

2016.1

Introdução a Linguagem C

Linguagem de Programação Estruturada

Histórico da Linguagem C

- Criada por Denis Ritchie, na década de 1970, para uso em um computador DEC PDP-11 em Unix;
- C é derivado de uma outra linguagem: o B, criado por Ken Thompson. O B, por sua vez, veio da linguagem BCPL, inventada por Martin Richards;
- O sistema Unix é escrito em C e C++;
- C++ é uma extensão da linguagem C.

Estrutura básica de um programa C

diretivas para o pré-processador

declaração de variáveis globais

main ()

{

declaração de variáveis locais da função main

comandos da função main

}

Diretivas para o processador - Bibliotecas

- Diretiva `#include` permite incluir uma biblioteca
- Bibliotecas contêm funções pré-definidas, utilizadas nos programas.

- Exemplos:

<code>#include <stdio.h></code>	Funções de entrada e saída
<code>#include <stdlib.h></code>	Funções padrão
<code>#include <math.h></code>	Funções matemáticas
<code>#include <string.h></code>	Funções de texto

O ambiente Dev-C++

- O Dev-C++ é um ambiente de desenvolvimento de programas em C e C++ (editor, compilador, bibliotecas...)
- Pode ser baixado de <http://www.bloodshed.net/devcpp.html>

Usando o Dev-C++

- Inicie o Dev-C++ pelo ícone ou pelo menu
- Crie um novo arquivo, com o comando Arquivo, Novo, Arquivo Fonte
- Edite o programa abaixo:

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    printf ("Alo mundo!");
    system("PAUSE");
}
```

Usando o Dev-C++

- Salve o programa com o nome exemplo.cpp em um diretório com o seu nome;
- Compile e execute o programa pressionando a tecla F9;
- Se houver algum erro de sintaxe, aparece uma ou mais mensagens no rodapé da janela. Neste caso, corrija o programa e repita.

Dicas

- Termine todos os comandos com ;
- Quando ocorrer um erro de compilação, dê um duplo clique sobre a mensagem de erro para destacar o comando errado no programa
- Verifique também a linha anterior, que pode ser a responsável pelo erro, especialmente se faltar o ;
- Use comentários, iniciados por // ou entre /* */
 - /* isto é um comentário */
 - // isto também é um comentário

Exemplo 1

```
/* meu primeiro programa C */  
#include <stdio.h>  
#include <stdlib.h>  
main()  
{  
    printf ("Alo mundo!"); //mostra  
    system("PAUSE"); //fica parado  
}
```

Declarações

- Declaram as variáveis e seus tipos
- Os nomes das variáveis devem conter apenas letras, dígitos e o símbolo `_` e iniciar com letra ou `_`
- Os principais tipos são: `int`, `float`, `double` e `char`
- Exemplos

```
int n;  
int quantidade_valores;  
float x, y, somaValores;  
char sexo;  
char nome[40];
```

OBS.: C diferencia letras maiúsculas de minúsculas!

```
int n, N;  
n é diferente de N
```

Declarações

Algoritmo

Var

n1, n2, n3, media: real

Na Linguagem C...

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    float n1, n2, n3, media;

    system("PAUSE");
}
```

Comando de atribuição

- Atribui o valor da direita à variável da esquerda
- O valor pode ser:
uma *constante*,
uma *variável* ou
uma *expressão*

- Exemplos

```
x = 4; // lemos: x recebe 4
y = x + 2; // lemos: y recebe (x mais 2)
y = y + 4; // lemos: y recebe (y mais 4)
valor = 2.5;
sexo = 'F' // constantes devem estar entre aspas
simples (apóstrofe)
```

Entrada e Saída de Dados

Entrada de Dados

- **Função scanf**

```
scanf ("formatos", &var1, &var2,...)
```

Exemplos:

```
int i, j;
float x;
char c;
scanf ("%d", &i);
scanf ("%d %f", &j, &x);
scanf ("%c", &c);
scanf ("%s", &nome);
```

Não deixar
espaço em
branco!!!

%d	inteiro decimal
%f	float
%lf	double
%c	char
%s	string

Entrada de Dados (Exemplo 2)

Algoritmo

ler n1

ler n2

ler n3

ler n1, n2, n3

Na Linguagem C...

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

```
#include <stdlib.h>
```

```
main()
```

```
{
```

```
float n1, n2, n3, media;
```

```
scanf ("%f",&n1);
```

```
scanf ("%f",&n2);
```

```
scanf ("%f",&n3);
```

```
scanf ("%f %f %f",&n1, &n2, &n3);
```

```
system("PAUSE");
```

```
}
```

OBS: não deixe espaço antes do fecha "

Operadores Matemáticos

Operador	Exemplo	Comentário
+	$x + y$	Soma x e y
-	$x - y$	Subtrai y de x
*	$x * y$	Multiplica x e y
/	x / y	Divide x por y
%	$x \% y$	Resto da divisão de x por y
++	$x++$	Incrementa em 1 o valor de x
--	$x--$	Decrementa em 1 o valor de x

ATENÇÃO!

- OBS: o operador “/” (divisão) terá um resultado inteiro se os dois operandos forem inteiros. Para um resultado real, um dos dois operandos deve ser real (ou os dois)

Exemplo:

int X,Y;

float Z,U,T;

X=2; Y=3; U=3;

Z=X/Y; // Qual a resposta?

// Z terá o valor zero

T=X/U; //Qual a resposta?

// T terá o valor 0.666667

Entrada de Dados (exemplo 3)

Algoritmo

ler n1, n2, n3
Media $\leftarrow (n1+n2+n3)/3$

Na Linguagem C...

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    float n1, n2, n3, media;
    scanf ("%f %f %f",&n1, &n2, &n3);
    media=(n1+n2+n3)/3;
    system("PAUSE");
}
```

Saída de Dados

- Função **printf**

`printf ("formatos", var1, var2,...)`

Exemplos:

```
int i, j;
float x;
char c;
printf("%d", i);
printf("%d %f", j, x);
printf("%c", c);
```

<code>%d</code>	inteiro
<code>%f</code>	float
<code>%lf</code>	double
<code>%c</code>	char
<code>%s</code>	string

Saída de Dados

- Função **printf**
Conversão/Formato do argumento

%d	Número decimal inteiro (int). Também pode ser usado %i como equivalente a %d.
%u	Número decimal natural (unsigned int), ou seja, sem sinal.
%o	Número inteiro representado na base octal. Exemplo: 41367 (corresponde ao decimal 17143).
%x	Número inteiro representado na base hexadecimal. Exemplo: 42f7 (corresponde ao decimal 17143). Se usarmos %X, as letras serão maiúsculas: 42F7.
%X	Hexadecimal com letras maiúsculas
%f	Número decimal de ponto flutuante. No caso da função printf, devido às conversões implícitas da linguagem C, serve tanto para float como para double. No caso da função scanf, %f serve para float e %lf serve para double.
%e	Número em notação científica, por exemplo 5.97e-12. Podemos usar %E para exibir o E maiúsculo (5.97E-12).
%E	Número em notação científica com o "e" maiúsculo
%g	Escolhe automaticamente o mais apropriado entre %f e %e. Novamente, podemos usar %G para escolher entre %f e %E.
%p	Ponteiro: exibe o endereço de memória do ponteiro em notação hexadecimal.
%c	Caractere: imprime o caractere que tem o código ASCII correspondente ao valor dado.
%s	Sequência de caracteres (string, em inglês).
%%	Imprime um %

Saída de Dados

- Função **printf**

Largura do campo

- Como o próprio nome já diz, especifica qual a largura mínima do campo. Se o valor não ocupar toda a largura do campo, este será preenchido com espaços ou zeros.

Exemplos:

```
printf ("%5d", 15); // exhibe " 15"  
printf ("%05d", 15); // exhibe "00015"  
printf ("% -5d", 15); // exhibe "15  "
```

Saída de Dados

- Função **printf**
Precisão

A precisão pode ter quatro significados diferentes:

- Se a conversão solicitada for **inteira** (d, i, o, u, x, X): o número mínimo de dígitos a exibir (será preenchido com zeros se necessário).

printf ("%%.5d", 314); // exibe "00314"

- Se a conversão for real (a, A, e, E, f, F): o número de casas decimais a exibir. O valor será arredondado se a precisão especificada no formato for menor que a do argumento.

printf ("%%.5f", 2.4); // exibe "2.40000"

- Se a conversão for em notação científica (g, G): o número de algarismos significativos. O valor será arredondado se o número de algarismos significativos pedido for maior que o do argumento.

printf ("%%.5g", 23456789012345); // exibe "2.3457e+13"

- Se a conversão for de uma sequência de caracteres (s): o número máximo de caracteres a exibir. **printf ("%%.5s", "Bom dia"); // exibe "Bom d"**

Saída de Dados (Exemplo 4)

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int i, j;
    float x;
    i = 1;
    j = 2;
    x = 3;
    printf("%d", i);
    printf(" %d %f", j, x);

    system("PAUSE");
}
```

