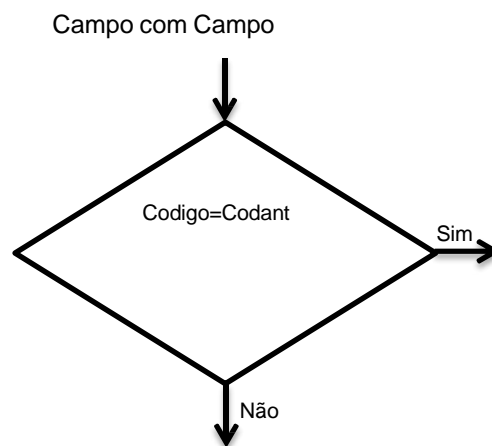
Os resultados das expressões seriam:

Expressão	Resultado
-----------	-----------

Expressão	Resultado
A = B	Falso
A <> B	Verdadeiro
A > B	Verdadeiro
A < B	Falso
A >= B	Verdadeiro
A <= B	Falso

Símbolo Utilizado para comparação entre expressões

## OPERADORES LÓGICOS



Os operadores lógicos servem para combinar resultados de expressões, retornando se o resultado final é verdadeiro ou falso.

Os operadores lógicos são:

E	AND
OU	OR
NÃO	NOT

**E / AND** - Uma expressão AND (E) é verdadeira se todas as condições forem verdadeiras

**OR/OU** - Uma expressão OR (OU) é verdadeira se pelo menos uma condição for verdadeira

**NOT** - Uma expressão NOT (NÃO) inverte o valor da expressão ou condição, se verdadeira inverte para falsa e vice-versa.

A tabela abaixo mostra todos os valores possíveis criados pelos três operadores lógicos

**(AND, OR e NOT)**

1º Valor	Operador	2º Valor	Resultado
T	AND	T	T
T	AND	F	F
F	AND	T	F
F	AND	F	F
T	OR	T	T
T	OR	F	T
F	OR	T	T
F	OR	F	F
T	NOT		F
F	NOT		T

Exemplos:

Suponha que temos três variáveis A = 5, B = 8 e C = 1

Os resultados das expressões seriam:

Expressões			Resultado
A = B	AND	B > C	Falso
A <> B	OR	B < C	Verdadeiro
A > B	NOT		Verdadeiro
A < B	AND	B > C	Verdadeiro
A >= B	OR	B = C	Falso
A <= B	NOT		Falso

## EXERCÍCIOS

1) Tendo as variáveis SALARIO, IR e SALLIQ, e considerando os valores abaixo. Informe se as expressões são verdadeiras ou falsas.

SALARIO	IR	SALLIQ	EXPRESSÃO	V ou F
100,00	0,00	100,00	(SALLIQ >= 100,00)	
200,00	10,00	190,00	(SALLIQ < 190,00)	
300,00	15,00	285,00	SALLIQ = SALARIO - IR	

2) Sabendo que A=3, B=7 e C=4, informe se as expressões abaixo são verdadeiras ou falsas.

a)  $(A+C) > B$  ( )

b)  $B \geq (A + 2)$  ( )

c)  $C = (B - A)$  ( )

d)  $(B + A) \leq C$  ( )

e)  $(C+A) > B$  ( )

3) Sabendo que A=5, B=4 e C=3 e D=6, informe se as expressões abaixo são verdadeiras ou falsas.

a)  $(A > C) \text{ AND } (C \leq D)$  ( )

b)  $(A+B) > 10 \text{ OR } (A+B) = (C+D)$  ( )

c)  $(A \geq C) \text{ AND } (D \geq C)$  ( )

## OPERAÇÕES LÓGICAS

Operações Lógicas são utilizadas quando se torna necessário tomar decisões em um diagrama de bloco.

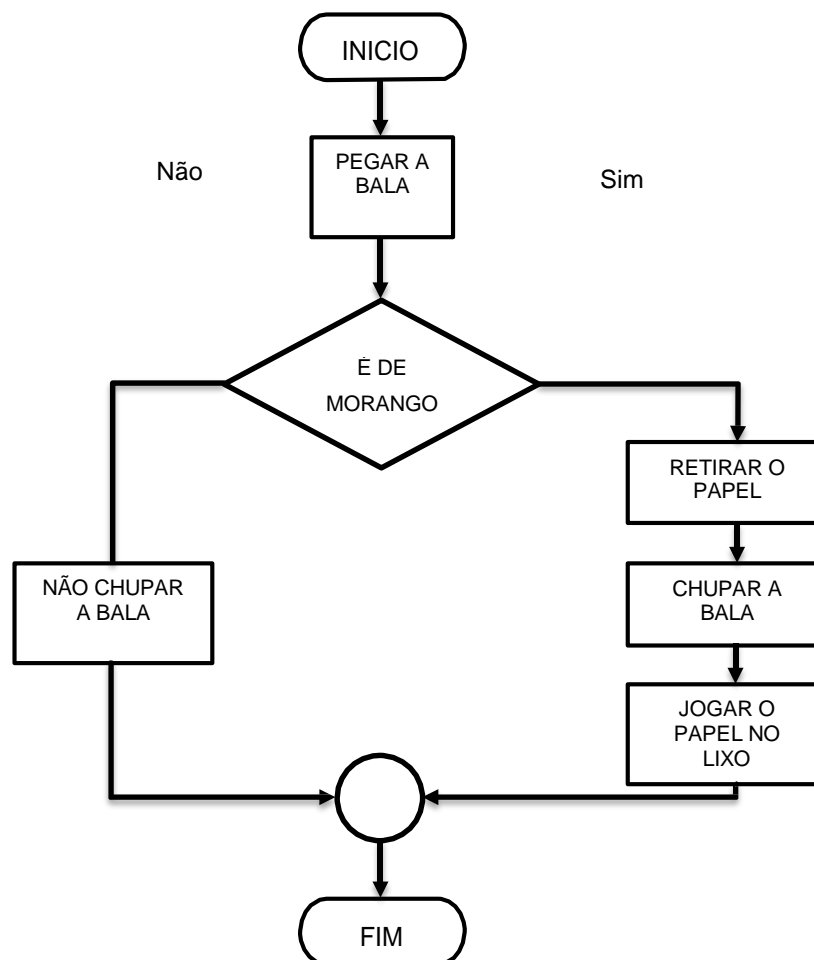
Num diagrama de bloco, toda decisão terá sempre como resposta o resultado VERDADEIRO ou FALSO.

Como no exemplo do algoritmo “CHUPAR UMA BALA”. Imaginemos que algumas pessoas não gostem de chupar bala de Morango, neste caso teremos que modificar o algoritmo para:

“Chupar uma bala”.

- Pegar a bala
- A bala é de morango?
  - ✓ Se **sim**, continue com o algoritmo
  - ✓ Se **não**, não chupe a bala
- Retirar o papel
- Chupar a bala
- Jogar o papel no lixo

Exemplo: Algoritmo “Chupar Bala” utilizando diagrama de Blocos



## EXERCÍCIOS

1) Elabore um diagrama de blocos que leia um número. Se positivo armazene-o em A, se for negativo, em B. No final mostrar o resultado.

2) Ler um número e verificar se ele é par ou ímpar. Quando for par armazenar esse valor em P e quando for ímpar armazená-lo em I. Exibir P e I no final do processamento.

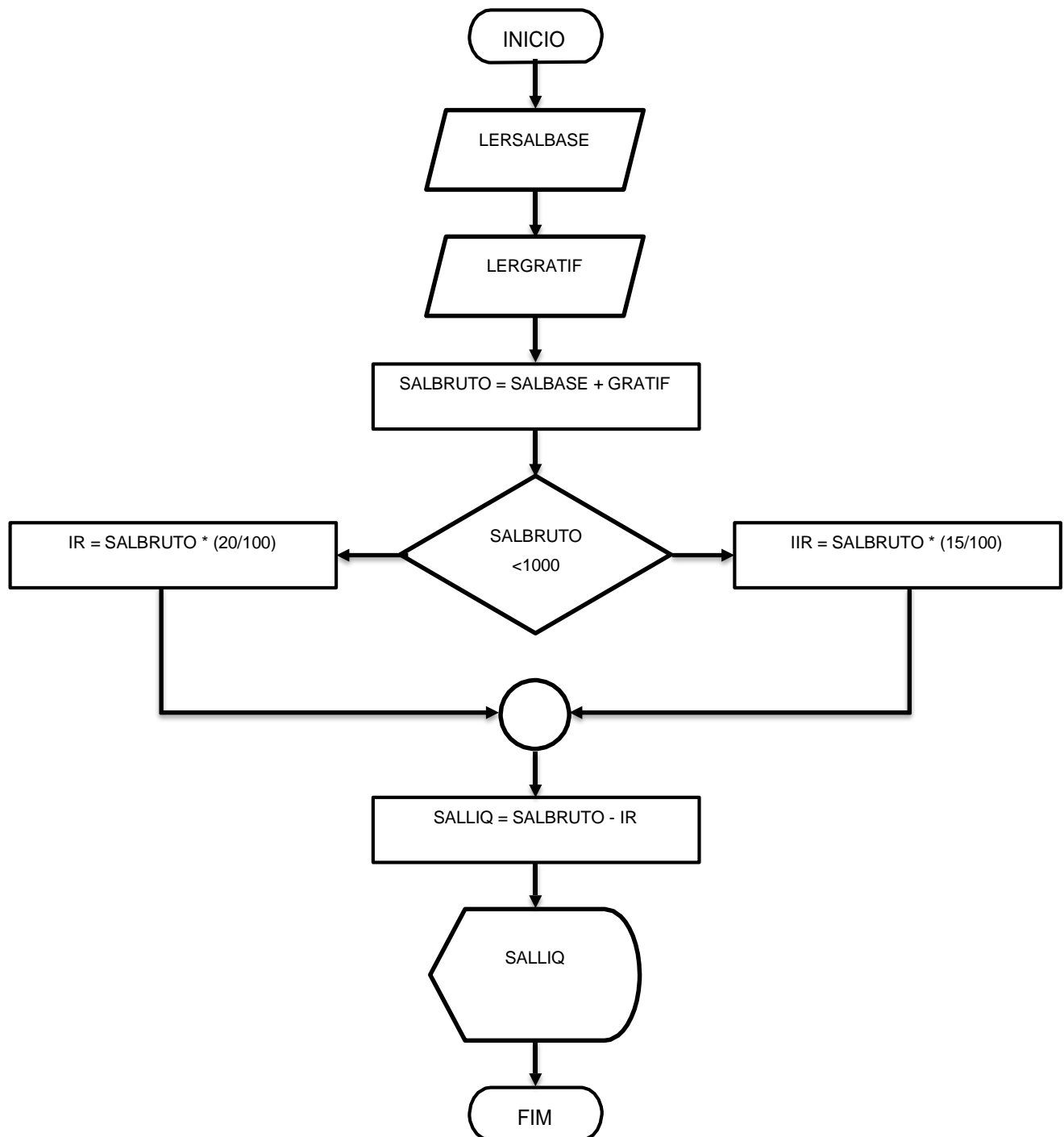


3) Construa um diagrama de blocos para ler uma variável numérica N e imprimi-la somente se a mesma for maior que 100, caso contrário imprimi-la com o valor zero.

4) Tendo como dados de entrada a altura e o sexo de uma pessoa, construa um algoritmo que calcule seu peso ideal, utilizando as seguintes fórmulas:

- Para homens:  $(72.7 * h) - 58$
- Para mulheres:  $(62.1 * h) - 44.7$  (h = altura)

5) Faça um teste de mesa do diagrama apresentado abaixo, de acordo com os dados fornecidos:



Teste o diagrama com os dados abaixo

SALBASE	GRAFIT
3.000,00	1.200,00
1.200,00	400,00
500,00	100,00

SALBASE	SALBRUTO	GRATIF	IR	SALLIQ

SALLIQ

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and extend across the width of the page. There are no margins, text, or other markings on the paper.

Como vimos no capítulo anterior em “Operações Lógicas”, verificamos que na maioria das vezes necessitamos tomar decisões no andamento do algoritmo. Essas decisões interferem diretamente no andamento do programa. Trabalharemos com dois tipos de estrutura. A estrutura de Decisão e a estrutura de Repetição

# COMANDOS DE DECISÃO

Os comandos de decisão ou desvio fazem parte das técnicas de programação que conduzem a estruturas de programas que não são totalmente sequenciais. Com as instruções de SALTO ou DESVIO pode-se fazer com que o programa proceda de uma ou outra maneira, de acordo com as decisões lógicas tomadas em função dos dados ou resultados anteriores.

As principais estruturas de decisão são:

- “**Se Então**”,
- “**Se então Senão**” e
- “**Caso Selecione**”

## SE ... ENTÃO

A estrutura de decisão “SE” normalmente vem acompanhada de um comando, ou seja, se determinada condição for satisfeita pelo comando SE então execute determinado comando.

Imagine um algoritmo que determinado aluno somente estará aprovado se sua média for maior ou igual a 5.0, veja no exemplo de algoritmo como ficaria.

