

# FPD – Ap 1 – C

Conteúdo: FUNÇÕES

Curso: Informática.

Turmas: 71A, 71B, 71C

Profª Ariane Scarelli

04/06/2020

*Identifique sua apostila*

Nome: \_\_\_\_\_

Turma: \_\_\_\_\_

## SUMÁRIO

1	INTRODUÇÃO	1
2	CONVENÇÕES DA LINGUAGEM C	1
3	FUNÇÕES	1
3.1	POR QUE USAR FUNÇÕES?	2
3.2	UTILIZANDO FUNÇÕES	2
3.3	O TIPO void	5
3.4	COMANDO return	6
3.5	ESCOPO DAS VARIÁVEIS	12
3.6	PASSAGEM DE PARÂMETROS POR VALOR E POR REFERÊNCIA	17
4	BIBLIOTECA conio.c	21
4.1	FUNÇÃO gotoxy()	21
4.2	FUNÇÃO Sleep()	21
4.3	FUNÇÃO textcolor()	22
4.4	FUNÇÃO textbackground()	23
4.5	FUNÇÃO clrscr()	24
4.6	FUNÇÃO clrhol()	25
5	REFERÊNCIAS BIBLIOGRÁFICAS	26

Todos os direitos reservados e protegidos por Lei de Direito Autoral, de  
19/02/1998, artigo 7º. Nenhuma parte desse material poderá ser  
reproduzida ou transmitida sem citação de autoria.

## 1 INTRODUÇÃO

Este material didático refere-se ao módulo de estudo das funções criadas pelo programador na linguagem C. É parte do módulo avançado de estudos em linguagem C da disciplina FPD.

## 2 CONVENÇÕES DA LINGUAGEM C

Ao criar seus programas utilizando a linguagem C, observe que existem convenções que devem ser seguidas para que o compilador traduza o código escrito em linguagem de máquina (binário).

- ☞ Não há formato específico para digitação das linhas de um programa;
- ☞ Convencionou-se o uso de endentação (tabulação que torna o programa mais legível);
- ☞ C distingue letras maiúsculas de minúsculas;
- ☞ Os parênteses que seguem a palavra (função) *main* são obrigatórios, assim como, os parênteses que seguem o nome de uma função criada pelo programador;
- ☞ Todas as instruções devem estar dentro das chaves que iniciam e terminam a função e são executadas na ordem em que aparecem;
- ☞ Deve-se colocar ponto-e-vírgula após cada instrução do programa;
- ☞ Uma função pode receber qualquer nome, desde que não seja um já existente, nem o nome dado ao programa fonte;
- ☞ Colocação de comentários em C: /\* para abrir e \*/ para fechar, sendo que tudo o que estiver entre os mesmos será ignorado pelo compilador; // indicam que tudo o que estiver à direita das barras será ignorado;
- ☞ O início lógico do programa é identificado pela função *main*, independentemente da sua posição física dentro do programa fonte.

## 3 FUNÇÕES

Uma função é um pedaço de código ou bloco de código reutilizável que executa uma ação específica. Toda linguagem de programação contém funções pré-definidas da linguagem e

funções que podem ser criadas pelo programador. Na linguagem C, por exemplo, podemos citar como função pré-definida: printf, scanf, getch, if, for, entre outras. Neste material, aprenderemos a criar funções do programador para executar diferentes tarefas.

Funções podem tanto retornar valores quando são chamadas, quanto simplesmente executar uma operação sem retornar valor algum.

### 3.1 POR QUE USAR FUNÇÕES?

Funções são extremamente úteis na criação dos códigos de programação pelas seguintes razões:

- Modularização – funções permitem agrupar blocos de códigos relacionados que executam uma tarefa específica juntos, dando melhor organização.
- Reutilização – uma vez definida, a função pode ser chamada por um número indefinido de vezes, em diferentes programas, poupando tempo do programador para ser aplicado em outras tarefas.
- Fácil manutenção – atualizações ficam localizadas em um só bloco.

### 3.2 UTILIZANDO FUNÇÕES

Uma função pode ser de tipo simples, que não envia valor e não retorna resultado, ou mais elaborada, sendo: a) envia valor(es) e não retorna resultado; b) envia valor(es) e retorna resultado.

Valores enviados à função são chamados de parâmetros ou argumentos.