Saída de Dados (Exemplo 5)

```

#include <stdio.h>
#include <stdlib.h>
main()
{
    // definicao de variaveis
    float n1, n2, n3, media;
    // entrada de dados
    scanf ("%f %f %f",&n1, &n2, &n3);
    // operacao
    media=(n1+n2+n3)/3;
    // saida de dados
    printf("%f", n1);
    printf("%f", n2);
    printf("%f", n3);
    printf("%f", media);

    system("PAUSE");
}

```


Saída de Dados (Exemplo 6)

```

#include <stdio.h>
#include <stdlib.h>
main()
{
    // definicao de variaveis
    int i, j;
    float x;

    //entrada de dados
    scanf("%d", &i);
    scanf("%d %f", &j, &x);

    // exibicao de dados
    printf("I= %d\n", i);
    printf("J= %d\nX= %f\n", j, x);

    system("PAUSE");
}

```

Entrada e Saída

Exemplo 7

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    float n1, n2, n3, media;
    scanf ("%f %f %f",&n1, &n2, &n3);
    media=(n1+n2+n3)/3;
    printf ("%f",media);

    system("PAUSE");
}
```

Exemplo 8

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    float n1, n2, n3, media;
    printf("Digite 3 notas: ");
    scanf ("%f %f %f",&n1, &n2, &n3);
    media=(n1+n2+n3)/3;
    printf ("Media: %.2f\n",media);

    system("PAUSE");
}
```

Operadores de Atribuição

Operador	Exemplo	Comentário
=	x = y	Atribui o valor de y a x
+=	x += y	Equivale a $x = x + y$
-=	x -= y	Equivale a $x = x - y$
*=	x *= y	Equivale a $x = x * y$
/=	x /= y	Equivale a $x = x / y$
%=	x %= y	Equivale a $x = x \% y$

Funções Matemáticas

Função	Exemplo	Comentário
<code>ceil</code>	<code>ceil(x)</code>	Arredonda o número real para cima; <code>ceil(3.2)</code> é 4
<code>cos</code>	<code>cos(x)</code>	Cosseno de x (x em radianos)
<code>exp</code>	<code>exp(x)</code>	e elevado à potencia x
<code>fabs</code>	<code>fabs(x)</code>	Valor absoluto de x
<code>floor</code>	<code>floor(x)</code>	Arredonda o número real para baixo; <code>floor(3.2)</code> é 3
<code>log</code>	<code>log(x)</code>	Logaritmo natural de x
<code>log10</code>	<code>log10(x)</code>	Logaritmo decimal de x
<code>pow</code>	<code>pow(x, y)</code>	Calcula x elevado à potência y
<code>sin</code>	<code>sin(x)</code>	Seno de x
<code>sqrt</code>	<code>sqrt(x)</code>	Raiz quadrada de x
<code>tan</code>	<code>tan(x)</code>	Tangente de x

```
#include <math.h>
```

Exemplo 9.a

Construa um programa que tendo como entrada dois pontos quaisquer do plano $P(x_1, y_1)$ e $Q(x_2, y_2)$, imprima a distância entre eles. A fórmula é $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
main()
{
    float x1, y1, x2, y2;
    float distancia;
    printf ("\nDigite o valor de x1: ");
    scanf ("%f",&x1);
    printf ("\nDigite o valor de y1: ");
    scanf ("%f",&y1);
    printf ("\nDigite o valor de x2: ");
    scanf ("%f",&x2);
    printf ("\nDigite o valor de y2: ");
    scanf ("%f",&y2);
    distancia=sqrt(pow((x2-x1),2)+pow((y2-y1),2));
    printf ("\nA distancia entre os pontos P1 e P2 e' %.2f\n", distancia);
    system("PAUSE");
}
```

Exemplo 9.b

Construa um programa que seja capaz de calcular a área de um triângulo, dados os comprimentos de seus lados. Utilize a fórmula abaixo, que dá a área do triângulo cujos os lados têm comprimentos a, b, c :

$$S = \sqrt{p \cdot (p - a) \cdot (p - b) \cdot (p - c)}$$

Onde

$$p = \frac{a + b + c}{2}$$

é o *semiperímetro* do triângulo.

Resposta do Exemplo 9.b.

```

/*Programa que determina a área de um triângulo de lados de comprimentos dados*/
#include <stdio.h>
#include <math.h>
main()
{
float x, y, z, Area, SemiPer;
printf("Digite os comprimentos dos lados do triangulo");
scanf("%f %f %f", &x, &y, &z);
SemiPer = (x + y + z)/2;
Area = sqrt(SemiPer * (SemiPer - x) * (SemiPer - y) * (SemiPer - z));
printf("A area do triangulo de lados %f , %f e %f e' igual a %f\n", x, y, z, Area);
}
    
```

Se este programa for executado com entrada 3, 4 e 5 temos SemiPer = 6 e

$$S = \sqrt{6 \cdot (6 - 3) \cdot (6 - 4) \cdot (6 - 5)} = \sqrt{36}$$

Exemplo 10

Construa um programa que calcule a quantidade de latas de tinta necessárias e o custo para pintar tanques cilíndricos de combustível, onde são fornecidos a altura e o raio desse cilindro.

Sabendo que:

- a lata de tinta custa R\$20,00
- cada lata contém 5 litros
- cada litro de tinta pinta 3 metros quadrados.

e que:

Área do cilindro= 2 vezes a área da base + circunferência da base vezes a altura e que raio e altura são dados de entrada.

Exemplo 10

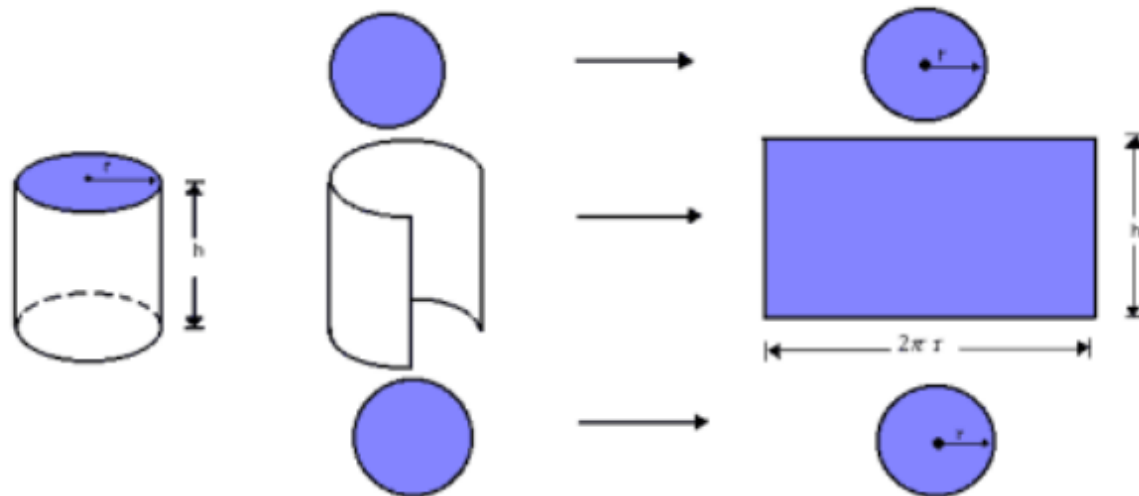
Mais algumas dicas:

Áreas

Num cilindro, consideramos as seguintes áreas:

a) área lateral (A_L)

Podemos observar a área lateral de um cilindro fazendo a sua planificação:



Assim, a área lateral do cilindro reto cuja altura é h e cujos raios dos círculos das bases são r é um retângulo de dimensões $2\pi r$ e h :

$$A_L = 2\pi r h$$

Exemplo 10

Mais algumas dicas:

b) área da base (A_B): área do círculo de raio r

$$A_B = \pi r^2$$

c) área total (A_T): soma da área lateral com as áreas das bases

$$A_T = A_L + 2 A_B = 2\pi r h + 2\pi r^2 = 2\pi r (h + r)$$

Resposta do Exemplo 10

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
main()
{
    float altura, raio, areaCilindro, qtdadeLitros, qtdadeLatas, custo;
    printf ("\nDigite o valor da altura em metros: ");
    scanf ("%f",&altura);
    printf ("\nDigite o valor do raio em metros: ");
    scanf ("%f",&raio);
    areaCilindro=2*3.14*raio*raio + 2*3.14*raio*altura;
    printf ("\nA area do cilindro e' %.2f metros quadrados", areaCilindro);
    qtdadeLitros=areaCilindro/3;
    printf ("\nA qtidade de litros necessaria e' de %.2f ", qtdadeLitros);
    qtdadeLatas=qtdadeLitros/5;
    printf ("\nA qtidade de latas necessaria e' de %.2f ", qtdadeLatas);
    custo=qtdadeLatas*20;
    printf ("\nO valor total das tintas e' de R$ %.2f \n", custo);
    system("PAUSE");
}