**SE (MEDIA >= 5.0) ENTAO**

**Mensagem = “ALUNO APROVADO”**

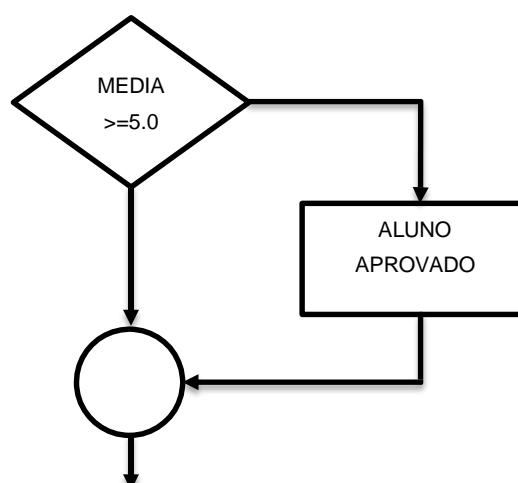
**FIM SE**

Em diagrama de blocos ficaria assim:

SE MEDIA >= 5 ENTÃO

Text1 = “APROVADO”

FIM SE



## SE... ENTÃO... SENÃO....

A estrutura de decisão “SE/ENTÃO/SENÃO”, funciona exatamente como a estrutura “SE”, com apenas uma diferença, em “SE” somente podemos executar comandos caso a condição seja verdadeira, diferente de “SE/SENÃO” pois sempre um comando será executado independente da condição, ou seja, caso a condição seja “verdadeira” o comando da condição será executado, caso contrário o comando da condição “falsa” será executado

Em algoritmo ficaria assim:

SE MÉDIA  $\geq$  5.0 ENTÃO

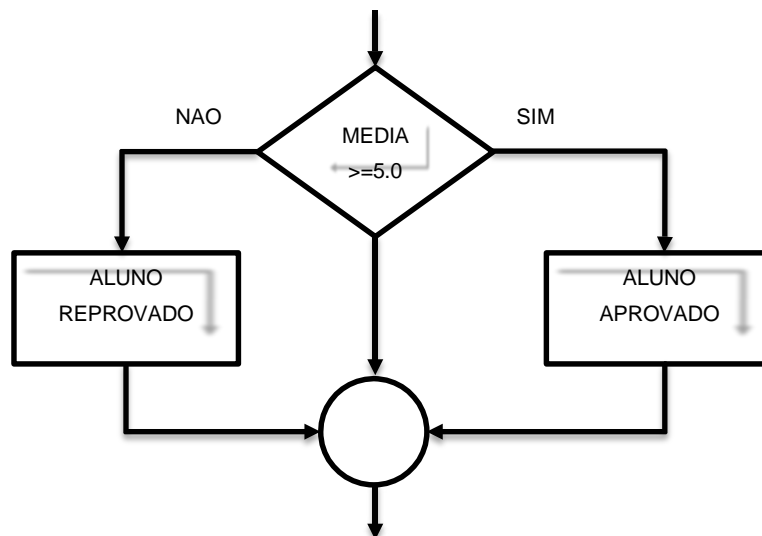
MENSAGEM = “ALUNO APROVADO”

SENÃO

MENSAGEM = “ALUNO REPROVADO”

FIM SE

Em diagrama:



No exemplo anterior está sendo executada uma condição que, se for verdadeira, executa o comando “APROVADO”, caso contrário executa o segundo comando “REPROVADO”. Podemos também dentro de uma mesma condição testar outras condições.

Como no exemplo abaixo:

SE MEDIA  $\geq$  5 ENTAO

SE MEDIA  $\geq$  7 ENTAO

Text1 = "Aluno APROVADO"

SENAO

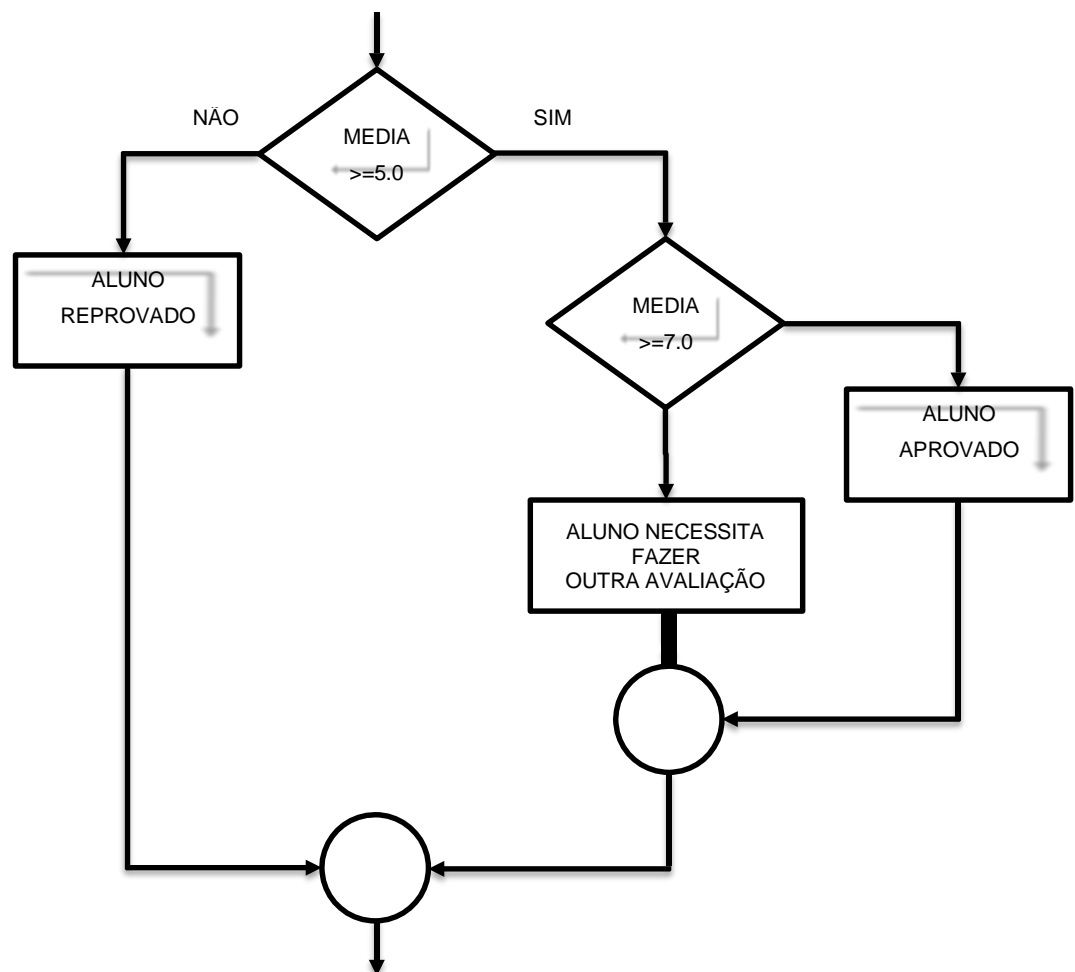
Text1 = "Aluno Necessita fazer outra Avaliação"

FIM SE

SENAO

Text1 = "Aluno REPROVADO"

FIM SE

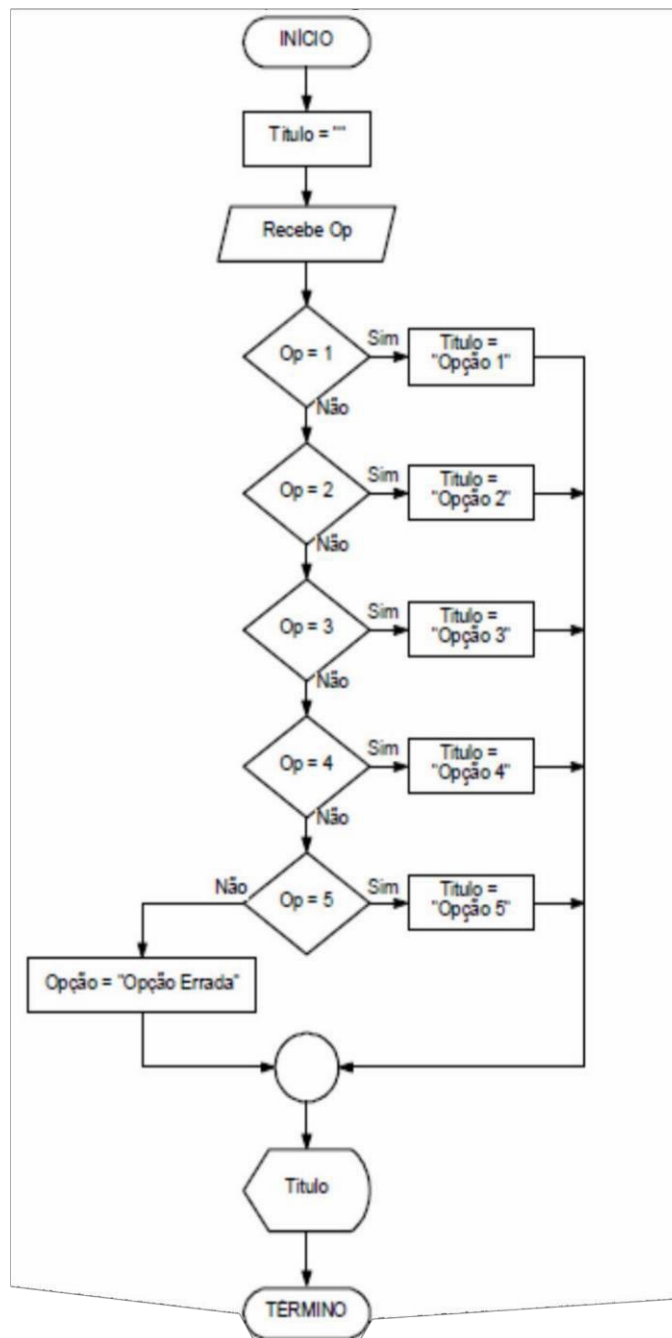


## CASO SELECIONE / SELECT... CASE

A estrutura de decisão CASO/SELECIONE é utilizada para testar, na condição, uma única expressão, que produz um resultado, ou, então, o valor de uma variável, em que está armazenado um determinado conteúdo.

Compara-se, então, o resultado obtido no teste com os valores fornecidos em cada cláusula “Caso”.

No exemplo do diagrama de blocos abaixo, é recebido uma variável “Op” e testado seu conteúdo, caso uma das condições seja satisfeita, é atribuído para a variável Título a String “Opção X”, caso contrário é atribuído a string “Opção Errada”.



Em algoritmo utilizamos a seguinte sequência de comandos para representar o diagrama anterior.

Informe a Opção OP

SELECIONE OP

CASO 1

TITULO = "OPÇÃO 1"

CASO 2

TITULO = "OPÇÃO 2"

CASO 3

TITULO = "OPÇÃO 3"

CASO 4

TITULO = "OPÇÃO 4"

CASO 5

TITULO = "OPÇÃO 5"

SENAO

TITULO = "OPÇÃO ERRADA"

FIM SELECIONE

Mostrar TITULO

## EXERCÍCIOS

- 1) João Papo-de-Pescador, homem de bem, comprou um microcomputador para controlar o rendimento diário de seu trabalho. Toda vez que ele traz um peso de peixes maior que o estabelecido pelo regulamento de pesca do estado de São Paulo (50 quilos) deve pagar uma multa de R\$ 4,00 por quilo excedente. João precisa que você faça um algoritmo que leia a variável P (peso de peixes) e verifique se há excesso. Se houver, gravar na variável E (Excesso) e na variável M o valor da multa que João deverá pagar. Caso contrário mostrar tais variáveis com o conteúdo ZERO.



2) Elabore um algoritmo que leia as variáveis C e N, respectivamente código e número de horas trabalhadas de um operário. E calcule o salário sabendo-se que ele ganha R\$ 10,00 por hora. Quando o número de horas exceder a 50 calcule o excesso de pagamento armazenando-o na variável E, caso contrário zerar tal variável. A hora excedente de trabalho vale R\$ 20,00. No final do processamento imprimir o salário total e o salário excedente.

3) Desenvolva um algoritmo que:

- Leia 4 (quatro) números;
- Calcule o quadrado de cada um;
- Se o valor resultante do quadrado do terceiro for  $\geq 1000$ , imprima-o e finalize;
- Caso contrário, imprima os valores lidos e seus respectivos quadrados.

4) Faça um algoritmo que leia um número inteiro e mostre uma mensagem indicando se este número é par ou ímpar, e se é positivo ou negativo.

5) A Secretaria de Meio Ambiente que controla o índice de poluição mantém 3 grupos de indústrias que são altamente poluentes do meio ambiente. O índice de poluição aceitável varia de 0,05 até 0,25. Se o índice sobe para 0,3 as indústrias do 1º grupo são intimadas a suspenderem suas atividades, se o índice crescer para 0,4 as indústrias do 1º e 2º grupo é intimado a suspenderem suas atividades, se o índice atingir 0,5 todos os grupos devem ser notificados a paralisarem suas atividades. Faça um algoritmo que leia o índice de poluição medido e emita a notificação adequada aos diferentes grupos de empresas.

6) Elabore um algoritmo que dada a idade de um nadador classifique-o em uma das seguintes categorias:

- Infantil A = 5 a 7 anos
- Infantil B = 8 a 11 anos
- Juvenil A = 12 a 13 anos
- Juvenil B = 14 a 17 anos
- Adultos = Maiores de 18 anos

## COMANDOS DE REPETIÇÃO

Utilizamos os comandos de repetição quando desejamos que um determinado conjunto de instruções ou comandos sejam executados um número definido ou indefinido de vezes, ou enquanto um determinado estado de coisas prevalecer ou até que seja alcançado.