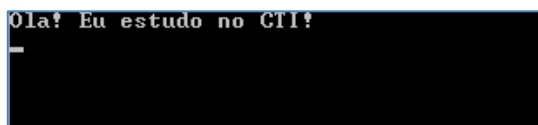
Os exemplos a seguir demonstram os diferentes tipos de funções.

Exemplo 1 – função simples, não envia valor e não retorna resultado

```
1 //bibliotecas
2 #include <stdio.h>
3 #include <conio.h>
4
5 void mensagem ()
6 {
7     printf ("Ola! ");
8 }
9
10 main ()
```

```
11 {
12     mensagem();
13     printf ("Eu estudo no CTI!\n");
14     getche();
15 }
```

A tela abaixo demonstra a execução do exemplo 1.



### EXERCÍCIOS (com função simples)

- 1) Faça um programa que contenha uma função para imprimir na tela o seu nome.
- 2) Faça um programa que contenha uma função que apresente o nome do seu bicho de estimação na tela 9 vezes (utilize laço de repetição).
- 3) Faça um programa que contenha uma função que apresente na tela o seguinte padrão de imagem. São 4 linhas com 10 asteriscos em cada (utilize laço de repetição).
 

```
*****
*****
*****
*****
```
- 4) Faça um programa que chame uma função que peça ao usuário para digitar seu ano de nascimento, calcular e apresentar sua idade atual na tela.

### Exemplo 2 – função com envio de um parâmetro, sem retorno de resultado

```
1 //bibliotecas
2 #include <stdio.h>
3
4 void quadrado (int x) /* Calcula o quadrado de x */
5 {
6     int quad;
7     quad = x * x;
8     printf ("\n\nO quadrado eh %d", quad);
9 }
10 main ()
11 {
12     int num;
13     printf ("Entre com um numero: ");
14     scanf ("%d",&num);
15     quadrado(num);
16 }
```

A tela abaixo demonstra a execução do exemplo 2.

```

Entre com um numero: 7
0 quadrado e 49
-----
Process exited with return value 0
Press any key to continue . . .

```

Exemplo 3 – função com envio de três parâmetros de dois tipos diferentes, sem retorno de resultado

```

1 //bibliotecas
2 #include <stdio.h>
3
4 void mult (float a, float b, int c)      /* Multiplica 3 numeros */
5 {
6     printf ("%0.2f",a*b*c);
7 }
8 main ()
9 {
10     float x,y;
11     int z;
12     x=2.0;
13     y=5.5;
14     z=2;
15     mult (x,y,z);
16 }

```

A tela abaixo demonstra a execução do exemplo 3.

```

22.00
-----
Process exited with return value 0
Press any key to continue . . .

```

EXERCÍCIOS (com envio de parâmetro(s) à função)

- 5) Faça um programa que contenha uma função para cálculo da idade atual de uma pessoa. Leia o ano de nascimento em *main* e envie como parâmetro para a função, calcule e mostre a idade na tela.
- 6) Faça um programa que contenha uma função para cálculo do cubo de um número float. Envie como parâmetro o número digitado pelo usuário. Apresente o resultado dentro da função.

- 7) Faça um programa que contenha uma função que calcule  $A+B*C$ . Crie variáveis int em main, que devem ser lidas. Chame a função, envie as variáveis como parâmetros, efetue o cálculo e apresente o resultado na tela.
- 8) Faça um programa que contenha uma função que determine se um número int lido pelo teclado em main é negativo, positivo ou nulo e imprima, na mesma função, o texto: "é negativo" ou "é positivo" ou "é nulo", conforme o caso.
- 9) Faça um programa que contenha uma função para cálculo da tabuada de um número (int lido em main). O número deverá ser enviado à função, que irá calcular e apresentar o resultado na tela no formato  $a \times b = c$ .