```

Revisão de Álgebra das Proposições

ALGEBRA DAS PROPOSIÇÕES

Chamaremos qualquer afirmação verbal que possamos dizer se é **VERDADEIRA** ou **FALSA** (nunca ambas).

Exemplos de proposições:

- O Brasil fica na América do Sul.
- São Paulo tem fronteira com Minas Gerais.
- A capital de Pernambuco é Porto Alegre.
- Manoel é mais alto que Pedro.
- $5 > 3$.

NÃO são Exemplos de proposições:

- Feliz aniversário!
- Até breve!
- Este livro é bom?
- Boa sorte na prova.

As proposições DEVEM
ser sentenças
DECLARATIVAS que
tenham um valor lógico:
VERDADEIRO ou FALSO

ALGEBRA DAS PROPOSIÇÕES

As proposições **simples** são representadas por letras minúsculas (p, q, r, s, etc)

Exemplos:

p: Pedro é médico.

q: $5 < 8$.

r: Lisboa é a capital do Brasil.

$VL(p) = V$

$VL(q) = V$

$VL(r) = F$

PRINCÍPIOS:

- Princípio da identidade

Uma proposição verdadeira é verdadeira; uma proposição falsa é falsa.

- Princípio da **Não-Contradição**

Nenhuma proposição poderá ser verdadeira e falsa ao mesmo tempo.

- Princípio do Terceiro Excluído

Uma proposição ou será verdadeira, ou será falsa: não há outra possibilidade.

ALGEBRA DAS PROPOSIÇÕES

PROPOSIÇÕES SIMPLES OU COMPOSTAS:

Proposições simples

EXEMPLOS:

s: Carla foi ao mercado.

p: Júlio é engenheiro.

t: Carla **não** foi ao mercado.

Proposições Compostas

(As proposições são compostas quando são formadas por **conectivos**).

Os conectivos são	Nome	Símbolo
e	CONJUNÇÃO	\wedge
ou	DISJUNÇÃO	\vee
se... Então ...	CONDICIONAL	\Rightarrow
se e somente se	BICONDICIONAL	\Leftrightarrow
ou ... ou	DISJUNÇÃO EXCLUSIVA	$\underline{\vee}$
NÃO	Negação	\sim ou \neg

ALGEBRA DAS PROPOSIÇÕES

PROPOSIÇÕES SIMPLES OU COMPOSTAS:

Tabela Verdade

A Tabela verdade é um instrumento usado para determinar os valores lógicos das proposições compostas, a partir de atribuições de todos os possíveis valores lógicos das proposições simples componentes.

A primeira das tabelas abaixo apresenta duas proposições simples: p e q e a segunda, três proposições simples: p, q e r.

As células de ambas as tabelas são preenchidas com valores lógicos V e F, de modo a esgotar todas as possíveis combinações. O número de linhas da tabela pode ser previsto efetuando o cálculo: 2 elevado ao número de proposições simples.

p	q
V	V
V	F
F	V
F	F

p	q	r
V	V	V
V	V	F
V	F	V
V	F	F
F	V	V
F	V	F
F	F	V
F	F	F

ALGEBRA DAS PROPOSIÇÕES

VALOR LÓGICO DA PROPOSIÇÃO:

Notação: O valor lógico de uma proposição simples indica-se por $V(p)$ e composta por $V(P)$ (letra maiúscula).

Exemplos de proposições simples:

p : um triângulo têm três lados.

q : Salvador é um país.

$V(p) = V$ $V(q) = F$ (Lê-se valor lógico de p é igual a V (verdadeiro) e de q é igual a F (falso))

Exemplo de proposição composta:

p : o sol é uma estrela ou

q : a terra é uma estrela.

$P(p,q) = p \vee q$ $V(P) = V$ (O símbolo "v" representa o conectivo "ou" visto abaixo)

ALGEBRA DAS PROPOSIÇÕES

p	q	$p \wedge q$
V	V	V
V	F	F
F	V	F
F	F	F

CONJUNÇÃO (\wedge) " $p \wedge q$ " lê-se "p e q".

Nas conjunções, se p é verdadeira e q é verdadeira então $p \wedge q$ será verdadeira, de outro modo $p \wedge q$ será falsa.

Exemplo:

p : A neve é branca (V)

q : $2 < 5$ (V)

$p \wedge q$: A neve é branca e $2 < 5$

Representação:

$$V(p \wedge q) = V(p) \wedge V(q) = V \wedge V = V$$

Leitura:

Valor lógico de (p e q) é igual a ou, de outro modo, valor lógico de (p) e valor lógico de (q) é igual a ou resulta em verdade e verdade que é igual a verdade.

ALGEBRA DAS PROPOSIÇÕESv

CONJUNÇÃO (Outro exemplo):

Sejam as proposições simples:

p: Paris está na França

q: Paris está na Inglaterra

r: $2 + 2 = 5$

s: $2 + 2 = 4$

Qual será o valor verdade das conjunções:

a) $p \wedge s$

c) $q \wedge s$

b) $p \wedge r$

d) $q \wedge r$

p	q	$p \wedge q$
V	V	V
V	F	F
F	V	F
F	F	F

Onde:

V: Verdadeira

F: Falsa

a) $p \wedge s$ -- significa “Paris está na França e $2 + 2 = 4$ ” \therefore Proposição **verdadeira**

b) $p \wedge r$ -- significa “Paris está na França e $2 + 2 = 5$ ” \therefore Proposição **falsa**

c) $q \wedge s$ -- significa “Paris está na Inglaterra e $2 + 2 = 4$ ” \therefore Proposição **falsa**

d) $q \wedge r$ -- significa “Paris está na Inglaterra e $2 + 2 = 5$ ” \therefore Proposição **falsa**

ALGEBRA DAS PROPOSIÇÕES

p	q	$p \vee q$
V	V	V
V	F	V
F	V	V
F	F	F

DISJUNÇÃO: (v) "p v q" lê-se "p ou q".

Duas proposições quaisquer podem ser combinadas pelo conectivo “**ou**” para formar uma nova proposição que é chamada disjunção das duas proposições originais. Designaremos a disjunção de duas proposições p e q por

$p \vee q$

Exemplo:

p : Salvador é a capital de Pernambuco (F)

q : $5 < 7$ (V)

$p \vee q$: Salvador é a capital de Pernambuco
ou $5 < 7$ é menor do que sete (V)

$$V(p \vee q) = V(p) \vee V(q) = F \vee V = V$$

ALGEBRA DAS PROPOSIÇÕES

DISJUNÇÃO (Outro exemplo)

Sejam as proposições simples:

p: Paris está na França

q: Paris está na Inglaterra

r: $2 + 2 = 5$

s: $2 + 2 = 4$

No caso do exemplo acima, qual será o valor das disjunções:

a) $p \vee s$

c) $q \vee s$

b) $p \vee r$

d) $q \vee r$

Concluimos que:

$p \vee s$ É verdadeira

$p \vee r$ É verdadeira

$q \vee s$ É verdadeira

$q \vee r$ É falsa

p	q	$p \vee q$
V	V	V
V	F	V
F	V	V
F	F	F

Onde:

V: Verdadeira

F: Falsa

ALGEBRA DAS PROPOSIÇÕES

DISJUNÇÃO EXCLUSIVA: ($\underline{\vee}$) " $p \underline{\vee} q$ " lê-se "ou p ou q", mas não ambos ou ainda "ou exclusivo".

p	q	$p \underline{\vee} q$
V	V	F
V	F	V
F	V	V
F	F	F

O valor lógico é Falso(F) quando p e q são ambas verdadeiras ou ambas falsas.