Trabalharemos com modelos de comandos de repetição:

- Enquanto x, processar ...
- Até que x, processar ...
- Processar ..., Enquanto x
- Processar ..., Até que x
- Para ... Até ... Seguinte

## ENQUANTO X, PROCESSAR

Neste caso, o bloco de operações será executado enquanto a condição x for verdadeira. O teste da condição será sempre realizado antes de qualquer operação.

Enquanto a condição for verdadeira o processo se repete. Podemos utilizar essa estrutura para trabalharmos com contadores.

Em algoritmo tem a seguinte estrutura:

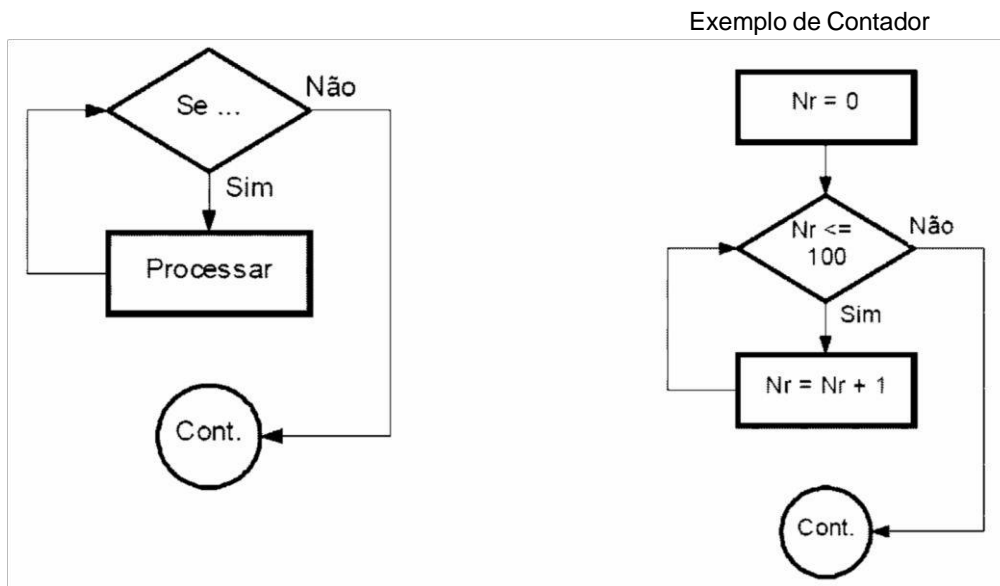
NR = 0

```

SOMA = 0
ENQUANTO ( NR <=100)
  NR = NR+1
  SOMA = SOMA + NR
PROCESSAR
MOSTRAR SOMA

```

Em diagrama de bloco a estrutura é a seguinte:



## ATÉ QUE X, PROCESSAR...

Neste caso, o bloco de operações será executado até que a condição seja satisfeita, ou seja, somente executará os comandos enquanto a condição for falsa.

Em algoritmo tem a seguinte estrutura:

```

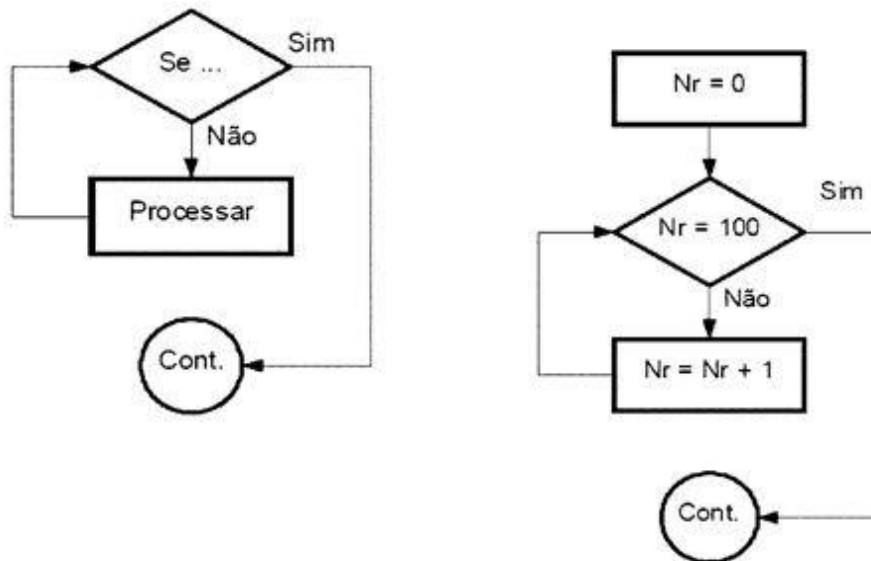
NR = 0
SOMA = 0

ATE QUE (NR > 100)
  NR = NR + 1;
  SOMA = SOMA + NR;
PROCESSAR

```

MOSTRAR SOMA;

Em diagrama de bloco a estrutura é a seguinte:



## PROCESSAR..., ENQUANTO X

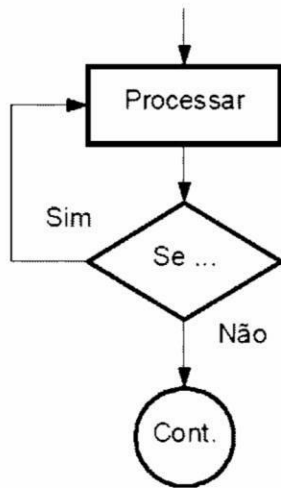
Neste caso primeiro são executados os comandos, e somente depois é realizado o teste da condição. Se a condição for verdadeira, os comandos são executados novamente, caso seja falso é encerrado o comando PROCESSAR.

Em algoritmo tem a seguinte estrutura:

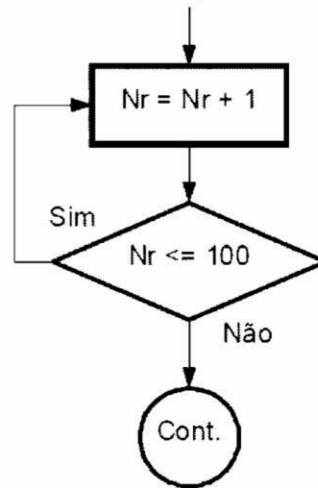
```

NR = 0
SOMA = 0
PROCESSAR
NR = NR+1
    SOMA = SOMA + NR
ENQUANTO ( NR <=100)
MOSTRAR SOMA
  
```

Em diagrama de bloco



Exemplo de Até Diagrama



## PROCESSAR..., ATÉ QUE X

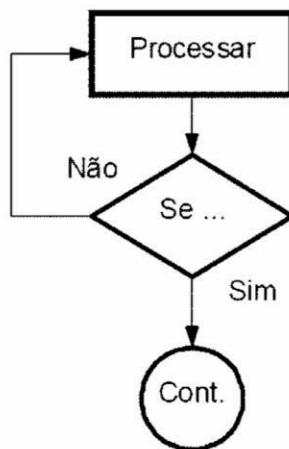
Neste caso, executa-se primeiro o bloco de operações e somente depois é realizado o teste de condição. Se a condição for verdadeira, o fluxo do programa continua normalmente. Caso contrário é processado novamente os comandos antes do teste da condição.

Em algoritmo tem a seguinte estrutura:

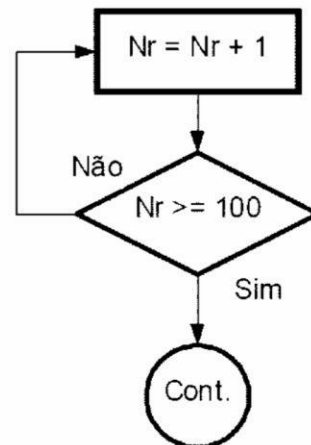
```

NR = 0
SOMA = 0
PROCESSAR
    NR = NR + 1;
    SOMA = SOMA + NR;
ATE QUE (NR > 100)
MOSTRAR SOMA;
  
```

Em diagrama de Bloco



Exemplo de Do .... Loop - Until



## PARA DE... ATÉ... FAÇA

É um comando de repetição que permite que um bloco de comandos seja repetido um número de vezes definido.

SINTAXE:

PARA X DE N ATE M [PASSO Y] FAÇA

X = é a variável de controle que armazenará o contador

N = é o número que começará a contagem

M = É até onde vai essa contagem

PASSO Y = é opcional, somente utiliza se não quiser que seja de 1 em 1.

EXEMPLO:

PARA num DE 1 ATE 10 FAÇA

    ESCREVA NUM

FIM PARA

Observação: Não é necessário incrementar o contador num, pois esse laço de repetição faz isso de forma automática. Nos demais tipos de repetição é necessário incrementar.





- 1000000

algoritmo para calcular o número de grãos que o monge esperava receber.

## ARQUIVOS DE DADOS

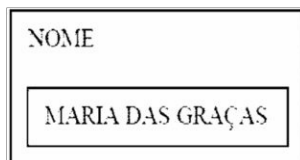
Os dados manipulados até o momento, estavam em memória, ou seja, após a execução do diagrama os dados se perdiam. Para resolver esse problema começaremos a trabalhar com arquivos, onde poderemos guardar os dados e também manipulá-los. Para isso precisamos rever alguns conceitos como: campos, registros e arquivos.

## CONCEITOS BÁSICOS

**CAMPO** é um espaço reservado em memória para receber informações (dados).

Exemplo: Campo Nome, Campo Endereço

Campo na memória



**REGISTRO** é um conjunto de campos

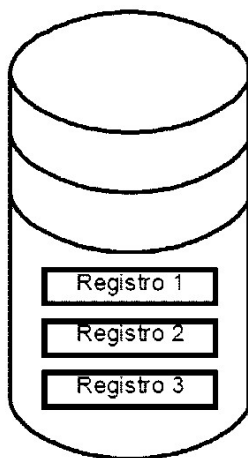
Exemplo: Registro de Clientes

COD-CLI	NOME	ENDEREÇO	FONE
00001	MARIA DAS GRAÇAS	RUA DAS DORES, 1400	888-9876

**ARQUIVO** é um conjunto de registros

Exemplo: O arquivo de Clientes da Empresa, onde estão armazenados os dados de todos os clientes da empresa.

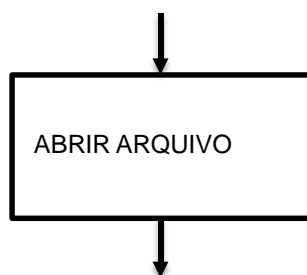
ARQ-CLI



## ABERTURA DE ARQUIVOS

Toda vez que for necessário trabalhar com arquivo, primeiramente precisamos **ABRIR** o arquivo. Abrir o arquivo significa alocar o periférico (disco, disquete) em que o arquivo se encontra, e deixá-lo disponível para leitura/gravação.

O símbolo para abertura de arquivo

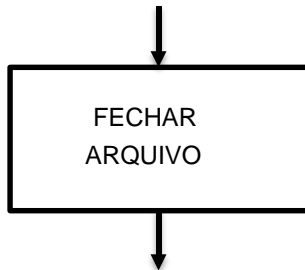


## FECHAMENTO DE ARQUIVOS

Da mesma maneira que precisamos abrir um arquivo antes do processamento, também se faz necessário o fechamento do mesmo, para que suas informações não possam ser violadas ou danificadas.

Fechar um arquivo significa liberar o periférico que estava sendo utilizado.

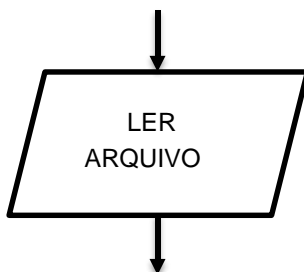
O símbolo para fechamento de arquivo



## LEITURA DE ARQUIVOS

Após abrir um arquivo é necessário **LER** os dados que estão em disco e transferi-los para memória. Essa transferência é feita por registro. Esse procedimento é gerenciado pelo próprio sistema operacional.

O símbolo para leitura de arquivo.