Exemplo: tabuada do 7, apresentar:

$$7 \times 1 = 7$$

$$7 \times 2 = 14, \text{ e assim, sucessivamente, até } 10$$

- 10) Faça um programa que contenha uma função para conversão de temperatura de Fahrenheit em Celsius. A temperatura, tipo float, deve ser digitada em main. Envie a temperatura como parâmetro para a função, converta em Celsius e apresente o resultado dentro função com uma casa decimal.

$$C = \frac{5}{9} (F - 32)$$

- 11) Faça um programa que contenha uma função que calcule e mostre na tela o volume e a área da esfera, baseando-se nas fórmulas dadas. Receba (leia com scanf) em *main* o valor do raio de uma esfera (tipo float) e envie-o à função. Utilize para pi o valor constante de 3.14159265.

$$AREA = 2\pi R^2$$

$$VOLUME = \frac{4}{3}\pi R^3$$

### 3.3 O TIPO void

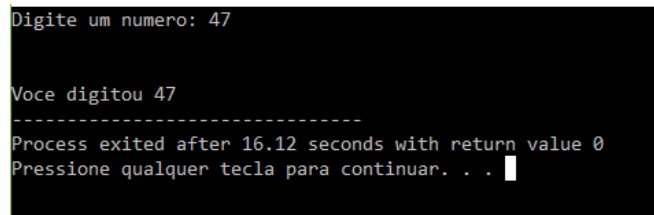
O compilador C assume que uma função retorna um valor inteiro até que se especifique o contrário. Contudo, nem todas as funções necessitam de um valor de retorno. Para esses casos, C fornece o tipo void (vazio).

Quando void é colocado antes do nome da função, está dizendo ao compilador C que a função não retorna um valor.

## Exemplo 4 – tipo void

```
1 //bibliotecas
2 #include<stdio.h>
3
4 //cabeçalhos de funções
5 void mostra_digito (int); /*cabeçalho da função*/
6
7 main()
8 {
9     //void mostra_digito (int); //cabeçalho da função tb pode vir aqui
10    int digito;
11    printf ("Digite um numero: ");
12    scanf ("%d",&digito);
13    mostra_digito (digito);
14 }
15
16 void mostra_digito (int num)
17 {
18     printf("\n\nVoce digitou %d", num);
19 }
```

A tela abaixo demonstra a execução do exemplo 4.



```
Digite um numero: 47
Voce digitou 47
-----
Process exited after 16.12 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

### 3.4 COMANDO return

O comando return possui dois usos importantes:

- . saída imediata da função
- . retorno de um valor da função

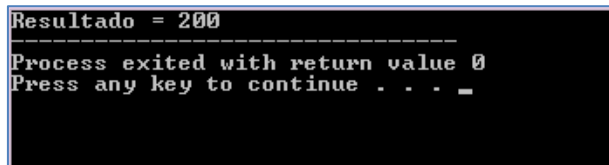
**Saída da função:** podem existir quantos comandos *return* forem necessários em uma função, porém quando o primeiro deles for executado ocorrerá o fim da função.



## Exemplo 5 – comando return

```
1 //bibliotecas
2 #include<stdio.h>
3
4 int calculo(int a1, int b1)
5 {
6     int c1;
7     c1 = a1 * b1;
8     return c1;
9 }
10
11 main()
12 {
13     int a,b,c;
14     a = 10;
15     b = 20;
16     c = calculo(a,b);
17     printf("Resultado = %d", c);
18 }
```

A tela abaixo demonstra a execução do exemplo 5.



```
Resultado = 200
-----
Process exited with return value 0
Press any key to continue . . . _
```

## Exemplo 6 – comando return como saída antecipada da função

```
1 //bibliotecas
2 #include<stdio.h>
3
4 void calculo(int idade)
5 {
6     int c1;
7     if (idade<=0)
8     {
9         printf("Idade invalida, impossivel calcular!");
10        return;
11    }
12    c1 = idade * 365;
13    printf("Voce ja viveu %d dias", c1);
14    return;
15 }
16
17 main()
18 {
19     int id;
20     printf("Digite sua idade: ");
21     scanf("%d",&id);
```

```
    calculo(id);
}
```

As telas abaixo demonstram a execução do exemplo 6.

(1) Execução 1

```
Digite sua idade: -10
Idade invalida, impossivel calcular!
-----
Process exited after 3.05 seconds with return value 0
Pressione qualquer tecla para continuar. . . _
```

(2) Execução 2

```
Digite sua idade: 15
Voce ja viveu 5475 dias
-----
Process exited after 3.2 seconds with return value 0
Pressione qualquer tecla para continuar. . . _
```

Exemplo 7 – comando return como saída antecipada da função; funções gets e getche

```
1 //bibliotecas
2 #include<stdio.h>
3 #include <conio.h>
4
5 void procura( char str[30], char p )
6 {
7     int i;
8     for( i = 0; str[i]; i++ )
9         if( str[i] == p )
10            {
11                printf("\n\nEncontrou !");
12                return;
13            }
14     printf("\n\nNao encontrou !");
15     return;
16 }
17
18 main()
19 {
20     char str[30], p;
21     printf("Digite uma frase: ");
22     gets(str); // para ler uma string
23     printf("\nDigite um caractere a ser procurado na frase: ");
24     p = getche(); //para ler um caractere
25     procura( str, p );
26 }
```

A tela abaixo demonstra a execução do exemplo 7.