Exemplo:

P : Carlos é médico ou professor

Q : Antônio é catarinense ou gaúcho.

Na proposição composta **P** pelo menos uma das proposições simples é verdadeira, podendo ser ambas verdadeiras. ("ou" inclusivo).

Na proposição composta **Q** apenas uma das proposições é verdadeira. ("ou" exclusivo).

ALGEBRA DAS PROPOSIÇÕES

CONDICIONAL: (\rightarrow) " $p \rightarrow q$ " lê-se "se p então q" (" \rightarrow " símbolo de implicação).

p	q	$p \rightarrow q$
V	V	V
V	F	F
F	V	V
F	F	V

O valor lógico é **Falso** (F) no caso em que **p** é verdadeira e **q** é falsa.

Exemplo:

p : A terra é uma estrela (F)

q : O ano tem nove meses (F)

$$V(p \rightarrow q) = V(p) \rightarrow V(q) = F \rightarrow F = V$$

ALGEBRA DAS PROPOSIÇÕES

BICONDICIONAL: (\longleftrightarrow) " $p \longleftrightarrow q$ " lê-se "p se e somente se q".

p	q	$p \longleftrightarrow q$
V	V	V
V	F	F
F	V	F
F	F	V

Uma bicondicional é verdadeira somente quando ambas proposições são verdadeiras ou ambas falsas. (p é condição necessária e suficiente para q ou q é condição necessária e suficiente para p).

Exemplo:

p : A terra é plana (F)

q : 10 é um número primo (F)

$p \longleftrightarrow q$: A terra é plana **se e somente se** 10 for um número primo (V)

$V(p \longleftrightarrow q) = V(p) \longleftrightarrow V(q) = F \longleftrightarrow F = V$

ALGEBRA DAS PROPOSIÇÕES

NEGAÇÃO: (\sim) " $\sim p$ " lê-se "não p".

p	$\sim p$
V	F
F	V

Exemplo:

p : Maria é bonita

$\sim p$: Maria não é bonita

ou $\sim p$: Não é verdade que Maria é bonita

ou $\sim p$: É falso que Maria é bonita

ALGEBRA DAS PROPOSIÇÕES

Construção de tabelas verdade

Procedimento:

Para determinar os valores lógicos de uma proposição composta, deve-se antes relacionar em colunas as proposições simples envolvidas e dar a elas todos os valores lógicos combinados.

Exemplo1:

Construir a tabela verdade da seguinte proposição: $P(p,q) = \sim(p \wedge \sim q)$.

p	q	$\sim q$	$p \wedge \sim q$	$\sim(p \wedge \sim q)$
V	V	F	F	V
V	F	V	V	F
F	V	F	F	V
F	F	V	F	V

ALGEBRA DAS PROPOSIÇÕES

Ordem de precedência dos conectivos

A precedência é o critério que especifica a ordem de avaliação dos conectivos ou operadores lógicos de uma expressão qualquer.

A lógica matemática prioriza as operações de acordo com a ordem listadas abaixo.

1) \sim 2) \wedge e \vee 3) \longrightarrow 4) \longleftrightarrow .

Parênteses podem ser utilizados para determinar uma forma específica de avaliação de uma proposição.

A proposição $p \longrightarrow q \longleftrightarrow s \wedge r$, por exemplo, é bicondicional e nunca uma condicional ou uma conjunção. Para convertê-la numa condicional deve-se usar parênteses: $p \longrightarrow (q \longleftrightarrow s \wedge r)$.

ALGEBRA DAS PROPOSIÇÕES

Tautologia, contradição e contingência

- **Tautologia**

proposição **composta** cuja última coluna de sua tabela verdade encerra somente a letra V(verdade). Exemplo: $p \vee \sim(p \wedge q)$.

- **Contradição**

proposição **composta** cuja última coluna de sua tabela verdade encerra somente a letra F(falsidade). Exemplo: $(p \wedge q) \wedge \sim(p \vee q)$.

- **Contingência**

proposição **composta** cuja última coluna de sua tabela verdade figuram as letras V e F cada uma pelo menos uma vez. Exemplo: $p \vee q \rightarrow p$.

ALGEBRA DAS PROPOSIÇÕES

Equivalência

Uma proposição $P(p, q, r, \dots)$ é equivalente a uma proposição $Q(p, q, r, \dots)$ se as tabelas verdade dessas duas proposições são idênticas.

Notação: $P(p, q, r, \dots) \iff Q(p, q, r, \dots)$.

Exemplo: A condicional " $p \rightarrow q$ " e a disjunção " $\sim p \vee q$ " são equivalentes como expõe sua tabela verdade:

p	q	$p \rightarrow q$	$\sim p$	$\sim p \vee q$
V	V	V	F	V
V	F	F	F	F
F	V	V	V	V
F	F	V	V	V

Equivalência: $p \rightarrow q \iff \sim p \vee q$

Estrutura de Decisão

A instrução "if"

A instrução "if" é a estrutura de teste mais básica, ela é encontrada em todas as linguagens (com sintaxes diferentes...). Ela executa uma série de instruções caso uma condição se realize.

A sintaxe desta expressão é a seguinte:

```
if (condição realizada) {  
    lista de instruções;  
}
```

Observações:

A condição deve ficar entre parênteses

É possível definir várias condições a serem cumpridas com os operadores **E** **OU** (&& e ||)

Por exemplo, a seguinte instrução testa se as duas condições são verdadeiras:

```
if ((condição1)&&(condição2))
```

A instrução if ... else

A instrução *if* em sua forma básica só testa uma condição ou, na maioria das vezes, gostaríamos de poder escolher as instruções a serem executadas **em caso do não cumprimento da condição...**

A expressão *if ... else* executa outra série de instruções, em caso do não cumprimento da condição.

A sintaxe desta expressão é a seguinte:

```
if (condição realizada) {
    lista de instruções
}
else {
    outra série de instruções
}
```


A instrução if ... else

Uma maneira mais rápida de fazer o teste

É possível fazer um teste com uma estrutura muito mais leve, graças à seguinte estrutura:

(condição) ? instrução se verdadeira : instrução se falsa

Observações:

- A condição deve ficar entre parênteses
- Quando a condição for verdadeira, a Instrução da esquerda será executada
- Quando a condição for falsa, a instrução da direita será executada

Assim, esta forma forma: é frequentemente utilizada como segue:

posição = ((Antes == 1) ? contador+1 : contador-1);

A instrução switch

A Instrução *switch* efetua vários testes de valores sobre o conteúdo de uma mesma variável. Esta conexão condicional simplifica muito o teste de vários valores de uma variável, pois esta operação teria sido complicada (mas possível) com *ifs aninhados*.

Sua sintaxe é a seguinte:

```
switch (variável) {  
  
    case Valor1 :  
        Lista d instruções;  
  
        break;  
  
    case Valor2 :  
        Lista d instruções;  
  
        break;  
  
    case Valores... :  
        Lista d instruções;  
  
        break;  
  
    default:  
  
        Lista d instruções;  
  
}
```

A instrução switch

A Instrução *switch* efetua vários testes de valores sobre o conteúdo de uma mesma variável. Esta conexão condicional simplifica muito o teste de vários valores de uma variável, pois esta operação teria sido complicada (mas possível) com *ifs aninhados*.

Sua sintaxe é a seguinte:

```
switch(variável)
{
case 1:
case 2:
{ instruções executadas para a variável = 1 ou para a
variável = 2 }
break;
case 3:
{ instruções executadas para a variável = 3 apenas }
break;
default:
{ instruções executadas para qualquer outro valor de
variável }
}
```

Loops

Os loops são estruturas que executam várias vezes a mesma série de instruções até que uma condição não seja mais realizada...

O loop for

A instrução for executa várias vezes a mesma série de instruções: é um loop!