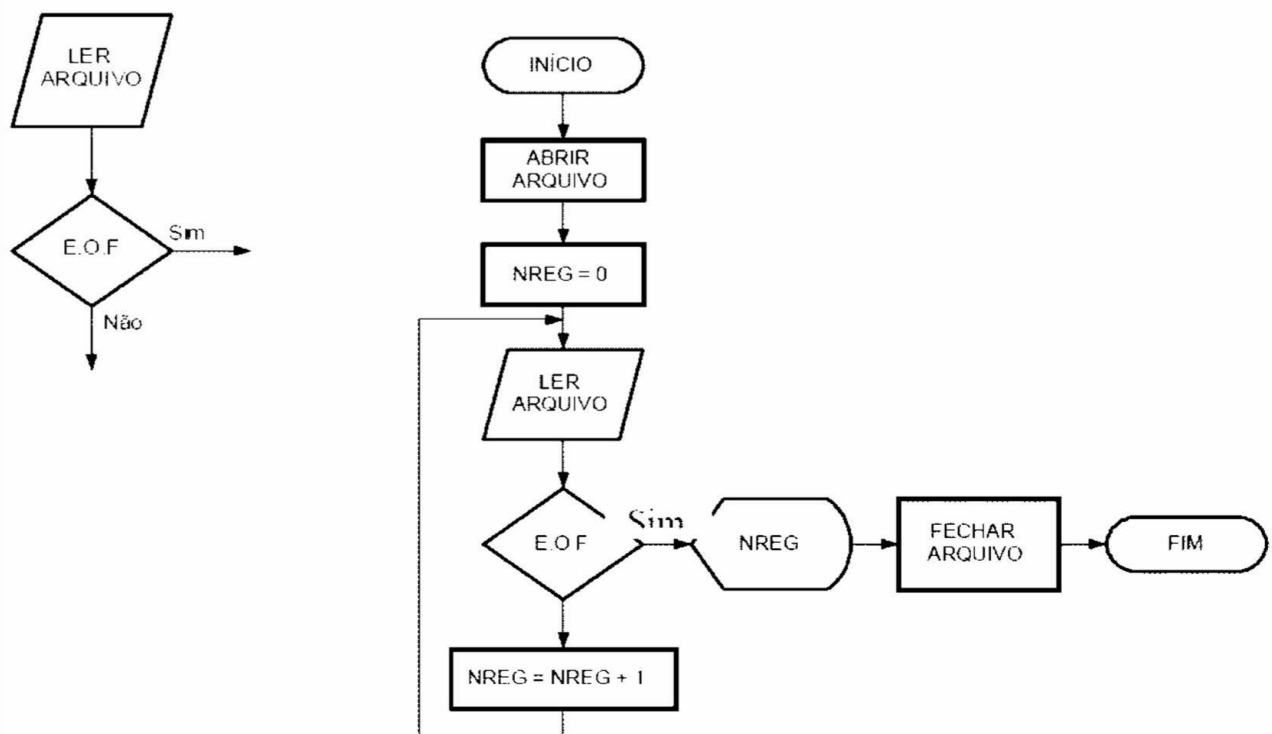


Toda vez que abrimos um arquivo ele posiciona o “**ponteiro**” no primeiro registro, ou seja, no início do arquivo. Para que possamos trabalhar com os dados se torna necessário sabermos onde está o ponteiro do registro. Isso poderemos fazer testando se o ponteiro está no início (**BOF – Bottom Of File**) ou no final do arquivo (**EOF – End Of File**). Esse é sempre executado após a leitura do registro (mudança da posição do ponteiro).

Simbolicamente podemos representar esse passo da seguinte maneira.

Exemplo de diagrama de bloco

Exemplo de diagrama de bloco



## MOVIMENTAÇÃO DE REGISTROS

Como dito no item anterior, quando um arquivo é aberto o ponteiro está no primeiro registro.

A cada leitura do Arquivo o ponteiro se movimenta para o próximo registro e assim por diante.

Como mostra a figura abaixo:

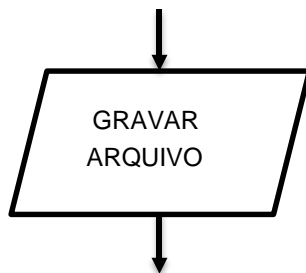


## GRAVAÇÃO DE ARQUIVOS

Da mesma maneira que os registros são lidos de um arquivo, também devemos gravar registros em um arquivo.

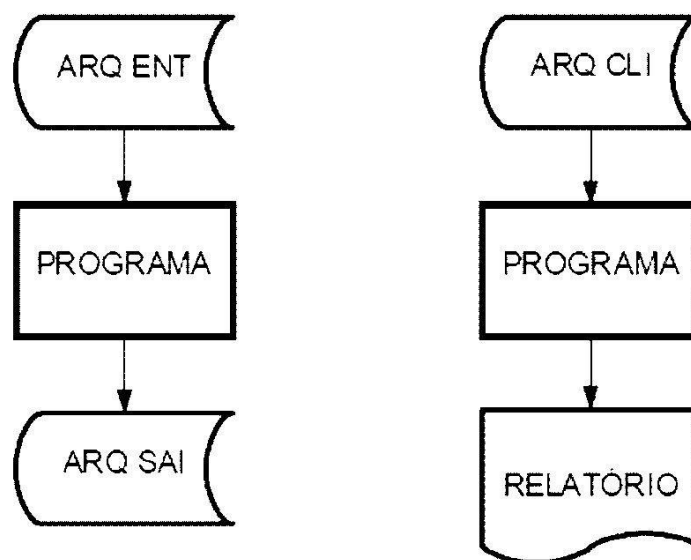
A gravação consiste na transferência de um registro da memória, para um periférico (disco, pen drive).

O símbolo para gravação de arquivos



## MACRO FLUXO

O macro fluxo é a representação gráfica dos arquivos que serão processados em um programa.



Estes dois exemplos de Macro-fluxo dão uma visão geral de como devemos proceder com cada um dos programas. O primeiro diz que haverá um arquivo de entrada, um processamento e um arquivo de saída. Já o segundo exemplo diz que haverá um arquivo de entrada, um processamento, e a saída serão um relatório.

## EXERCÍCIOS

1) Foi feita uma pesquisa entre os habitantes de uma região. Foram coletados os dados de idade, sexo (M/F) e salário. Faça um algoritmo que informa:

- a) A média de salário do grupo
- b) Maior e menor idade do grupo
- c) Quantidade de mulheres com salário até R\$ 100,00
- d) Quantidade de homens

2) Um arquivo de produtos tem os seguintes campos: Código do produto, Descrição, Quantidade em Estoque, Preço de custo, Margem Custo/Venda.

Crie um arquivo com os seguintes campos:

Código do Produto e Preço de Venda. Utilize o cálculo  $\text{Preço de Venda} = \text{Preço de Custo} * \text{Margem CustoVenda}$ .




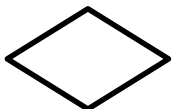


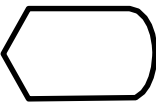

3) Elabore um diagrama de blocos para verificar que produtos precisam ser comprados e a quantidade a ser adquirida:

Tendo as seguintes informações

Código do produto (CODPROD), Quantidade Mínima (QTDMIN), Quantidade Máxima (QTDMAX) e a quantidade em estoque (QTDEST) de cada produto. Um produto somente deverá ser comprado quando: a quantidade em estoque for menor ou igual a quantidade mínima:

$\text{QTCOMPRAR} = (\text{QTDMAX} - \text{QTDEST})$

Grave em outro arquivo: Código do Produto e Quantidade a Comprar

Símbolo	Função
 TERMINAL	Indica o INÍCIO ou FIM de um processamento Exemplo: Início do algoritmo
 PROCESSAMENTO	Processamento em geral Exemplo: Calculo de dois números
 ENTRA/SAÍDA	Operação de entrada e saída de dados Exemplo: Leitura e Gravação de Arquivos
 DECISÃO	Indica uma decisão a ser tomada Exemplo: Verificação de Sexo
 DESVIO	Permite o desvio para um ponto qualquer do programa
 ENTRADA 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
 RELATÓRIO	Relatórios



# Linguagem C



# Linguagem C

## INTRODUÇÃO

Vamos, neste curso, aprender os conceitos básicos da linguagem de programação C a qual tem se tornado cada dia mais popular, devido à sua versatilidade e ao seu poder. Uma das grandes vantagens do C é que ele possui tanto características de "alto nível" quanto de "baixo nível".

Apesar de ser bom, não é pré-requisito do curso um conhecimento anterior de linguagens de programação. É importante uma familiaridade com computadores. O que é importante é que você tenha vontade de aprender, dedicação ao curso e, caso esteja em uma das turmas do curso, acompanhe atentamente as discussões que ocorrem na lista de discussões do curso.

O C nasceu na década de 70. Seu inventor, Dennis Ritchie, implementou-o pela primeira vez usando um DEC PDP-11 rodando o sistema operacional UNIX. O C é derivado de outra linguagem: o B, criado por Ken Thompson. O B, por sua vez, veio da linguagem BCPL, inventada por Martin Richards.

O C é uma linguagem de programação genérica que é utilizada para a criação de programas diversos como processadores de texto, planilhas eletrônicas, sistemas operacionais, programas de comunicação, programas para a automação industrial, gerenciadores de bancos de dados, programas de projeto assistido por computador, programas para a solução de problemas da Engenharia, Física, Química e outras Ciências, etc... É bem provável que o Navegador que você está usando para ler este texto tenha sido escrito em C ou C++.

Estudaremos a estrutura do ANSI C, o C padronizado pela ANSI. Veremos ainda algumas funções comuns em compiladores para alguns sistemas operacionais. Quando não houver equivalentes para as funções em outros sistemas, apresentaremos formas alternativas de uso dos comandos.

Sugerimos que o aluno realmente use o máximo possível dos exemplos, problemas e exercícios aqui apresentados, gerando os programas executáveis com o seu compilador. Quando utilizamos o compilador aprendemos a lidar com mensagens de aviso, mensagens de erro, bugs, etc. Apenas ler os exemplos não basta. O conhecimento de uma linguagem de programação transcende o conhecimento de estruturas e funções. O C exige, além do domínio da linguagem em si, uma familiaridade com o compilador e

experiência em achar "bugs" nos programas. É importante então que o leitor digite, compile e execute os exemplos apresentados.

## PRIMEIROS PASSOS

### O C é "Case Sensitive"

Vamos começar o nosso curso ressaltando um ponto de suma importância: o C é "Case Sensitive", isto é, maiúsculas e minúsculas fazem diferença. Se declarar uma variável com o nome soma ela será diferente de Soma, SOMA, SoMa ou sOmA. Da mesma maneira, os comandos do C if e for, por exemplo, só podem ser escritos em minúsculas, pois senão o compilador não irá interpretá-los como sendo comandos, mas sim como variáveis.

### Dois Primeiros Programas

Vejamos um primeiro programa em C:

```
#include <stdio.h>
/* Um Primeiro Programa */
int main ()
{
    printf ("Ola! Eu estou vivo!\n");
    return(0);
}
```

Compilando e executando este programa você verá que ele coloca a mensagem Olá! Eu estou vivo! Na tela.

### Vamos analisar o programa por partes.

A linha `#include <stdio.h>` diz o compilador que ele deve incluir o arquivocabeçalho `stdio.h`. Neste arquivo existem declarações de funções úteis para entrada e saída de dados (`std` = standard, padrão em inglês; `io` = Input/Output, entrada e saída ==> `stdio` = Entrada e saída padronizadas). Toda vez que você quiser usar uma destas funções deve-se incluir este comando. O C possui diversos Arquivos-cabeçalho.

Quando fazemos um programa, uma boa ideia é usar comentários que ajudem a elucidar o funcionamento do mesmo. No caso acima temos um comentário: `/* Um Primeiro Programa */`. O compilador C desconsidera qualquer coisa que estejam começando com `/*` e terminando com `*/`. Um comentário pode, inclusive, ter mais de uma linha.

A linha `int main()` indica que estamos definindo uma função de nome `main`. Todos os programas em C têm que ter uma função `main`, pois é esta função que será chamada quando o programa for executado. O conteúdo da função é delimitado por chaves `{ }`. O código que estiver dentro das chaves será executado sequencialmente quando a função for chamada. A palavra `int` indica que esta função retorna um inteiro. O que significa este retorno será visto posteriormente, quando estudarmos um pouco mais detalhadamente as funções do C. A última linha do programa, `return(0);`, indica o número inteiro que está sendo retornado pela função, no caso o número 0.

A única coisa que o programa realmente faz é chamar a função `printf()`, passando a string (uma string é uma sequência de caracteres, como veremos brevemente) "Ola! Eu estou vivo!\n" como argumento. É por causa do uso da função `printf()` pelo programa que devemos incluir o arquivo- cabeçalho `stdio.h`. A função `printf()` neste caso irá apenas colocar a string na tela do computador. O `\n` é uma constante chamada de constante barra invertida. No caso, o `\n` é a constante barra invertida de "new line" e ele é interpretado como um comando de mudança de linha, isto é, após imprimir Ola! Eu estou vivo! o cursor passará para a próxima linha. É importante observar também que os comandos do C terminam com;

Podemos agora tentar um programa mais complicado:

```
#include <stdio.h>
#include <conio.h>
int main()
{
    /* Declaracao de Variaveis */
    int Dias;
    float Anos;

    /* Entrada de Dados */
    printf ("Entre com o numero de dias: ");
    scanf ("%d",&Dias);

    /* Conversao Dias->Anos */
    Anos=Dias/365.25;

    printf ("\n\n%d dias equivalem a %2.2f anos.\n",Dias,Anos);

    getch();
}
```

Vamos entender como o programa acima funciona. São declaradas duas variáveis chamadas `Dias` e `Anos`. A primeira é um `int` (inteiro) e a segunda um `float` (ponto flutuante). As variáveis declaradas como ponto flutuante existem para armazenar números que possuem casas decimais, como 5,1497.

É feita então uma chamada à função `printf()`, que coloca uma mensagem na tela.

Queremos agora ler um dado que será fornecido pelo usuário e colocá-lo na variável inteira Dias. Para tanto usamos a função `scanf()`. A string `"%d"` diz à função que iremos ler um inteiro. O segundo parâmetro passado à função diz que o dado lido deverá ser armazenado na variável Dias. É importante ressaltar a necessidade de se colocar um `&` antes do nome da variável a ser lida quando se usa a função `scanf()`. O motivo disto só ficará claro mais tarde. Observe que, no C, quando temos mais de um parâmetro para uma função, eles serão separados por vírgula.

Temos então uma expressão matemática simples que atribui a Anos o valor de Dias dividido por 365.25 (365.25 é uma constante ponto flutuante 365,25). Como Anos é uma variável float o compilador fará uma conversão automática entre os tipos das variáveis (veremos isto com detalhes mais tarde).

A segunda chamada à função `printf()` tem três argumentos. A string `"\n\n%d dias equivalem a %f anos.\n"` diz à função para pular duas linhas, colocar um inteiro na tela, colocar a mensagem " dias equivale a ", colocar um valor float na tela, colocar a mensagem " anos." e pular outra linha. Os outros parâmetros são as variáveis, Dias e Anos, das quais devem ser lidos os valores do inteiro e do float, respectivamente.