```
Digite uma frase: colegio tecnico
Digite um caractere a ser procurado na frase: c
Encontrou ?
-----
Process exited with return value 0
Press any key to continue . . . _
```

**Retorno de um valor:** todas as funções exceto as do tipo *void* devolvem um valor. Se não existir um comando *return*, o valor retornado será indefinido.

Exemplo 8.a – comando *return* devolvendo o resultado da função

```
1 //bibliotecas
2 #include<stdio.h>
3
4 //cabeçalhos de funções
5 int soma(int, int);
6
7 main()
8 {
9     int a = 10, b = 5;
10    printf("Soma = %d\n", soma(a,b) );
11    //observe acima que o retorno da função será impresso
12 }
13
14 int soma( int x, int y )
15 {
16     return x + y;
17 }
18 }
```

Exemplo 8.b – comando *return* devolvendo o resultado da função (outra forma)

```
1 //bibliotecas
2 #include<stdio.h>
3
4 //cabeçalhos de funções
5 int soma(int, int);
6
7 main()
8 {
9     int a = 10, b = 5, result;
10    result = soma(a,b); //observe que o retorno da função será atribuído a var. result
11    printf("\nSoma = %d", result);
12 }
13 int soma(int x, int y)
14 {
15     int res;
16     res = x + y;
```

```
17 | return res;  
18 | }
```

Ambos os exemplos, 8.a e 8.b, produzirão o mesmo resultado. A tela abaixo apresenta a execução.