A sintaxe desta expressão é a seguinte:

```
for (contador; condição; modificação do contador) {  
    lista de instruções;  
}
```

Por exemplo:

```
for (i=1; i<6; i++) {  
    printf("%d", i);  
}
```

OBS.: É preciso contar o número de vezes que você deseja executar o loop:

- `for(i=0;i<10;i++)` executa 10 vezes o loop (i de 0 a 9)
- `for(i=0;i<=10;i++)` executa 11 vezes o loop (i de 0 a 10)
- `for(i=1;i<10;i++)` executa 9 vezes o loop (i de 1 a 9)

Loops

while e do-while: aplicações em C

Em **C**, o enquanto é substituído por **while** e o faça-enquanto por **do-while**

```
while ( condição ) {
    comando_1;
    comando_2;
    . . .
    comando_n;
}
```

```
do {
    comando_1;
    comando_2;
    . . .
    comando_n;
} while ( condição );
```

Loops

A instrução While

A instrução While representa outro meio de executar várias vezes a mesma série de instruções.

A sintaxe desta expressão é a seguinte:

```
while (condição realizada) {  
    lista de instruções;  
}
```

Loops

Pulo incondicional

A sintaxe desta expressão é " *continue*" (esta instrução é colocada em um loop!), e é geralmente associada a uma estrutura condicional, caso contrário, as linhas entre esta instrução e o final do loop seriam obsoletas.

```

x = 1;

while (x<=10) {
    if (x == 7) {
        printf("Division par 0");
        x++;
        continue;
    }
    a = 1/(x-7);
    printf("%f", a);
    x++;
}

```

VARIÁVEIS INDEXADAS

VARIÁVEIS INDEXADAS

Existem diversas situações na área de processamento de dados que existe a necessidade de se armazenar um grande conjunto de dados na memória RAM do computador. Muitas dessas situações os dados estão relacionados.

Por Exemplo:

- Ler todos os nomes dos alunos da turma de LPE e ordena-los alfabeticamente.
- Ler todas as notas dos alunos desta turma e classifica-los imprimindo os que tiveram média acima de sete.

Nestes casos é inviável utilizarmos uma variável para cada aluno ou nota, sendo assim, as linguagens de programação incluem um mecanismo chamado **variável indexada** que permite realizar este armazenamento com uma única variável. O termo *indexada* provém da maneira como esta individualização é feita: por meio de **índices**.

VARIÁVEIS INDEXADAS

Variáveis indexadas são denominadas arrays que se dividem em duas partes:

- Arrays Unidimensionais : também chamadas vetores (um único índice é usado);
- Arrays Multidimensionais : duas ou mais dimensões (possui dois ou mais índices).

Operações Básicas com Variáveis Indexadas

Do mesmo modo que acontece com variáveis simples, também é possível operar com variáveis indexadas. Contudo, não é possível operar diretamente com o conjunto completo, mas com cada um de seus componentes isoladamente. O acesso individual a cada componente de um conjunto é realizado pela especificação de sua posição no mesmo por meio de um ou mais índices (no caso de uma matriz).

VARIÁVEIS INDEXADAS

Exemplo 01: Ler três números inteiros, achar a média e imprimir os números maiores que a média

Solução Tradicional:

Algoritmo <Programa_Imprime>

Início

Inteiro A1, A2, A3;

Real Media;

Ler A1;

Ler A2;

Ler A3;

$Media \leftarrow (A1 + A2 + A3) / 3;$

Se (A1 > Media) então

Escreva A1;

Fim_Se

Se (A2 > Media) então

Escreva A2;

Fim_Se

Se (A3 > Media) então

Escreva A3;

Fim_Se

Fim

Modelo de Memória

A1	A2	A3	Media
5	7	8	6.6

Obs.: Imagine o mesmo exercício para 100 valores.

VARIÁVEIS INDEXADAS

O uso de variáveis indexadas nos dá a possibilidade de combinar o nome de uma variável com um índice numérico. O nome da variável associado com o valor numérico está relacionado a uma posição de memória.

Algoritmo <Programa_10>

Início

Real X1, X2, X3, ..., X10, Media;

Ler X1;

Ler X2;

Ler X3;

:

:

Ler X10;

Media $\leftarrow (X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10) / 10$;

Escreve Media;

Escreve X1;

Escreve X2;

:

:

Escreve X10;

Fim

1ª Entrada de Dados

Aux[0] = 1;

Aux[1] = 5;

Aux[2] = 3;

:

:

Aux[9] = 45;

2ª Entrada de Dados (Mudança de Valores)

Aux[0] = 15;

Aux[1] = 25;

Aux[9] = 2;

Vetores e Matrizes

- Neste tipo de estrutura, os valores armazenados devem pertencer ao mesmo tipo.
- Entre outros nomes que estas estruturas recebem, iremos chamá-las de **Vetores e Matrizes**.

Vetores

São estruturas lineares e estáticas, ou seja, são compostas por um número finito e pré-determinado de valores.

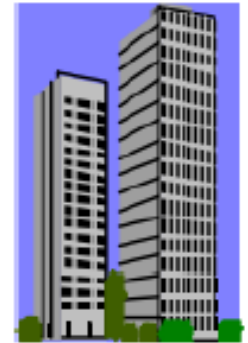
vetor1[5 3 7 6 6 12 23 8 9 7]

vetor1

5	3	7	6	6	12	23	8	9	7
---	---	---	---	---	----	----	---	---	---

Vetores

- Estrutura de Dados Homogênea e Estática
 - Unidimensional
- Exemplo :
 - Prédio com **um** apartamento por andar
 - Conjunto habitacional com apenas uma rua
- Todos os elementos pertencentes ao mesmo tipo de dado;



Posicionamento em Vetores

Levando em consideração que a primeira posição do vetor seja 0, teremos:

`vetor1[0] = 5`

`vetor1[1] = 3`

`vetor1[2] = 7`

`vetor1[3] = 6`

`vetor1[4] = 6`

...

`vetor1[9] = 7`

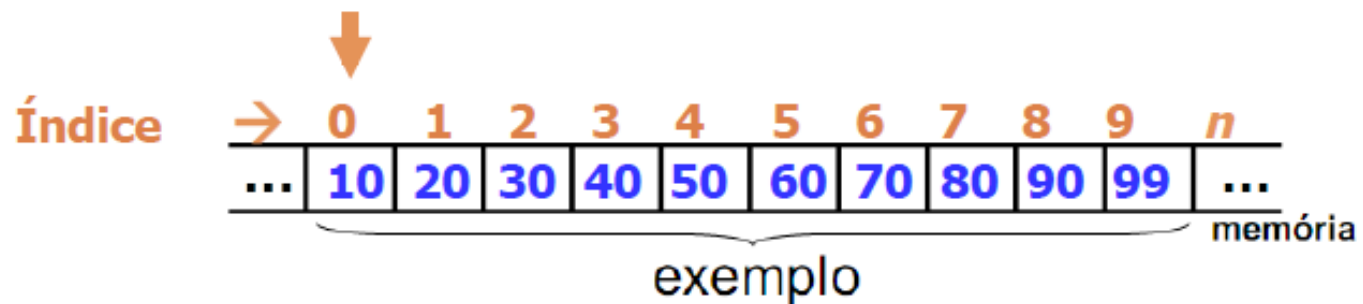
vetor1									
5	3	7	6	6	12	23	8	9	7
0	1	2	3	4	5	6	7	8	9

Para fazer referência a um valor de um elemento contido em um vetor, usamos a notação `vetor[índice]`, que serve tanto para obter quanto para definir o valor de um elemento específico, dada sua posição

Note que os elementos são numerados a começar do zero, e, portanto, se o número de elementos é N , o índice ou posição do último elemento será $N - 1$.

Posicionamento em Vetores

- Índices (iniciam em “0”, até “n”);
- Índices utilizados para Recuperar/Inserir valores.



- Forma geral para se declarar um vetor:
- **tipo_da_variável nome_da_variável [tamanho];**

Vetores

- Mas ninguém o impede (programador) de escrever:
 - teste[30]
 - teste[103]
 - teste[-2]

- O C não verifica se o índice que você usou está dentro dos limites válidos. Este é um cuidado que você deve tomar.

- Se o programador não tiver atenção com os limites de validade para os índices ele corre o risco de ter variáveis sobrescritas ou de ver o computador travar. Inúmeros bugs podem surgir.