## EXERCÍCIOS

1 - Veja como você está. O que faz o seguinte programa?

```
#include <stdio.h>
int main()
{
    int x;
    scanf("%d", &x);
    printf("%d", x);
    return(0);
}
```

## INTRODUÇÃO BÁSICA ÀS ENTRADAS E SAÍDAS

### CARACTERES

Os caracteres são um tipo de dado: o char. O C trata os caracteres ('a', 'b', 'x', etc. ...) como sendo variáveis de um byte (8 bits). Um bit é a menor unidade de armazenamento de informações em um computador. Os inteiros (ints) têm um número maior de bytes. Dependendo da implementação do compilador, eles podem ter 2 bytes (16 bits) ou 4 bytes (32 bits).

Na linguagem C, também podemos usar um char para armazenar valores numéricos inteiros, além de usá-lo para armazenar caracteres de texto. Para indicar um caractere de texto usamos apóstrofes. Veja um exemplo de programa que usa caracteres:

```
#include <stdio.h>

#include <conio.h>

int main()

{

    char Ch;

    Ch='D';

    printf ("%c",Ch);

    getch();

}
```

No programa acima, %c indica que printf() deve colocar um caractere na tela. Como vimos anteriormente, um char também é usado para armazenar um número inteiro. Este número é conhecido como o código ASCII correspondente ao caractere. Veja o programa abaixo:

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    char Ch;
    Ch='D';
    printf ("%d",Ch); /* Imprime o caracter como inteiro */
    getch();
}
```

Este programa vai imprimir o número 68 na tela, que é o código ASCII correspondente ao caractere 'D' (d maiúsculo).

Muitas vezes queremos ler um caractere fornecido pelo usuário. Para isto as funções mais usadas, quando se está trabalhando em ambiente DOS ou Windows, são getch() e getche(). Ambas retornam o caractere pressionado. getche() imprime o caractere na tela antes de retorná-lo e getch() apenas retorna o caractere pressionado sem imprimi-lo na tela. Ambas as funções podem ser encontradas no arquivo de cabeçalho conio.h. Geralmente estas funções não estão disponíveis em ambiente Unix (compiladores cc e gcc), pois não fazem parte do padrão ANSI. Podem ser substituídas pela função scanf(), porém sem as mesmas funcionalidades.

# STRINGS

No C uma string é um vetor de caracteres terminado com um caractere nulo. O caractere nulo é um caractere com valor inteiro igual a zero (código ASCII igual a 0). O terminador nulo também pode ser escrito usando a convenção de barra invertida do C como sendo '\0'. Embora o assunto vetores seja discutido posteriormente, veremos aqui os fundamentos necessários para que possamos utilizar as strings. Para declarar uma string, podemos usar o seguinte formato geral:

```
char nome_da_string[tamanho];
```

Isto declara um vetor de caracteres (uma string) com número de posições igual a tamanho. Note que, como temos que reservar um caractere para ser o terminador nulo, temos que declarar o comprimento da string como sendo, no mínimo, um caractere maior que a maior string que pretendemos armazenar.

Vamos supor que declaremos uma string de 7 posições e coloquemos a palavra João nela. Teremos:

J	o	a	o	\0	...	...
---	---	---	---	----	-----	-----

No caso acima, as duas células não usadas têm valores indeterminados. Isto acontece porque o C não inicializa variáveis, cabendo ao programador esta tarefa. Portanto as únicas células que são inicializadas são as que contêm os caracteres 'J', 'o', 'a', 'o' e '\0'.

Se quisermos ler uma string fornecida pelo usuário podemos usar a função `gets()`. Um exemplo do uso desta função é apresentado abaixo. A função `gets()` coloca o terminador nulo na string, quando você aperta a tecla "Enter".

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

int main ()
{
    char string[100];
    printf ("Digite uma string: ");
    gets (string);
    printf ("\n\nVoce digitou %s",string);
    getch();
}
```

Neste programa, o tamanho máximo da string que você pode entrar é uma string de 99 caracteres. Se você entrar com uma string de comprimento maior, o programa irá aceitar, mas os resultados podem ser desastrosos. Veremos porque posteriormente.

Como as strings são vetores de caracteres, para se acessar um determinado caracter de uma string, basta "indexarmos", ou seja, usarmos um índice para acessarmos o caracter desejado dentro da string. Suponha uma string chamada str. Podemos acessar a segunda letra de str da seguinte forma:

```
str[1] = 'a';
```

Por que se está acessando a segunda letra e não a primeira? Na linguagem C, o índice começa em zero. Assim, a primeira letra da string sempre estará na posição 0. A segunda letra sempre estará na posição 1 e assim sucessivamente. Segue um exemplo que imprimirá a segunda letra da string "Joao", apresentada acima. Em seguida, ele mudará esta letra e apresentará a string no final.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    char str[10] = "Joao";
    printf("\n\nString: %s", str);
    printf("\nSegunda letra: %c", str[1]);
    str[1] = 'U';
    printf("\nAgora a segunda letra eh: %c", str[1]);
    printf("\n\nString resultante: %s", str);

    getch();
}
```

Nesta string, o terminador nulo está na posição 4. Das posições 0 a 4, sabemos que temos caracteres válidos, e, portanto, podemos escrevê-los. Note a forma como inicializamos a string str com os caracteres 'J' 'o' 'a' 'o' e '\0' simplesmente declarando `char str[10] = "Joao"`. Veremos, posteriormente que "Joao" (uma cadeia de caracteres entre aspas) é o que chamamos de string constante, isto é, uma cadeia de caracteres que está pré-carregada com valores que não podem ser modificados. Já a string str é uma string variável, pois podemos modificar o que nela está armazenado, como de fato fizemos.

No programa acima, %s indica que printf() deve colocar uma string na tela. Vamos agora fazer uma abordagem inicial às duas funções que já temos usado para fazer a entrada e saída.

## COMANDO printf

A função printf() tem a seguinte forma geral:

```
printf (string_de_controle,lista_de_argumentos);
```



Teremos, na string de controle, 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. Apresentamos agora alguns dos códigos %:

#### CÓDIGO SIGNIFICADO

%d	Inteiro
%f	Float
%c	Caractere
%s	String
%%	Coloca na tela um %

Vamos ver alguns exemplos de printf() e o que eles exibem:

```
printf ("Teste %% %%") -> "Teste % %"
printf ("%f",40.345) -> "40.345"
printf ("Um caractere %c e um inteiro %d",'D',120) -> "Um caractere D
e um inteiro 120"
printf ("%s e um exemplo","Este") -> "Este e um exemplo"
printf ("%s%d%%","Juros de ",10) -> "Juros de 10%"
```

Maiores detalhes sobre a função printf() (incluindo outros códigos de controle) serão vistos posteriormente, mas podem ser consultados de antemão pelos interessados.

## COMANDO scanf

O formato geral da função scanf() é:

scanf (string-de-controle,lista-de-argumentos);

Usando a função scanf() podemos pedir dados ao usuário. Um exemplo de uso, pode ser visto acima. Mais uma vez, devemos ficar atentos a fim de colocar o mesmo número de argumentos que o de códigos de controle na string de controle. Outra coisa importante é lembrarmos-nos de colocar o & antes das variáveis da lista de argumentos. É impossível justificar isto agora, mas veremos depois a razão para este procedimento. Maiores detalhes sobre a função scanf() serão vistos posteriormente, mas podem ser consultados de antemão pelos interessados.

## EXERCÍCIOS

- a) Escreva um programa que leia um caracter digitado pelo usuário, imprima o caracter digitado e o código ASCII correspondente a este caracter.
- b) Escreva um programa que leia duas strings e as coloque na tela. Imprima também a segunda letra de cada string.

## COMENTÁRIOS

Como já foi dito, o uso de comentários torna o código do programa mais fácil de entender. Os comentários do C devem começar com `/*` e terminar com `*/`. O C padrão não permite comentários aninhados (um dentro do outro), mas alguns compiladores os aceitam.

## PALAVRAS RESERVADAS DO C

Todas as linguagens de programação têm palavras reservadas. As palavras reservadas não podem ser usadas a não ser nos seus propósitos originais, isto é, não podemos declarar funções ou variáveis com os mesmos nomes. Como o C é "case sensitive" podemos declarar uma variável `For`, apesar de haver uma palavra reservada `for`, mas isto não é uma coisa recomendável de se fazer pois pode gerar confusão. Apresentamos a seguir as palavras reservadas do ANSI C. Veremos o significado destas palavras chave à medida que o curso for

--	--	--	--

# VARIÁVEIS, CONSTANTES, OPERADORES E EXPRESSÕES

## NOMES DAS VARIÁVEIS

As variáveis no C podem ter qualquer nome se duas condições forem satisfeitas: o nome deve começar com uma letra ou sublinhado (`_`) e os caracteres subsequentes devem ser letras, números ou sublinhado (`_`). Há apenas mais duas restrições: o nome de uma variável não pode ser igual a uma palavra reservada, nem igual ao nome de uma função declarada pelo programador, ou pelas bibliotecas do C. Variáveis de até 32 caracteres são aceitas. Mais uma coisa: é bom sempre lembrar que o C é "case sensitive" e, portanto, deve-se prestar atenção às maiúsculas e minúsculas.

Dicas quanto aos nomes de variáveis...

- É uma prática tradicional do C, usar letras minúsculas para nomes de variáveis e maiúsculas para nomes de constantes. Isto facilita na hora da leitura do código;
- Quando se escreve código usando nomes de variáveis em português, evita-se possíveis conflitos com nomes de rotinas encontrados nas diversas bibliotecas, que são em sua maioria absoluta, palavras em inglês.

## OS TIPOS DO C

O C tem tipos básicos: `char`, `int`, `float`, `double`.

- O `double` é o ponto flutuante duplo e pode ser visto como um ponto flutuante com muito mais precisão.

Para cada um dos tipos de variáveis existem os modificadores de tipo. Os modificadores de tipo do C são quatro: `signed`, `unsigned`, `long` e `short`. Ao `float` não se pode aplicar nenhum e ao `double` pode-se aplicar apenas o `long`. Os quatro modificadores podem ser aplicados a inteiros. A intenção é que `short` e `long` devam prover tamanhos diferentes de inteiros onde isto for prático. Inteiros menores (`short`) ou maiores (`long`). `int` normalmente terá o tamanho natural para determinada máquina. Assim, numa máquina de 16 bits, `int` provavelmente terá 16 bits. Numa máquina de 32, `int` deverá ter 32 bits. Na verdade, cada compilador é livre para escolher tamanhos adequados para o seu próprio hardware, com a única restrição de que `short` ints e `ints` devem ocupar pelo menos 16 bits, `longs` ints pelo menos 32 bits, e `short int` não pode ser maior que `int`, que não pode ser maior que `long int`. O modificador `unsigned` serve para especificar variáveis sem sinal. Um `unsigned`

int será um inteiro que assumirá apenas valores positivos. A seguir estão listados os tipos de dados permitidos e seus valores máximos e mínimos em um compilador típico para um hardware de 16 bits. Também nesta tabela está especificado o formato que deve ser utilizado para ler os tipos de dados com a função scanf():

O tipo long double é o tipo de ponto flutuante com maior precisão. É importante observar que os intervalos de ponto flutuante, na tabela acima, estão indicados em faixa de expoente, mas os números podem assumir valores tanto positivos quanto negativos.

Tipo	Nro de bits	Formato leitura	Intervalo	
			Início	Fim
Char	8	%c	-128	127
unsigned char	8	%c	0	255
signed char	8	%c	-128	127
int	16	%i	-32.768	32.767
unsigned int	16	%u	0	65.535
signed int	16	%i	-32.768	32.767
short int	16	%hi	-32.768	32.767
unsigned short	16	%hu	0	65.535
intsigned short int	16	%hi	-32.768	32.767
long int	32	%li	-	2.147.483.647
signed long int	32	%li	-2.147.483.648	2.147.483.647
unsigned long int	32	%lu	2.147.483.648	4.294.967.295
float	32	%f	3,4E-38	3,4E+38
double	64	%lf	1,7E-308	1,7E+308
long double	80	%Lf	3,4E-4932	3,4E+4932

## Declaração e Inicialização de Variáveis

As variáveis no C devem ser declaradas antes de serem usadas. A forma geral da declaração de variáveis é:

```
tipo_da_variável lista_de_variáveis;
```

As variáveis da lista de variáveis terão todas o mesmo tipo e deverão ser separadas por vírgula. Como o tipo default do C é o int, quando vamos declarar variáveis int com algum dos modificadores de tipo, basta colocar o nome do modificador de tipo. Assim um long basta para declarar um long int. Por exemplo, as declarações declaram duas variáveis do tipo char (ch e letra), uma variável long int (count) e um float pi.