```
Soma = 15  
-----  
Process exited with return value 0  
Press any key to continue . . . _
```

### EXERCÍCIOS (com retorno de resultado)

- 12) Escreva um programa para acessar pelo teclado valores para as variáveis (float) A, B e C, e calcular:  $A * B + C$ .

Obs.: - Acesse os valores na função *main*;  
- Efetue o cálculo na função *calculo*;  
- Apresente o resultado em *main*, com 2 casas decimais.

- 13) Escreva um programa para acessar pelo teclado os valores da resistência (float) e da corrente elétrica (float) para calcular o valor da tensão.

Obs.: - tensão = resistência \* corrente;  
- Acesse os valores na função *main*;  
- Crie uma função para efetuar o cálculo;  
- Crie uma função para apresentar o resultado.

- 13.1) Escreva um programa para acessar pelo teclado os valores da resistência (float) e da corrente elétrica (float) para calcular o valor da tensão.

Obs.: - tensão = resistência \* corrente;  
- Acesse os valores na função *main*;  
- Crie uma função para efetuar o cálculo;  
- Se a resistência ou a corrente forem negativas dê uma mensagem avisando o usuário e retorne da função sem executar o cálculo;  
- Apresente o resultado em *main*, se foi possível calculá-lo.

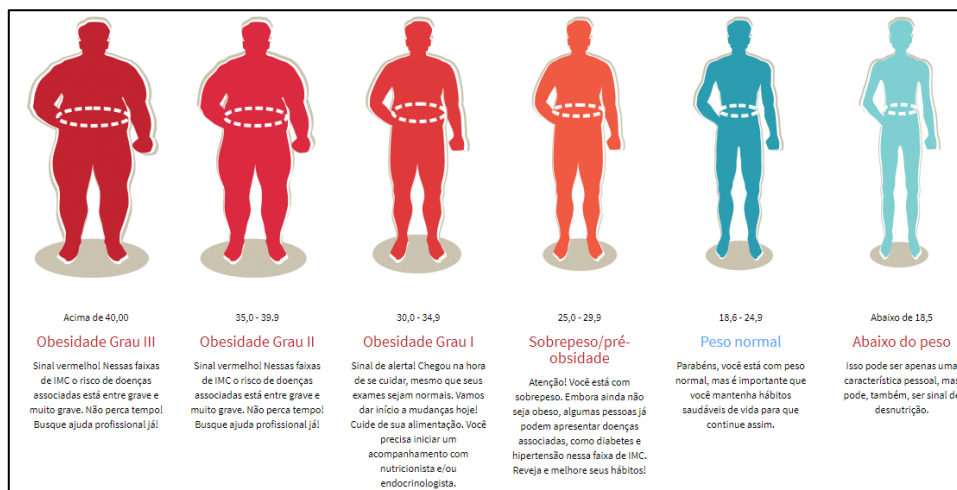
- 14) Faça um programa para calcular o IMC (índice de massa corporal) de uma pessoa, utilizando a fórmula:

$$\text{IMC} = \text{Peso} \div \text{Altura}^2$$

Onde o Peso é dado em Kg (float) e a Altura é dada em metros (float).

O programa deve enviar os dados para uma função para fazer o cálculo e retornar o resultado para *main*.

Apresente em *main* o IMC calculado e sua faixa de risco, baseando-se na seguinte tabela:



Fonte: <http://www.abeso.org.br/atitude-saudavel/imc>

IMC	Faixa de Risco
Abaixo ou igual a 18,5	Abaixo do peso
De 18,6 a 24,9	Peso normal
De 25 a 29,9	Sobrepeso/pré-obesidade
De 30 a 34,9	Obesidade grau I
De 35 a 39,9	Obesidade grau II
Igual ou acima de 40	Obesidade grau III

- 15) Faça um programa que contenha uma função para teste de consistência (validação) de uma nota digitada em *main*. A nota deve ser enviada à função para ser testada. Se a nota estiver incorreta permaneça na função (laço de repetição) para redigitá-la e retorne somente quando a nota for validada. Toda vez que a nota digitada estiver fora do intervalo de 0 a 10 dê uma mensagem de erro e peça para redigitar. **Mostre em *main* a nota válida.**

- 16) Faça um programa para calcular a média aritmética de 4 notas (float) digitadas em *main*. Crie uma função para calcular a média e retorne o resultado à *main* para ser apresentado na tela com uma casa decimal. Utilize a função de validação desenvolvida no exercício anterior para checar as notas, uma a uma, antes do cálculo da média.

- 17) Escreva um programa para calcular o fatorial de um número inteiro digitado em *main*. Crie uma função para efetuar o cálculo. O resultado deve ser apresentado na tela em *main*.

Exemplos: fatorial de 3 é igual a 6 →  $3 \times 2 \times 1 = 6$   
fatorial de 4 é igual a 24 →  $4 \times 3 \times 2 \times 1 = 24$

- 18) Escreva um programa para calcular a exponenciação de um número  $x$  elevado a  $y$ . Os valores de  $x$  e  $y$  devem ser digitados em *main*. Crie uma função para fazer o cálculo. Apresente o resultado na tela em *main*.

Exemplos: 2 elevado a 3 é igual a 8  $\rightarrow 2 \times 2 \times 2 = 8$   
5 elevado a 2 é igual a 25  $\rightarrow 5 \times 5 = 25$

### 3.5 ESCOPO DAS VARIÁVEIS

Quando criamos e utilizamos funções, as variáveis declaradas dentro de *main* e das funções apresentam escopo de utilização diferente das variáveis declaradas fora dos símbolos de agrupamento (inclusive de *main*). Elas recebem os nomes de variáveis locais e globais.

#### ✓ Variáveis Locais

Variáveis definidas dentro dos símbolos de agrupamento da função, inclusive *main*, são chamadas de variáveis locais.

Seus valores são conhecidos somente pela função que as declarou.

Existem apenas durante a execução do bloco de código no qual foram declaradas, sendo destruídas na saída.

#### ✓ Variáveis Globais

Variáveis definidas fora dos símbolos de agrupamento de qualquer função (inclusive a *main*) são chamadas de variáveis globais.

Seus valores são conhecidos e podem ser usados por todas as funções do programa.

Exemplos com variáveis locais:

Exemplo 9.a – variáveis locais, declaração de cabeçalho de função (*main* acima da função)

```
1 //bibliotecas
2 #include<stdio.h>
3
4 //cabeçalhos de funções
5 void calculo(int,int);
6
7 main()
```