Exemplo:

74

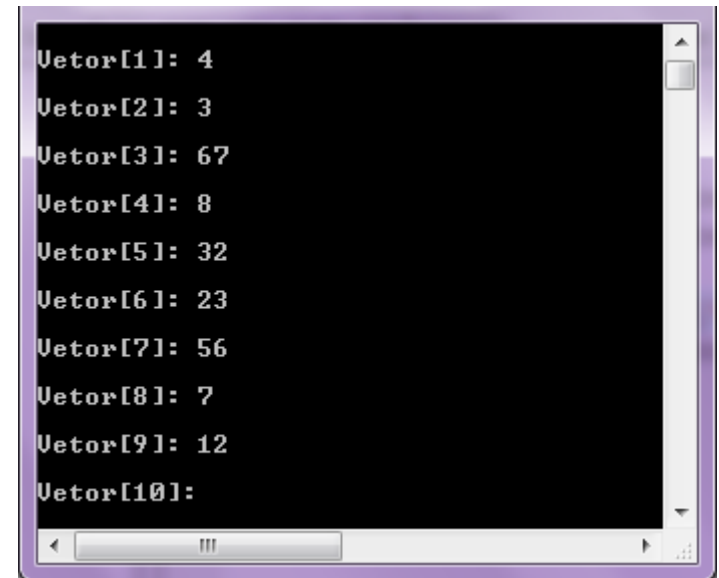
```
int vetorDois[10];
int x;
vetorDois[11] = 32;
x = vetorDois[13];
```

Vetores

Exemplo:

Construa um programa que declare um vetor de inteiros com 11 elementos e o inicialize com números fornecidos pelo usuário, através da entrada padrão.

```
#include <stdio.h>
main()
{
    int vetor[10], indice;
    for (indice=0; indice<10; indice++)
    {
        printf("\nVetor[%d]: ",indice);
        scanf("%d",&vetor[indice]);
    }
}
```



```
Vetor[1]: 4
Vetor[2]: 3
Vetor[3]: 67
Vetor[4]: 8
Vetor[5]: 32
Vetor[6]: 23
Vetor[7]: 56
Vetor[8]: 7
Vetor[9]: 12
Vetor[10]:
```

Vetores

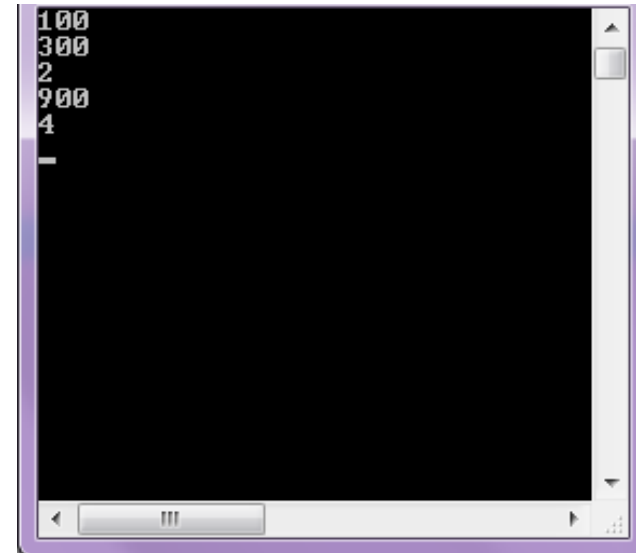
- Um vetor pode ser inicializado na declaração, exemplo:
 - `int vetor[10]={0,1,2,3,4,5,6,7,8,9};`
- E ainda pode-se deixar em aberto o número de elementos, que será preenchido pelo números de elementos na inicialização, que ocorre no momento da declaração. Ou seja:
 - `int vetor[]={0,1,2,3,4,5,6,7,8,9};`

Vetores

Valores das posições podem ser modificados no programa:

```
#include <stdio.h>

int main() {
    int sal[]={0,1,2,3,4};
    sal[0]=100;
    sal[1]=300;
    sal[3]=900;
    printf ("%d\n",sal[0]);
    printf ("%d\n",sal[1]);
    printf ("%d\n",sal[2]);
    printf ("%d\n",sal[3]);
    printf ("%d\n",sal[4]);
    getchar();
    return (0); }
```

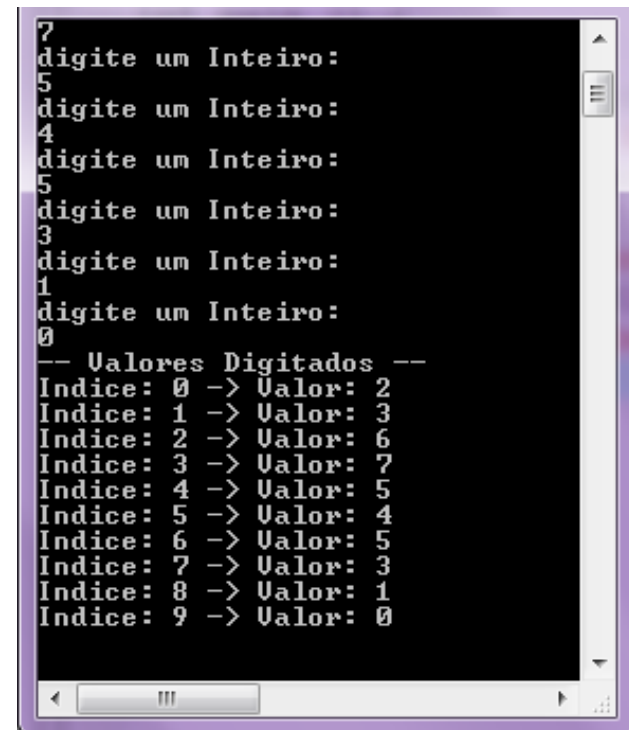


Vetores

Exercício1:

Construa um programa que declare e receba um vetor de inteiros com 10 elementos com números fornecidos pelo usuário, através da entrada padrão e depois exiba os índices e seus valores armazenados.

```
#include <stdio.h>
int main () {
    int vetorInteiros[10];
    for(int x=0; x<10; x++){
        printf("digite um Inteiro: \n");
        scanf("%d",&vetorInteiros[x]);
        if(x==9){
            printf("-- Valores Digitados -- \n");
            for(int y=0; y<10; y++){
                printf("Indice: %d -> Valor: %d \n", y, vetorInteiros[y]); } } }
    getchar();
    return(0); }
```



```
7
digite um Inteiro:
5
digite um Inteiro:
4
digite um Inteiro:
5
digite um Inteiro:
3
digite um Inteiro:
1
digite um Inteiro:
0
-- Valores Digitados --
Indice: 0 -> Valor: 2
Indice: 1 -> Valor: 3
Indice: 2 -> Valor: 6
Indice: 3 -> Valor: 7
Indice: 4 -> Valor: 5
Indice: 5 -> Valor: 4
Indice: 6 -> Valor: 5
Indice: 7 -> Valor: 3
Indice: 8 -> Valor: 1
Indice: 9 -> Valor: 0
```

Vetores

Exercício2:

Construa um programa que declare e receba um vetor de inteiros com 10 elementos e que o conteúdo de cada posição do vetor será o seu índice ao quadrado. Criei um **for** para receber o conteúdo e outro para imprimir.

```
#include <stdio.h>

int main ( ) {

    int i;

    int vetor[10]; // declara um vetor de inteiros

    // Insere o quadrado dos números de 0 a 9 em cada posição do vetor
    for ( i = 0 ; i < 10 ; i ++ ) {
        vetor[i] = i * i;
    }

    //Exibe o conteúdo do vetor
    for ( i = 0 ; i < 10 ; i ++ ) {
        printf("\n%d" , vetor[i] );
    }

    printf("\n");

}
```

Vetores

Exercício3:

Faça um programa que leia um vetor tamanho 10 com dados obtidos a partir do usuário e exiba os valores recebidos de forma invertida.

```

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

int main () {
    int vet1[10];
    for (int x=0; x<10;x++)
    {
        printf ("Indice: %d - Digite um inteiro:", x);
        scanf ("%d", &vet1[x]);
        if (x==9){
            printf("\n \n -- Valores digitados --\n \n Vetor 2(Invertido)\n \n");
            for (int y=9; y>=0; y--){
                printf ("Indice: %d - Valor %d \n", y, vet1[y]);
            }
        }
    }
    getch();
    return(0);
}
  
```


Vetores

Exercício4:

Faça um programa que leia um vetor de 10 posições e crie um segundo vetor substituindo os valores negativos por 1. Exiba os resultados do segundo vetor.

```
#include <stdio.h>
#include <conio.h>
int main () {
    int vet[10];
    int i=0;
    do {
        printf ("Digite o valor %d do vetor: ", i);
        scanf ("%d", &vet[i]);
        i++; }
    while (i<=9);
    printf ("\n");
    for (i=0; i<10; i++){
        if (vet[i]<0)
            vet[i]=1;
        printf(" %d ", vet[i]);
    }
    getch();
    return(0);
}
```

```
C:\Users\jorge\Documents\univas\d...
Digite o valor 0 do vetor: 1
Digite o valor 1 do vetor: 2
Digite o valor 2 do vetor: 3
Digite o valor 3 do vetor: -4
Digite o valor 4 do vetor: 4
Digite o valor 5 do vetor: -6
Digite o valor 6 do vetor: -3
Digite o valor 7 do vetor: 2
Digite o valor 8 do vetor: -9
Digite o valor 9 do vetor: -3

1 2 3 1 4 1 1 2 1 1
```

Strings

Strings

Uma **string** é uma cadeia ou sequência de caracteres. As strings são usadas para armazenar nomes, palavras e frases.