```
char ch, letra; long count;
float pi;
```

Há três lugares nos quais podemos declarar variáveis. O primeiro é fora de todas as funções do programa. Estas variáveis são chamadas variáveis globais e podem ser usadas a partir de qualquer lugar no programa. Pode-se dizer que, como elas estão fora de todas as funções, todas as funções as veem. O segundo lugar no qual se pode declarar variáveis é no início de um bloco de código. Estas variáveis são chamadas locais e só têm validade dentro do bloco no qual são declaradas, isto é, só a função à qual ela pertence sabe da existência desta variável, dentro do bloco no qual foram declaradas. O terceiro lugar onde se pode declarar variáveis é na lista de parâmetros de uma função. Mais uma vez, apesar de estas variáveis receberem valores externos, estas variáveis são conhecidas apenas pela função onde são declaradas.

Veja o programa abaixo:

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

int contador;

int func1(int j)
{
    /* aqui viria o código da funcao ... */
}

int main()
{
    char condicao;
    int i;
    for (i=0; i<100; i=i+1)
    {
        /* Bloco do for */
        float f2;
        /* etc. ... */
        func1(i);
    }
}
```

```

    /* etc. ... */
    getch();
}

```

A variável contador é uma variável global, e é acessível de qualquer parte do programa. As variáveis condição e i, só existem dentro de main(), isto é são variáveis locais de main. A variável float f2 é um exemplo de uma variável de bloco, isto é, ela somente é conhecida dentro do bloco do for, pertencente à função main. A variável inteira j é um exemplo de declaração na lista de parâmetros de uma função (a função func1).

As regras que regem onde uma variável é válida chamam-se regras de escopo da variável. Há mais dois detalhes que devem ser ressaltados. Duas variáveis globais não podem ter o mesmo nome. O mesmo vale para duas variáveis locais de uma mesma função. Já duas variáveis locais, de funções diferentes, podem ter o mesmo nome sem perigo algum de conflito.

Podemos inicializar variáveis no momento de sua declaração. Para fazer isto podemos usar a forma geral.

```
tipo_da_variável nome_da_variável = constante;
```

Isto é importante pois quando o C cria uma variável ele não a inicializa. Isto significa que até que um primeiro valor seja atribuído à nova variável ela tem um valor indefinido e que não pode ser utilizado para nada. Nunca presuma que uma variável declarada vale zero ou qualquer outro valor. Exemplos de inicialização são dados abaixo:

```

char ch='D';
int count=0;
float pi=3.141;

```

Ressalte-se novamente que, em C, uma variável tem que ser declarada no início de um bloco de código. Assim, o programa a seguir não é válido em C (embora seja válido em C++).

```

int main()
{
    int i;
    int j;
    j = 10;
    int k = 20;
    /* Esta declaracao de variável não é válida, pois não está sendo feita no
    início do bloco */
    getch();
}

```

## EXERCÍCIOS

- a) Escreva um programa que:
- declare 6 variáveis inteiras e atribua os valores 10, 20, 30, ..., 60 a elas.
  - declare 6 variáveis caracteres e atribua a elas as letras c, o, e, l, h, a .

Finalmente, o programa deverá imprimir, usando todas as variáveis declaradas:

As variáveis inteiras contém os números: 10,20,30,40,50,60

O animal contido nas variáveis caracteres é a coelha.

## Constantes

Constantes são valores que são mantidos fixos pelo compilador. Já usamos constantes neste curso. São consideradas constantes, por exemplo, os números e caracteres como 45.65 ou 'n', etc...

## Constantes dos tipos básicos

Abaixo vemos as constantes relativas aos tipos básicos do C:

Tipo de Dado	Exemplos de Constantes
char	'b' '\n' '\0'
int	2 32000 -
long int	100000130 -
short int	100467 -30
unsigned int	50000
float	0.0 2335678.7 -12 .3e-10
double	12546354334.0 0.0000034236556 -

## Constantes hexadecimais e octais

Muitas vezes precisamos inserir constantes hexadecimais (base dezesseis) ou octais (base oito) no nosso programa. O C permite que se faça isto. As constantes hexadecimais começam com 0x. As constantes octais começam em 0.

Alguns exemplos:

Constante	Tipo
0xEF	Constante Hexadecimal (8 bits)
0x12A4	Constante Hexadecimal (16 bits)
03212	Constante Octal (12 bits)
034215432	Constante Octal (24 bits)

Nunca escreva, portanto 013 achando que o C vai compilar isto como se fosse 13. Na linguagem C 013 é diferente de 13!

## Constantes strings

Já mostramos como o C trata strings. Vamos agora alertar para o fato de que uma string "Joao" é na realidade uma constante string. Isto implica, por exemplo, no fato de que 't' é diferente de "t", pois 't' é um char enquanto que "t" é uma constante string com dois chars onde o primeiro é 't' e o segundo é '\0'.

## Constantes de barra invertida

O C utiliza, para nos facilitar a tarefa de programar, vários códigos chamados códigos de **barra invertida**. Estes são caracteres que podem ser usados como qualquer outro. Uma lista com alguns dos códigos de barra invertida é dada a seguir:

Código	Significado
\b	Retrocesso ("back")
\f	Alimentação de formulário ("form feed")
\n	Nova linha ("new line")
\t	Tabulação horizontal ("tab")
\"	Aspas
\'	Apóstrofo
\0	Nulo (0 em decimal)
\\	Barra invertida
\v	Tabulação vertical
\a	Sinal sonoro ("beep")
\N	Constante octal (N é o valor da constante)
\xN	Constante hexadecimal (N é o valor da constante)



# Operadores Aritméticos e de Atribuição

Os operadores aritméticos são usados para desenvolver operações matemáticas. A seguir apresentamos a lista dos operadores aritméticos do C:

Operador	Ação
+	Soma (inteira e ponto flutuante)
-	Subtração ou Troca de sinal (inteira e ponto flutuante)
*	Multiplicação (inteira e ponto flutuante)
/	Divisão (inteira e ponto flutuante)
%	Resto de divisão (de inteiros)
++	Incremento (inteiro e ponto flutuante)
--	Decremento (inteiro e ponto flutuante)

O C possui operadores unários e binários. Os unários agem sobre uma variável apenas, modificando ou não o seu valor, e retornam o valor final da variável. Os binários usam duas variáveis e retornam um terceiro valor, sem alterar as variáveis originais. A soma é um operador binário pois pega duas variáveis, soma seus valores, sem alterar as variáveis, e retorna esta soma. Outros operadores binários são os operadores - (subtração), \*, / e %. O operador

- Como troca de sinal é um operador unário que não altera a variável sobre a qual é aplicado, pois ele retorna o valor da variável multiplicado por -1.

O operador / (divisão) quando aplicado a variáveis inteiras, nos fornece o resultado da divisão inteira; quando aplicado a variáveis em ponto flutuante nos fornece o resultado da divisão "real". O operador % fornece o resto da divisão de dois inteiros.

Assim seja o seguinte trecho de código:

```
int a = 17, b = 3;
int x, y;
float z = 17. , z1, z2;
x = a / b;
y = a % b;
z1 = z / b;
z2 = a/b;
```

Ao final da execução destas linhas, os valores calculados seriam  $x = 5$ ,  $y = 2$ ,  $z1 = 5.666666$  e  $z2 = 5.0$ . Note que, na linha correspondente a  $z2$ , primeiramente é feita uma divisão inteira (pois os dois operandos são inteiros). Somente após efetuada a divisão é que o resultado é atribuído a uma variável float.

Os operadores de incremento e decremento são unários que alteram a variável sobre a qual estão aplicados. O que eles fazem é incrementar ou decrementar, a variável sobre a qual estão aplicados, de 1. Então

- `x++;`
- `x--;`

São equivalentes a

- `x=x+1;`
- `x=x-1;`

Estes operadores podem ser pré-fixados ou pós-fixados. A diferença é que quando são pré-fixados eles incrementam e retornam o valor da variável já incrementada. Quando são pós-fixados eles retornam o valor da variável sem o incremento e depois incrementam a variável. Então, em

```
x=23;
y=x++;
```

Teremos, no final, `y=23` e `x=24`. Em

```
x=23; y=++x;
```

Teremos, no final, `y=24` e `x=24`.

Uma curiosidade: a linguagem de programação C++ tem este nome pois ela seria um "incremento" da linguagem C padrão. A linguagem C++ é igual à linguagem C só que com extensões que permitem a programação orientada a objeto, o que é um recurso extra.

O operador de atribuição do C é o `=`. O que ele faz é pegar o valor à direita e atribuir à variável da esquerda. Além disto ele retorna o valor que ele atribuiu.

Isto faz com que as seguintes expressões sejam válidas:

```
x=y=z=1.5; /* Expressao 1 */
if (k=w) ... /* Expressão 2 */
```

A expressão 1 é válida, pois quando fazemos `z=1.5` ela retorna 1.5, que é passado adiante, fazendo `y = 1.5` e posteriormente `x = 1.5`. A expressão 2 será verdadeira se `w` for diferente de zero, pois este será o valor retornado por `k=w`. Pense bem antes de usar a expressão dois, pois ela pode gerar erros de interpretação. Você não está comparando `k` e `w`. Você está atribuindo o valor de `w` a `k` e usando este valor para tomar a decisão.

## EXERCÍCIOS

a) Diga o resultado das variáveis `x`, `y` e `z` depois da seguinte sequência de operações:

```
int x,y,z;
x=y=10;
```

```

z=++x;
x=-x;
y++;
x=x+y-(z--);

```

## OPERADORES RELACIONAIS E LÓGICOS

Os operadores relacionais do C realizam comparações entre variáveis.

São eles:

Operador	Ação
>	Maior do que
>=	Maior ou igual a
<	Menor do que
<=	Menor ou igual a
==	Igual a
!=	Diferente de

Os operadores relacionais retornam verdadeiro (1) ou falso (0). Para verificar o funcionamento dos operadores relacionais, execute o programa abaixo:

```

int main()
{
    int i, j;

    printf("\nEntre com dois números inteiros: ");
    scanf("%d%d", &i, &j);

    printf("\n%d == %d é %d\n", i, j, i==j);
    printf("\n%d != %d é %d\n", i, j, i!=j);
    printf("\n%d <= %d é %d\n", i, j, i<=j);
    printf("\n%d >= %d é %d\n", i, j, i>=j);
    printf("\n%d < %d é %d\n", i, j, i<j);
    printf("\n%d > %d é %d\n", i, j, i>j);
    getch();
}

```

Você pode notar que o resultado dos operadores relacionados é sempre igual a 0 (falso) ou 1 (verdadeiro).

Para fazer operações com valores lógicos (verdadeiro e falso) temos os operadores lógicos:

Operador	Ação
&&	AND (E)
	OR (OU)
!	NOT (NÃO)

Usando os operadores relacionais e lógicos podemos realizar uma grande gama de testes. A tabela-verdade destes operadores é dada a seguir:

p	q	p AND q	p OR q
Falso	Falso	Falso	Falso
Falso	Verdadeiro	Falso	Verdadeiro
Verdadeiro	Falso	Falso	Verdadeiro
Verdadeiro	Verdadeiro	Verdadeiro	Verdadeiro

O programa a seguir ilustra o funcionamento dos operadores lógicos. Compile-o e faça testes com vários valores para i e j:

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int i, j;
    printf("informe dois números(cada um sendo 0 ou 1): ");
    scanf("%d%d", &i, &j);
    printf("%d AND %d é %d\n", i, j, i && j);
    printf("%d OR %d é %d\n", i, j, i || j);
    printf("NOT %d é %d\n", i, !i);
    getch();
}
```

Exemplo: No trecho de programa abaixo a operação j++ será executada, pois o resultado da expressão lógica é verdadeiro:

```
int i = 5, j = 7;
if ( (i > 3) && (j <= 7) && (i != j) )
    j++;
V      AND    V      AND    V = V
```

Mais um exemplo. O programa abaixo, imprime na tela somente os números pares entre 1 e 100, apesar da variação de i ocorrer de 1 em 1:

```
/* Imprime os números pares entre 1 e 100. */
#include <stdio.h>
#include <conio.h>
int main()
{
    int i;
    for(i=1; i<=100; i++)
        if(!(i%2))
            printf("%d ",i);
    /* o operador de resto dará falso (zero) */
    /* quando usada c/ número par. Esse resultado*/
    /* é invertido pelo ! */
    getch();
}
```

## Expressões

Expressões são combinações de variáveis, constantes e operadores. Quando montamos expressões temos que levar em consideração a ordem com que os operadores são executados, conforme a tabela de precedências da linguagem C.

Exemplos de expressões:

```
Anos=Dias/365.25;
i = i + 3;
c= a *b + d/e;
c= a*(b + d)/e;
```

## Conversão de tipos em expressões

Quando o C avalia expressões onde temos variáveis de tipos diferentes o compilador verifica se as conversões são possíveis. Se não são, ele não compilará o programa, dando uma mensagem de erro. Se as conversões forem possíveis ele as faz, seguindo as regras abaixo:

1. Todos os chars e short ints são convertidos para ints. Todos os floats são convertidos para doubles.

2. Para pares de operandos de tipos diferentes: se um deles é long double o outro é convertido para long double; se um deles é double o outro é convertido para double; se um é long o outro é convertido para long; se um é unsigned o outro é convertido para unsigned.

## Expressões que Podem ser Abreviadas

O C admite as seguintes equivalências, que podem ser usadas para simplificar expressões ou para facilitar o entendimento de um programa:

Expressão Original	Expressão Equivalente
<code>x=x+k;</code>	<code>x+=k;</code>
<code>x=x-k;</code>	<code>x-=k;</code>
<code>x=x*k;</code>	<code>x*=k;</code>
<code>x=x/k;</code>	<code>x/=k;</code>
<code>x=x&gt;&gt;k;</code>	<code>x&gt;&gt;=k;</code>
<code>x=x&lt;&lt;k;</code>	<code>x&lt;&lt;=k;</code>
<code>x=x&amp;k;</code>	<code>x&amp;=k;</code>
etc...	

## Encadeando expressões: o operador “,”

O operador “,” determina uma lista de expressões que devem ser executadas sequencialmente. Em síntese, a vírgula diz ao compilador: execute as duas expressões separadas pela vírgula, em sequência. O valor retornado por uma expressão com o operador “,” é sempre dado pela expressão mais à direita. No exemplo abaixo:

```
x=(y=2,y+3);
```

O valor 2 vai ser atribuído a y, se somará 3 a y e o retorno (5) será atribuído à variável x. Pode-se encadear quantos operadores “,” forem necessários.

O exemplo a seguir mostra outro uso para o operador “,” dentro de um for:

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int x, y;
    for (x=0 , y=0 ; x+y < 100 ; ++x , y++)
```

```

/* Duas variáveis de controle: x e y */
printf("\n%d ", x+y);
/* o programa imprimirá os números pares de 2 a 98 */
getch();
}

```

Foi atribuído o valor zero a cada uma delas na inicialização do for e ambas são incrementadas na parte de incremento do for.