```
8 {
9     int a, b, x;
10    a = 10;
11    b = 20;
12    x = 500;
13    calculo( a, b );
14    printf("Valor de x em main() = %d", x);
15 }
16 void calculo(int a, int b)
17 {
18     int x;
19     x = a * b;
20     printf("Valor de x em calculo = %d\n\n", x);
21 }
```

Exemplo 9.b – variáveis locais, sem declaração de cabeçalho de função (*main* abaixo da função)

```
1 //bibliotecas
2 #include<stdio.h>
3
4 void calculo(int a, int b)
5 {
6     int x;
7     x = a * b;
8     printf("Valor de x em calculo = %d\n\n", x);
9 }
10 main()
11 {
12     int a, b, x;
13     a = 10;
14     b = 20;
15     x = 500;
16     calculo( a, b );
17     printf("Valor de x em main= %d", x);
18 }
```

Ambos os exemplos, 9.a e 9.b, produzirão o mesmo resultado. A tela abaixo apresenta a execução.

```
Valor de x em calculo = 200
Valor de x em main() = 500
-----
Process exited with return value 0
Press any key to continue . . . _
```

Exemplos com variáveis globais:

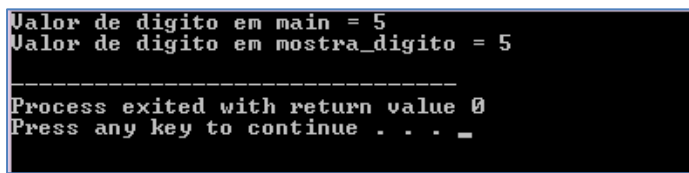
Exemplo 10.a – variáveis globais, declaração de cabeçalho de função (*main* acima da função)

```
1 //bibliotecas
2 #include<stdio.h>
3
4 //variáveis globais
5 int numero = 5;
6
7 //cabeçalhos de funções
8 void mostra_digito();
9
10 main()
11 {
12     printf("Valor de digito em main = %d\n", numero);
13     numero = 8;
14     mostra_digito();
15     printf("Valor de digito em main = %d\n", numero);
16 }
17 void mostra_digito()
18 {
19     printf("Valor de digito em mostra_digito = %d\n", numero);
20     numero = 14;
21 }
```

Exemplo 10.b – variáveis globais, sem declaração de cabeçalho de função (*main* abaixo da função)

```
1 //bibliotecas
2 #include<stdio.h>
3
4 //variáveis globais
5 int numero = 5;
6
7 void mostra_digito()
8 {
9     printf("Valor de digito em mostra_digito = %d\n", numero);
10 }
11 main()
12 {
13     printf("Valor de digito em main = %d\n", numero);
14     mostra_digito();
15 }
```

Ambos os exemplos, 10.a e 10.b, produzirão o mesmo resultado. A tela abaixo apresenta a execução.



```
Valor de digito em main = 5
Valor de digito em mostra_digito = 5

-----
Process exited with return value 0
Press any key to continue . . . _
```



## Exemplo 11 – variáveis globais, duas funções

```
1 //bibliotecas
2 #include<stdio.h>
3
4 //variáveis globais
5 int x;
6
7 void func1()
8 {
9     x=50;
10    printf("\n O conteudo de x eh %d", x);
11 }
12 void func2()
13 {
14     int x = 20;
15     printf("\n O conteudo de x eh %d", x);
16 }
17 main()
18 {
19     x=10;
20     func1();
21     func2();
22     printf("\n O conteudo de x eh %d", x);
23     x=100;
24     func2();
25     printf("\n O conteudo de x eh %d", x);
26 }
```

A tela abaixo demonstra a execução do exemplo 11.

```
0 conteudo de x eh 50
0 conteudo de x eh 20
0 conteudo de x eh 50
0 conteudo de x eh 20
0 conteudo de x eh 100
-----
Process exited after 0.6161 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Como func2() declarou uma variável x local, no momento da execução da função a referência à variável x será à local.

## EXERCÍCIOS (com ou sem envio de parâmetro(s); variáveis globais; variáveis locais)

- 19) **(Exercício resolvido)** Faça uma função para calcular a média aritmética de 4 notas (float; local) digitadas em *main*. A média deve ser armazenada numa variável global que será impressa em *main*.