Na linguagem C strings são vetores de caracteres que possuem um caractere que indica o término de seu conteúdo, o caractere nulo '\0' (contra barra zero).

Declaração de strings

Como a string possui o caractere nulo para delimitar o final do seu conteúdo, o tamanho da string deve ser definido com um caractere a mais do que será efetivamente necessário.

Sintaxe: char identificador-da-string [tamanho+1];

Strings

Exemplo:

```
char vetc [6];
```

vetc é um vetor de caracteres (string) de tamanho 6. Pode receber uma palavra de no máximo 5 letras

Strings

Inicialização de strings

Uma string pode ser inicializada na sua declaração com uma sequência de caracteres entre chaves e separadas por vírgula.

```
char vetc[6]= {'T', 'e', 'x', 't', 'o', '\0'};
```

*Lembre-se que o compilador só reconhecerá um caractere se este estiver entre aspas simples, logo usar uma atribuição do tipo {t,e,x,t,o,\0} ou {texto\0} **irá gerar um erro de compilação.***

Strings

Inicialização de strings

Uma string pode também ser inicializada por uma sequência de caracteres entre aspas duplas. Neste caso, não é necessário o uso de aspas simples e vírgulas, o compilador C coloca automaticamente o '\0' no final.

```
char vetc[6] = "Texto";
```

Assim como vetores e matrizes, na inicialização de uma string o seu tamanho pode ser omitido.

```
char vetc[ ] = "Texto"; /* vetor não-dimensionado, o compilador coloca automaticamente o '\0' no final */
```

Strings

Leitura de strings

*Utilizando a função **scanf()***

A sintaxe para receber uma string por meio da `scanf()` é:

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

OBS.: não é necessário colocar o operador `&`, pois o nome da string em si já é um endereço de memória.

Strings

Leitura de strings

Exemplo: Crie um aplicativo em C que peça ao usuário seu nome, armazene em uma String, peça o sobrenome, armazene em outra string e exiba o nome do usuário de maneira formal (Sobrenome, Nome).

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char nome[21], sobrenome[21];

    printf("Primeiro nome: ");
    scanf("%s", nome);

    printf("Ultimo sobrenome: ");
    scanf("%s", sobrenome);

    printf("Ola senhor %s, %s. Bem-vindo ao curso de linguagem C.\n", sobrenome, nome);

    system("pause");
}
```


Strings

Leitura de strings

O que aconteceria se fosse digitado um nome composto no
'nome' ou no 'sobrenome'?

A *scanf()* vai simplesmente cortar seu nome composto. Essa função pega tudo **até** encontrar um espaço em branco, caractere *new line* `\n`, tab ou ENTER.

Strings

Funções para Manipulação de Strings

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    char nome[] = "fulano";
    char sobrenome[] = "de tal";
    char nomeCompleto[] = nome + sobrenome;
    int i=0;

    while(nomeCompleto[i] != '\0'){
        printf("%c", nomeCompleto[i]);
        nomeCompleto[i++];
    }

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

PORQUE O PROGRAMA NÃO COMPILA?

Um erro muito comum no uso de string em C esta sendo cometido na linha 7. char nomeCompleto[] = nome + sobrenome. String não poder ser concatenadas utilizando o operador +. Existe uma diretiva em C que implementa diversas funções de manipulação de valores em string.

Strings

Funções para Manipulação de Strings

Get String (gets)

A função **gets()** lê os caracteres do dispositivo padrão de entrada (teclado) para o seu argumento – um vetor do tipo **char** – até que um caractere de nova linha ou o indicador de fim de arquivo seja encontrado. Um caractere **NULL** (**'\0'**) será adicionado ao vetor quando a leitura terminar. Sua forma geral é:

gets (nome_da_string);

Strings

Funções para Manipulação de Strings

Get String (gets)

OBS.: Mais uma vez, nunca use & quando for armazenar uma string.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char string[100];

    printf("Digite o seu nome: ");
    gets(string);
    printf ("\n\n Ola %s\n",string);

    system("pause");
}
```

Como o primeiro argumento da função **printf()** é uma string também é válido fazer:

```
printf (string); /* isto simplesmente imprimirá a string. */
```

Strings

Funções para Manipulação de Strings

Get String (gets)

Não é uma função segura, pois o tamanho da string não é especificado.

A função *scanf()* pega tudo até aparecer o primeiro espaço em branco, e pára.

Já a *gets()* não, ela pega tudo até aparecer uma *new line* `\n`, inclusive nada.
Ou seja, se você der um ENTER, a *gets()* vai armazenar esse enter na string.

Strings

Funções para Manipulação de Strings

Get String (gets) e Scanf

Que erro acontece no exemplo abaixo?

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    char nome[31], sobrenome[31], nascimento[11];
    int idade;

    printf("Nome: ");
    gets(nome);

    printf("Sobrenome: ");
    gets(sobrenome);

    printf("Idade: ");
    scanf("%d", &idade);

    printf("Data de nascimento: ");
    gets(nascimento);

    printf("\nNome completo: %s %s\n", nome, sobrenome);
    printf("Idade: %d\n", idade);
    printf("Data de nascimento: "); puts(nascimento);

    system("pause");
}
```

a função *scanf()* pega tudo até aparecer o primeiro espaço em branco, e pára antes dele.

Já a *gets()* não, ela pega tudo até aparecer uma *new line* `\n`, inclusive nada. Ou seja, se você der um ENTER, a *gets()* **vai armazenar esse ENTER na string.**

Strings

Funções para Manipulação de Strings

Get String (gets) e Scanf

O problema é que a função `gets()` vai pegar o que está armazenado nesse buffer e vai armazenar o que estiver lá na string de data de nascimento!

E como evitar isso? É só apagar esse ENTER que está no buffer, usando o `fflush(stdin)` caso use Windows, ou `__fpurge(stdin)` caso use Linux.

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    char nome[31], sobrenome[31], nascimento[11];
    int idade;

    printf("Nome: ");
    gets(nome);

    printf("Sobrenome: ");
    gets(sobrenome);

    printf("Idade: ");
    scanf("%d", &idade);

    fflush(stdin);

    printf("Data de nascimento: ");
    gets(nascimento);

    printf("\nNome completo: %s %s\n", nome, sobrenome);
    printf("Idade: %d\n", idade);
    printf("Data de nascimento: "); puts(nascimento);

    system("pause");
}
```

Strings

Funções para Manipulação de Strings

Get String (gets) e Scanf

É possível alterar o funcionamento da ***scanf()***. Por exemplo, se quisermos ler strings que tenham espaço, nós temos que dizer isso dentro da função.

Para dizer para a ***scanf()*** parar de pegar nossa string somente quando encontrar um caractere de NEW LINE (um enter). Usamos o operador: **`[\n]`**

Logo, nosso código da ***scanf()*** para ler strings com espaços e armazenar na variável "str" é:

```
scanf ( "[%\n]", str);
```

(OBS.: lembre-se de limpar o buffer, usando **`__fpurge(stdin)`**).

Strings

Funções para Manipulação de Strings

Get String (gets) e Scanf

Podemos ainda limitar o tamanho de nossa string, basta colocar um numero inteiro ao lado do %, representando o número de caracteres máximo, o que é uma excelente prática, pois essa função pode ocasionar problemas na memória, caso você estoure os limites da string.