## Tabela de Precedências do C

Esta é a tabela de precedência dos operadores em C. Alguns (poucos) operadores ainda não foram estudados, e serão apresentados em aulas posteriores.

Maior precedência

Operador	Associatividade
() []	da esquerda para a direita
! ++ --	da direita para a esquerda
* / %	da esquerda para a direita
+ -	da esquerda para a direita
< <= > >=	da esquerda para a direita
== !=	da esquerda para a direita
&&	da esquerda para a direita
	da esquerda para a direita

Dica: Considerar a regra geral:

Operações aritméticas  
Operações relacionais  
Operações lógicas

Uma dica aos iniciantes: Você não precisa saber toda a tabela de precedências de cor. É útil que você conheça as principais relações, mas é aconselhável que ao escrever o seu código, você tente isolar as expressões com parênteses, para tornar o seu programa mais legível.

## Modeladores (Casts)

Um modelador é aplicado a uma expressão. Ele força a mesma a ser de um tipo especificado. Sua forma geral é:

(tipo)expressão

Um exemplo:

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    int num;
    float f;

    num=10;
    f=(float)num/7;
    /* Uso do modelador .
    Força a transformação de num em um float */
    printf ("%f",f);
    getch( );
}
```

Se não tivéssemos usado o modelador no exemplo acima o C faria uma divisão inteira entre 10 e 7. O resultado seria 1 (um) e este seria depois convertido para float mas continuaria a ser 1.0. Com o modelador temos o resultado correto.

## ESTRUTURAS DE CONTROLE DE FLUXO

As estruturas de controle de fluxo são fundamentais para qualquer linguagem de programação. Sem elas só haveria uma maneira do programa ser executado: de cima para baixo comando por comando. Não haveria condições, repetições ou saltos. A linguagem C possui diversos comandos de controle de fluxo. É possível resolver todos os problemas sem utilizar todas elas, mas devemos nos lembrar que a elegância e facilidade de entendimento de um programa dependem do uso correto das estruturas no local certo.



# TOMADA DE DECISÃO

Os comandos de controle de fluxo são aqueles que permitem ao programador alterar a sequência de execução do programa. Vamos dar uma breve introdução a dois comandos de controle de fluxo. Outros comandos serão estudados posteriormente.

## COMANDO if

O comando if representa uma tomada de decisão do tipo "SE isto ENTÃO aquilo". A sua forma geral é:

```
if (condição) declaração;
```

A condição do comando if é uma expressão que será avaliada. Se o resultado for zero a declaração não será executada. Se o resultado for qualquer coisa diferente de zero a declaração será executada. A declaração pode ser um bloco de código ou apenas um comando. É interessante notar que, no caso da declaração ser um bloco de código, não é necessário (e nem permitido) o uso do ";" no final do bloco. Isto é uma regra geral para blocos de código. Abaixo apresentamos um exemplo:

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    if (num>10)
        printf ("\n\nO numero e maior que 10");

    if (num==10)
    {
        printf ("\n\nVoce acertou!\n");
        printf ("O numero e igual a 10.");
    }
    if (num<10)
        printf ("\n\nO numero e menor que 10");
    getch();
}
```

No programa acima a expressão `num>10` é avaliada e retorna um valor diferente de zero, se verdadeira, e zero, se falsa. No exemplo, se `núm.` for maior que 10, será impressa a frase: "O número é maior que 10". Repare que, se o número for igual a 10, estamos executando dois comandos. Para que isto fosse possível, tivemos que agrupá-los em um bloco que se inicia logo após a comparação e termina após o segundo `printf`. Repare também que quando queremos testar igualdades usamos o operador `==` e não `=`. Isto porque o operador `=` representa apenas uma atribuição. Pode parecer estranho à primeira vista, mas se escrevêssemos o compilador iria atribuir o valor 10 à variável `num` e a expressão `num=10` iria retornar 10, fazendo com que o nosso valor de `núm.` fosse modificado e fazendo com que a declaração fosse executada sempre. Este problema gera erros frequentes entre iniciantes e, portanto, muita atenção deve ser tomada.

```
if (num=10) ... /* Isto esta errado */
```

Os operadores de comparação são:

`==` (igual)

`!=` (diferente de)

`>` (maior que)

`<` (menor que)

`>=` (maior ou igual)

`<=` (menor ou igual).

A expressão, na condição, será avaliada. Se ela for zero, a declaração não será executada. Se a condição for diferente de zero a declaração será executada.

## COMANDO `else`

Podemos pensar no comando `else` como sendo um complemento do comando `if`. O comando `if` completo tem a seguinte forma geral:

```
if (condição)
    declaração_1;
else
    declaração_2;
```

A expressão da condição será avaliada. Se ela for diferente de zero a declaração 1 será executada. Se for zero a declaração 2 será executada. É importante nunca esquecer que, quando usamos a estrutura `if-else`, estamos garantindo que uma das duas declarações será

executada. Nunca serão executadas as duas ou nenhuma delas. Abaixo está um exemplo do uso do if- else que deve funcionar como o programa da seção anterior.

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

int main ()
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d", &num);

    if (num==10)
    {
        printf ("\n\nVoce acertou!\n");
        printf ("O numero e igual a 10.\n");
    }
    else
    {
        printf ("\n\nVoce errou!\n");
        printf ("O numero e diferente de 10.\n");
    }
    getch();
}
```

## COMANDO if-else-if

A estrutura if-else-if é apenas uma extensão da estrutura if-else. Sua forma geral pode ser escrita como sendo

```
if (condição_1)
    declaração_1;
else if (condição_2)
    declaração_2;
else if (condição_3)
    declaração_3;
.
.
.
else if (condição_n)
    declaração_n;
else
    declaração_default;
```

Testar as condições começando pela 1 e continua a testar até que ele ache uma expressão cujo resultado dê diferente de zero. Neste caso ele executa a declaração correspondente. Só uma declaração será executada, ou seja, só será executada a declaração equivalente à primeira condição que der diferente de zero. A última declaração (default) é a que será executada no caso de todas as condições darem zero e é opcional.

Um exemplo da estrutura acima:

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

int main ()
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d", &num);
    if (num>10)
        printf ("\n\nO numero e maior que 10");
    else if (num==10)
    {
        printf ("\n\nVoce acertou!\n");
        printf ("O numero e igual a 10.");
    }
    else if (num<10)
        printf ("\n\nO numero e menor que 10");
    getch();
}
```

## A expressão condicional

Quando o compilador avalia uma condição, ele quer um valor de retorno para poder tomar a decisão. Mas esta expressão não necessita ser uma expressão no sentido convencional. Uma variável sozinha pode ser uma "expressão" e está retorna o seu próprio valor. Isto quer dizer que teremos as seguintes expressões:

```
Equivalem a
int num;
if (num!=0) ....
if (num==0) ....
for (i = 0; string[i] != '\0'; i++)

int num;
if (num) ....
```

```
if (!num) ....  
for (i = 0; string[i]; i++)
```

Isto quer dizer que podemos simplificar algumas expressões simples.

## COMANDO ifs aninhados

O if aninhado é simplesmente um if dentro da declaração de um outro if externo. O único cuidado que devemos ter é o de saber exatamente a qual if um determinado else está ligado. Vejamos um exemplo:

```
#include <stdio.h>  
#include <conio.h>  
int main ()  
{  
    int num;  
    printf ("Digite um numero: ");  
    scanf ("%d",&num);  
    if (num==10)  
    {  
        printf ("\n\nVoce acertou!\n");  
        printf ("O numero e igual a 10.\n");  
    }  
    else  
    {  
        if (num>10)  
        {  
            printf ("O numero e maior que 10.");  
        }  
        else  
        {  
            printf ("O numero e menor que 10.");  
        }  
    }  
    getch( );  
}
```

## COMANDO Operador “?”

Uma expressão como:

```
if (a>0)
    b=-150;
else
    b=150;
```

Pode ser simplificado usando-se o operador “?” da seguinte maneira:

```
b=a>0?-150:150;
```

De uma maneira geral a expressão do tipo:

```
if (condição)
    expressão_1;
else
    expressão_2;
```

podem ser substituída por:

```
condição?expressão_1:expressão_2;
```

O operador “?” é limitado (não atende a uma gama muito grande de casos) mas pode ser usado para simplificar expressões complicadas. Uma aplicação interessante é a do contador circular. Veja o exemplo:

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int index = 0, contador;
    char letras[5] = "Joao";
    for (contador=0; contador < 1000; contador++)
    {
        printf("\n%c",letras[index]);
        (index==3) ? index=0: ++index;
    }

    getch();
}
```

O nome Joao é escrito na tela verticalmente até a variável contador determinar o término do programa. Enquanto isto a variável index assume os valores 0, 1, 2, 3, , 0, 1, ... progressivamente.

## EXERCÍCIOS

- Ler 2 números inteiros e soma-los. Se a soma for maior que 10, mostrar o resultado da soma.
- Ler um número e imprimir: maior que 20, igual a 20 ou menor que 20.
- Receber um número do teclado e informar se ele é divisível por 10, por 5, por 2 ou se não é divisível por nenhum destes.
- Altere o último exemplo da página 35 para que ele escreva cada letra 5 vezes seguidas. Para isto, use um 'if' para testar se o contador é divisível por cinco (utilize o operador %) e só então realizar a atualização em index.

## COMANDO switch

O comando if-else e o comando switch são os dois comandos de tomada de decisão. Sem dúvida alguma o mais importante dos dois é o if, mas o comando switch tem aplicações valiosas. Mais uma vez vale lembrar que devemos usar o comando certo no local certo. Isto assegura um código limpo e de fácil entendimento. O comando switch é próprio para se testar uma Variável em relação a diversos valores pré-estabelecidos. Sua forma geral é:

```
switch
(variável)
{
    case constante_1:
        declaração_1;
        break;
    case constante_2:
        declaração_2;
        break;
    .
    .
    .
```

```

    case constante_n:
        declaração_n;
        break;
    default:
        declaração_default;
}

```

Podemos fazer uma analogia entre o switch e a estrutura if-else-if apresentada anteriormente. A diferença fundamental é que a estrutura switch não aceita expressões. Aceita apenas constantes. O switch testa a variável e executa a declaração cujo case corresponda ao valor atual da variável. A declaração default é opcional e será executada apenas se a variável, que está sendo testada, não for igual a nenhuma das constantes.

O comando break, faz com que o switch seja interrompido assim que uma das declarações seja executada. Mas ele não é essencial ao comando switch. Se após a execução da declaração não houver um break, o programa continuará executando. Isto pode ser útil em algumas situações, mas eu recomendo cuidado.

Veremos agora um exemplo do comando switch:

```

#include <stdio.h>
#include <conio.h>

int main ()
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    switch (num)
    {
        case 9:
            printf ("\n\nO numero e igual a 9.\n");
            break;
        case 10:
            printf ("\n\nO numero e igual a 10.\n");
            break;
        case 11:
            printf ("\n\nO numero e igual a 11.\n");
            break;
        default:
            printf ("\n\nO numero nao e nem 9 nem 10 nem 11.\n");
    }
    getch();
}

```



## EXERCÍCIOS

- a) Escreva um programa que pede para o usuário entrar um número correspondente a um dia da semana e que então apresente na tela o nome do dia. utilizando o comando switch.

## LAÇOS DE REPETIÇÃO

### COMANDO for

O for é a primeira de uma série de três estruturas para se trabalhar com loops de repetição. As outras são while e do. As três compõem a segunda família de comandos de controle de fluxo. Podemos pensar nesta família como sendo a das estruturas de repetição controlada.

Como já foi dito, o loop for é usado para repetir um comando, ou bloco de comandos, diversas vezes, de maneira que se possa ter um bom controle sobre o loop. Sua forma geral é:

```
for (inicialização; condição; incremento)
{
    declaração;
}
```

O melhor modo de se entender o loop for é ver como ele funciona "por dentro". O loop for é equivalente a se fazer o seguinte:

```
inicialização;
if (condição)
{
    declaração;
    incremento;
    "Volte para o comando if"
}
```

Podemos ver, então, que o for executa a inicialização incondicionalmente e testa a condição. Se a condição for falsa ele não faz mais nada. Se a condição for verdadeira ele executa a declaração, faz o incremento e volta a testar a condição. Ele fica repetindo estas operações até que a condição seja falsa. Um ponto importante é que podemos omitir qualquer um dos elementos do for, isto é, se não quisermos uma inicialização poderemos omiti-la. Abaixo vemos um programa que coloca os primeiros 100 números inteiros na tela:

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