Por exemplo:

```
scanf ( "%256[^\n]", str);
```

Strings

Funções para Manipulação de Strings

Get String (gets) e Scanf

A função **gets()** peca nesse quesito, de tamanho da string, pois podemos digitar mais caracteres do que a string alocou de memória, e "quebraríamos" o programa por conta de um *overflow*.

Uma solução para isso é usar a função **fgetc()**, que é mais segura.

Ela recebe três dados:

- A string que vai armazenar o que vai ser digitado (no nosso caso é a variável "str");
- O tamanho da string e de onde vai ler (ela pode ler de um arquivo de texto, por exemplo);
- Para ler do teclado, usamos **stdin**.

Por exemplo:

fgetc(str, 256, stdin);

Strings

Funções para Manipulação de Strings

Outras Funções

Incluir a biblioteca **string.h**

*Strings não podem ser comparadas com o operador de comparação padrão (==), neste caso deve-se usar função **strcmp()** ou a função **stricmp()**.*

- **strcmp(s1,s2)** – Retorna 0 se s1 e s2 são iguais; menor que 0 se s1<s2; maior que 0 se s1>s2 (comparação alfabética).
- **stricmp(s1,s2)** – Retorna 0 se s1 e s2 são iguais; menor que 0 se s1<s2; maior que 0 se s1>s2 (comparação alfabética). Essa função considera letras maiúsculas ou minúsculas como símbolos iguais.

Strings

Funções para Manipulação de Strings

Outras Funções

Incluir a biblioteca **string.h**

*Strings não podem ser atribuídas com o operador de atribuição (=), para uma atribuição usa-se a função **strcpy()**.*

strcpy(s1,s2) – Copia s2 em s1.

OBS.: A string-destino deve ser grande o suficiente para armazenar a string origem e seu caractere **NULL** de terminação que também é copiado.

Strings

Funções para Manipulação de Strings

Outras Funções

Incluir a biblioteca **string.h**

*Strings não podem ser concatenadas com o operador (+), para tal usa-se a função **strcat()**.*

strcat(s1,s2) – Concatena s2 ao final de s1.

strlen(s) – Retorna o número de caracteres em s (sem contar o caracter nulo (/0)).

Matrizes

Matrizes

- Vetores Multidimensionais

- Estrutura de Dados Homogênea
 - Bidimensional

- Exemplo :
 - Prédio com mais de um apartamento por andar
 - Conjunto habitacional com várias ruas

Matrizes

- Assim como nos vetores, cada posição dentro de uma matriz possui um índice.
- Nos vetores, este índice era composto de apenas um valor
- Exemplos:

Vetor de 5 posições (1 dimensão: colunas)

5	3	7	6	6
0	1	2	3	4

Matriz de 5x5 posições (2 dimensões: colunas e linhas)

0	A	B	C	D	E
1	F	G	H	I	J
2	K	L	M	N	O
3	P	Q	R	S	T
4	U	V	W	X	Y
	0	1	2	3	4

104

Matrizes

- As matrizes precisam de n índices para identificar uma de suas posições, sendo n o número de dimensões da matriz.

Vetor V =

0	A
1	B
2	C
3	D
4	E

Vetor V[3] = D

Vetor V[0] = A

Vetor V[2] = C

Matriz M =

	0	1	2	3	4
0	A	B	C	D	E
1	F	G	H	I	J
2	K	L	M	N	O
3	P	Q	R	S	T
4	U	V	W	X	Y

Matriz M[2,0] = K

Matriz M[0,0] = A

Matriz M[2,4] = O

Matriz M[4,4] = Y

Matrizes

A declaração de matrizes é feita de forma muito semelhante a declaração dos vetores. A diferença será o acréscimo de mais um intervalo (dimensão):

Matrizes

- Declarando uma matriz
tipo_da_variável nome_da_variável [d1][d2]...[dn];

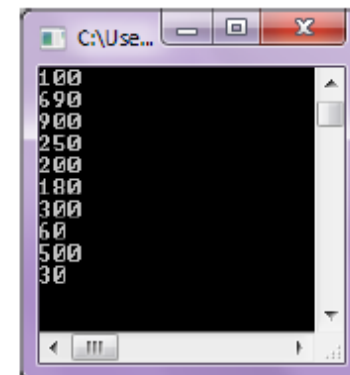
Exemplo:

Declarar uma estrutura com o valor de salario, com dez elementos inteiros dispostos numa matriz com cinco linhas e duas colunas.

```
#include <stdio.h>
main(){
    int sal[5][2],x,y;
    sal[0][0]=100;
    sal[0][1]=690;
    sal[1][0]=900;
    sal[1][1]=250;
    sal[2][0]=200;
    sal[2][1]=180;
    sal[3][0]=300;
    sal[3][1]=60;
    sal[4][0]=500;
    sal[4][1]=30;

    for(x=0;x<5;x++){
        for(int y=0;y<2;y++){
            printf ("%d\n", sal[x][y]);}
        getchar();
    }
    return(0);
}
```

4	500	30
3	300	60
2	200	180
1	900	250
0	100	690
	0	1



Matrizes

- Exemplo: O programa abaixo cria e exibe uma matriz com dimensões e valores definidas pelo usuário .

```

Entre com o numero de linhas da matriz = 2
Entre com o numero de colunas da matriz = 2
Entre com o elemento[1][1]=1
Entre com o elemento[1][2]=2
Entre com o elemento[2][1]=3
Entre com o elemento[2][2]=4

| 1 | 2 |
| 3 | 4 |
    
```

```

#include <stdio.h>
int main()
{
    // definição do tamanho da matriz
    int nl, nc;
    printf ("\nEntre com o numero de linhas da matriz = ");
    scanf ("%d",&nl);
    printf ("\nEntre com o numero de colunas da matriz = ");
    scanf ("%d",&nc);
    // entrada de dados da matriz
    int matriz[nl][nc], i, j;
    for (i=0;i<nl;i++)
        for (j=0;j<nc;j++)
        {
            printf ("\nEntre com o elemento[%d][%d]=",i+1,j+1);
            scanf ("%d",&matriz[i][j]); }

    // impressão da matriz na tela
    for (i=0;i<nl;i++) {
        printf("\n |"); // barra vertical
        for (j=0;j<nc;j++)
            printf (" %d ",matriz[i][j]);
        printf("|");
    }
    getchar();
    return (0);
}
    
```

Matrizes

Exercício8 - Faça um programa que leia uma matriz **mat** 3 x 4 de inteiros, substitua seus elementos negativos por 0 e imprima a matriz **mat** original e a modificada.

```
#include <stdio.h>
#include <conio.h>
main() {
    int matriz[3][4], i, j;
    for (i=0;i<3;i++) {
        for (j=0;j<4;j++) {
            printf ("\nEntre com o elemento[%d][%d]=",i+1,j+1);
            scanf ("%d",&matriz[i][j]); } }
    printf ("\n Matriz Principal \n");
    for (i=0;i<3;i++) {
        for (j=0;j<4;j++) {
            printf ("\n Valor: %d",matriz[i][j]); }}
    printf ("\n \n Matriz Modificada \n");
    for (i=0;i<3;i++) {
        if (matriz [i][j]<0) {
            matriz[i][j]=0;
            printf ("\n Valor: %d",matriz[i][j]); }
    }

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

Introdução à Ciência da Computação – 2015.2

REFERÊNCIAS

- Damas, Luãs. Linguagem C. 10ª Ed. Editora LTC, 2007.
- GUIMARÃES, Ângelo de Moura e LAGES, Newton Alberto de Castilho. **Introdução à Ciência da Computação**. Rio de Janeiro: LTC-Livros Técnicos e Científicos, 2010. 165p.
- Márcio Alexandre Marques. Algoritmos - Lógica para Desenvolvimento de Programação de Computadores. 1ª Ed. Editora Érica, 2010.
- Sandra Rita. TREINAMENTO EM LOGICA DE PROGRAMAÇÃO, Digerati Books, 1 ed. 2009.
- SIMÃO, DANIEL HAIASHIDA; REIS, WELLINGTON JOSÉ DOS. LOGICA DE PROGRAMAÇÃO. São Paulo : EDITORA VIENA, 2015. 176p.
- Souza, Marco Antonio Furlan de *et. all*, Algoritmos e Lógica de Programação. 2 ed. São Paulo : Nobel, 2011.