```

int main ()
{
    int count;
    for (count=1; count<=100; count++)
    {
        printf ("%d ",count);
    }
    getch( );
}

```

Note que, no exemplo acima, há uma diferença em relação ao exemplo anterior. O incremento da variável count é feito usando o operador de incremento que nós agora já conhecemos. Esta é a forma usual de se fazer o incremento (ou decremento) em um loop for.

O for na linguagem C é bastante flexível. Temos acesso à inicialização, à condição e ao incremento. Qualquer uma destas partes do for pode ser uma expressão qualquer do C, desde que ela seja válida. Isto nos permite fazer o que quisermos com o comando. As três formas do for abaixo são válidas:

```

for ( count = 1; count < 100 ; count++)
{
    ...
}
for (count = 1; count < NUMERO_DE_ELEMENTOS ; count++)
{
    ...
}
for (count = 1; count < BusqueNumeroDeElementos() ; count+=2)
{
    ...
}
etc ...

```

Preste atenção ao último exemplo: o incremento está sendo feito de dois em dois. Além disto, no teste está sendo utilizada uma função (BusqueNumeroDeElementos() ) que retorna um valor que está sendo comparado com count.

## O loop infinito

O loop infinito tem a forma

```
for (inicialização; incremento) declaração;
```

Este loop chama-se loop infinito porque será executado para sempre (não existindo a condição, ela será sempre considerada verdadeira), a não ser que ele seja interrompido. Para interromper um loop como este usamos o comando break. O comando break vai quebrar o loop infinito e o programa continuará sua execução normalmente.

Como exemplo vamos ver um programa que faz a leitura de uma tecla e sua impressão na tela, até que o usuário aperte uma tecla sinalizadora de final (um FLAG). O nosso FLAG será a letra 'X'. Repare que tivemos que usar dois scanf() dentro do for. Uma busca o caractere que foi digitado e o outro busca o outro caracter digitado na sequência, que é o caractere correspondente ao <ENTER>.

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    int Count;
    char ch;
    printf(" Digite uma letra OU (X) para sair >> ");
    for (Count=1;;Count++)
    {
        scanf("%c", &ch);
        if (ch == 'X')
            break;
        printf("\nLetra: %c n",ch);
        scanf("%c", &ch);
    }
    getch( );
}
```

## O loop sem conteúdo

Loop sem conteúdo é aquele no qual se omite a declaração. Sua forma geral é (atenção ao ponto e vírgula!):

```
for (inicialização;condição;incremento);
```

Uma das aplicações desta estrutura é gerar tempos de espera.

O programa faz isto.

```
#include <conio.h>
#include <stdio.h>
int main ()
{
    long int i;
    /* Imprime o caracter de alerta (um beep) */
    printf("\a");

    /* Espera 10.000.000 de iteracoes */
    for (i=0; i<10000000; i++);

    /* Imprime outro caracter de alerta */
    printf("\a");

    getch();
}
```

## O loop para strings

O loop para percorrer uma string tem a seguinte forma:

for (inicialização ; condição ; incremento) declaração;

Um exemplo interessante é mostrado a seguir: o programa lê uma string e conta quantos dos caracteres desta string são iguais à letra 'c'

```
#include <conio.h>
#include <stdio.h>
int main ()
{
    char string[100]; /* String, ate' 99 caracteres */
    int i, cont;
    printf("\n\nDigite uma frase: ");
    gets(string); /* Le a string */
    printf("\n\nFrase digitada:\n%s", string);
    cont = 0;
    for (i=0; string[i] != '\0'; i=i+1)
    {
```

```

        if ( string[i] == 'c' ) /* Se for a letra 'c' */
            /* Incrementa o contador de caracteres */
            cont = cont +1;

    }

    printf("\nNumero de caracteres c = %d", cont);
    getch();
}

```

Note o teste que está sendo feito no for: o caractere armazenado em `string[i]` é comparado com `'\0'` (caractere final da string). Caso o caractere seja diferente de `'\0'`, a condição é verdadeira e o bloco do for é executado. Dentro do bloco existe um if que testa se o caractere é igual a `'c'`. Caso seja, o contador de caracteres `c` é incrementado. Mais um exemplo, agora envolvendo caracteres:

```

/* Este programa imprime o alfabeto: letras maiúsculas */
#include <stdio.h>
#include <conio.h>
int main()
{
    char letra;
    for (letra = 'A' ; letra <= 'Z' ; letra =letra+1)
        printf("%c ", letra);
    getch();
}

```

Este programa funciona porque as letras maiúsculas de A a Z possuem código inteiro sequencial.

## EXERCÍCIOS

- Escreva um programa que coloque os números de 1 a 100 na tela na ordem inversa (começando em 100 e terminando em 1).
- Escreva um programa que leia uma string, conte quantos caracteres desta string são iguais a 'a' e substitua os que forem iguais a 'a' por 'b'. O programa deve imprimir o número de caracteres modificados e a string modificada.

## O comando While

O comando while tem a seguinte forma geral:

`while (condição) declaração;`

Assim como fazemos para o comando `for`, vamos tentar mostrar como o `while` funciona fazendo uma analogia. Então o `while` seria equivalente a:

```
if (condição)
{
    declaração;
    "Volte para o comando if"
}
```

Podemos ver que a estrutura `while` testa uma condição. Se esta for verdadeira a declaração é executada e faz-se o teste novamente, e assim por diante. Assim como no caso do `for`, podemos fazer um loop infinito. Para tanto basta colocar uma expressão eternamente verdadeira na condição. Pode-se também omitir a declaração e fazer um loop sem conteúdo. Vamos ver um exemplo do uso do `while`. O programa abaixo é executado enquanto `i` for menor que 100. Veja que ele seria implementado mais naturalmente com um `for` ...

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    int i = 0;
    while ( i < 100)
    {
        printf(" %d", i); i++;
    }
    getch();
}
```

O programa abaixo espera o usuário digitar a tecla 'q' e só depois finaliza:

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    char Ch;
    Ch='\0';
    while (Ch!='q')
    {
        scanf("%c", &Ch);
    }
    getch();
}
```

## O comando While

A terceira estrutura de repetição que veremos é o do-while de forma geral:

```
do
{
    declaração;
} while (condição);
```

Mesmo que a declaração seja apenas um comando é uma boa prática deixar as chaves. O ponto e vírgula final é obrigatório. Vamos, como anteriormente, ver o funcionamento da estrutura do-while "por dentro":

```
declaração;
if (condição)
    "Volta para a declaração"
```

Vemos pela análise do bloco acima que a estrutura do-while executa a declaração, testa a condição e, se esta for verdadeira, volta para a declaração. A grande novidade no comando do-while é que ele, ao contrário do for e do while, garante que a declaração será executada pelo menos uma vez. Um dos usos da estrutura do-while é em menus, nos quais você quer garantir que o valor digitado pelo usuário seja válido, conforme apresentado abaixo:

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    int i;
    do
    {
        printf ("\n\nEscolha a fruta pelo numero:\n\n");
        printf ("\t(1) ...Mamão\n");
        printf ("\t(2) ...Abacaxi\n");
        printf ("\t(3) ...Laranja\n\n");
        scanf("%d", &i);

        } while ((i<1) || (i>3));

    switch (i)
    {
        case 1:
            printf ("\t\tVoce escolheu Mamão.\n");
            break;
        case 2:
```

```
        printf ("\t\tVoce escolheu Abacaxi.\n");
        break;
    case 3:
        printf ("\t\tVoce escolheu Laranja.\n");
        break;
    }
    getch( );
}
```

## EXERCÍCIOS

- a) Receber um nome e imprimir as letras na posição impar
- b) Escrever seu nome na tela 10 vezes. Um nome por linha.
- c) Receber do teclado um nome e imprimir tantas vezes quantos forem seus Caracteres.
- d) Solicitar a idade de várias pessoas e imprimir: Total de pessoas com menos de 21 anos. Total de pessoas com mais de 50 anos. O programa termina quando idade for ==-99
- e) Fazer um programa que receba um valor n no teclado e determine o maior. A condição de término do programa é quando o usuário digitar zero.
- f) Faça um programa que inverta uma string: leia a string com gets e armazene-a invertida em outra string. Use o comando for para varrer a string até o seu final.
- g) Solicitar um nome e escrevê-lo de trás pra frente

## COMANDO break

Nós já vimos dois usos para o comando break: interrompendo os comandos switch e for. Na verdade, estes são os dois usos do comando break: ele pode quebrar a execução de



um comando (como no caso do switch) ou interromper a execução de qualquer loop (como no caso do for, do while ou do do while). O break faz com que a execução do programa continue na primeira linha seguinte ao loop ou bloco que está sendo interrompido. Observe que um break causará uma saída somente do laço mais interno. Por exemplo:

```
for(t=0; t<100; ++t)
{
    count=1;
    for(;;)
    {
        printf("%d", count);
        count++;
        if(count==10)
            break;
    }
}
```

O código acima imprimirá os números de 1 a 10 cem vezes na tela. Toda vez que o break é encontrado, o controle é devolvido para o laço for externo.

Outra observação é o fato que um break usado dentro de uma declaração switch afetará somente os dados relacionados com o switch e não qualquer outro laço em que o switch estiver.

## COMANDO continue

O comando continue pode ser visto como sendo o oposto do break. Ele só funciona dentro de um loop. Quando o comando continue é encontrado, o loop pula para a próxima iteração, sem o abandono do loop, ao contrário do que acontecia no comando break. O programa abaixo exemplifica o uso do continue:

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

    while (opcao != 5)
    {
        printf("\n\n Escolha uma opcao entre 1 e 5: ");
        scanf("%d", &opcao);
        if ((opcao > 5) || (opcao < 1))
```

```

        continue; /* Opcao invalida: volta ao inicio do loop */
switch (opcao)
{
case 1:
    printf("\n --> Primeira opcao..");
    break;
case 2:
    printf("\n --> Segunda opcao..");
    break;
case 3:
    printf("\n --> Terceira opcao..");
    break;
case 4:
    printf("\n --> Quarta opcao..");
    break;
case 5:
    break;
    printf("\n --> Abandonando..");
}
}
}

```

O programa acima ilustra uma aplicação simples para o continue. Ele recebe uma opção do usuário. Se esta opção for inválida, o continue faz com que o fluxo seja desviado de volta ao início do loop. Caso a opção escolhida seja válida o programa segue normalmente.

## O Comando goto

Vamos mencionar o goto apenas para que você saiba que ele existe. O goto é o último comando de controle de fluxo. Ele pertence a uma classe à parte: a dos comandos de salto incondicional. O goto realiza um salto para um local especificado. Este local é determinado por um rótulo. Um rótulo, na linguagem C, é uma marca no programa. Você dá o nome que quiser a esta marca. Podemos tentar escrever uma forma geral:

nome\_do\_rótulo:

....

goto nome\_do\_rótulo;

....

Devemos declarar o nome do rótulo na posição para a qual vamos dar o salto seguido de: O goto pode saltar para um rótulo que esteja mais à frente ou para trás no programa. Uma

observação importante é que o rótulo e o goto devem estar dentro da mesma função. Como exemplo do uso do goto vamos reescrever o equivalente ao comando for apresentado na seção equivalente ao mesmo:

inicialização;

```
início_do_loop: if (condição)
{
    declaração;
    incremento;
    goto início_do_loop;
}
```

O comando goto deve ser utilizado com parcimônia, pois o abuso no seu uso tende a tornar o código confuso. O goto não é um comando necessário, podendo sempre ser substituído por outras estruturas de controle. Recomendamos que o goto nunca seja usado.

Existem algumas situações muito específicas onde o comando goto pode tornar um código mais fácil de entender se ele for bem empregado. Um caso em que ele pode ser útil é quando temos vários loops e ifs aninhados e se queira, por algum motivo, sair destes loops e ifs todos de uma vez. Neste caso um goto resolve o problema mais elegantemente que vários breaks, sem contar que os breaks exigiriam muito mais testes. Ou seja, neste caso o goto é mais elegante e mais rápido.

O exemplo da página anterior pode ser reescrito usando-se o goto:

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int opcao;
    while (opcao != 5)
    {
        REFAZ:      printf("\n\n Escolha uma opcao entre 1 e5: ");
                   scanf("%d", &opcao);
        if ((opcao > 5) || (opcao <1))
            goto REFAZ;
        /* Opcao invalida: volta ao rotulo REFAZ */
        switch (opcao)
        {
            case 1:
                printf("\n --> Primeira opcao..");
                break;
            case 2:
                printf("\n --> Segunda opcao..");
```

```
        break;
    case 3:
        printf("\n --> Terceira opcao..");
        break;
    case 4:
        printf("\n --> Quarta opcao..");
        break;
    case 5:
        printf("\n -->Abandonando..");
        break;
    }
}
getch( );
}
```

## EXERCÍCIOS

a) Escreva um programa para verificar uma data:

**Parte 1:** peça três inteiros, correspondentes a dia, mês e ano. Peça os números até conseguir valores que estejam na faixa correta (dias entre 1 e 31, mês entre 1 e 12 e ano entre 1900 e 2100).

**Parte 2:** Verifique se o mês e o número de dias batem (incluindo verificação de anos bissextos).

**Parte 3:** Se estiver tudo certo imprima o número que aquele dia corresponde no ano.

Comente seu programa.

Obs: Um ano é bissexto se for divisível por 4 e não for divisível por 100, exceto para os anos divisíveis por 400, que também são bissextos