**Solução:**

```
1 //bibliotecas
2 #include<stdio.h>
3
4 //variáveis globais
5 float med;
6
7 void media(float no1, float no2, float no3, float no4)
8 {
9     med = (no1+no2+no3+no4)/4;
10 }
11 main()
12 {
13     float n1, n2, n3, n4;
14     printf("Digite a primeira nota: ");
15     scanf("%f",&n1);
16     printf("\nDigite a segunda nota: ");
17     scanf("%f",&n2);
18     printf("\nDigite a terceira nota: ");
19     scanf("%f",&n3);
20     printf("\nDigite a quarta nota: ");
21     scanf("%f",&n4);
22     media(n1,n2,n3,n4);
23     printf("\n\nMedia das notas digitadas = %.1f", med);
24 }
```

Digite a solução do exercício e observe a execução.

- 20) Faça um programa para calcular a soma de dois números (int; local) lidos em *main*. Crie uma função para somar os números enviando-os como parâmetros. O resultado deverá ser guardado numa variável global. Imprima esse resultado em *main*.
- 21) Faça um programa que peça para o usuário digitar um número inteiro em *main* maior que 2 (declare a var como global). Chame uma função para imprimir na tela todos os números de 1 até este valor (não haverá envio de parâmetro).
- 22) Faça um programa que peça para o usuário digitar um número inteiro em *main* maior que 0 (declare a var como global). Chame uma função que imprimirá na tela o símbolo @ (arroba) 5 vezes em uma linha, na quantidade de linhas digitada na variável global. Por exemplo, se for digitado o número 3, a saída será (não haverá envio de parâmetro):

```
@@@@@
@@@@@
@@@@@
```

- 23) Faça um programa que peça para o usuário digitar um número inteiro em *main* maior que 0 (declare a var como global). Chame uma função que imprimirá na tela o símbolo \$ (cifrão) na quantidade de linhas digitada na variável global, seguindo o padrão mostrado no exemplo.

Exemplo: se for digitado o número 4, a saída será:

```
$  
$$  
$$$  
$$$$
```

### 3.6 PASSAGEM DE PARÂMETROS POR VALOR E POR REFERÊNCIA

A passagem de parâmetros para funções em C/C++ pode ser feita por valor ou por referência. Quando o parâmetro é passado por valor, a função manipula uma cópia do conteúdo armazenado no endereço de memória original da variável, de forma que quaisquer alterações sofridas dentro da função não se refletirão no conteúdo original da variável. Se, por sua vez, o parâmetro é passado por referência, a função recebe o endereço de memória da variável, não mais uma cópia de seu valor, e, portanto, o conteúdo original da variável sofrerá quaisquer alterações feitas dentro da função. A passagem por referência é feita utilizando-se um recurso da linguagem C chamada de ponteiro.

Exemplo de passagem de parâmetros por valor (sem ponteiro):

Exemplo 12 – passagem de parâmetro à função por valor

```
1 //bibliotecas  
2 #include <stdio.h>  
3  
4 void calculo(int a, int b)  
5 {  
6     a=17;  
7     b=23;  
8     printf("Valor de a e b em calculo = %d e %d\n", a,b);  
9 }  
10 main()  
11 {  
12     int a, b;  
13     a = 10;  
14     b = 20;  
15     printf("Valor de a e b em main = %d e %d\n", a,b);  
16     calculo( a, b );  
17     printf("Valor de a e b em main = %d e %d\n", a,b);  
18 }
```

A tela abaixo demonstra a execução do exemplo 12.

```
Valor de a e b em main = 10 e 20
Valor de a e b em calculo = 17 e 23
Valor de a e b em main = 10 e 20

-----
Process exited after 0.9529 seconds with return va
Pressione qualquer tecla para continuar. . .
```

🔑 O exemplo 12 e todos os demais exemplos que estudamos neste material até aqui, além de todos os exercícios que fizemos, utilizaram parâmetros POR VALOR.

Exemplos de passagem de parâmetros por referência (com ponteiro):

Exemplo 13 – passagem de parâmetro à função por referência (uso de ponteiros)

```
1 //biblioteca
2 #include <stdio.h>
3
4 void calculo(int *a, int *b) // o asterisco indica passagem por referência (ponteiro)
5 {
6     *a=17;
7     *b=23;
8     printf("Valor de a e b em calculo = %d e %d\n", *a,*b);
9 }
10 main()
11 {
12     int a, b;
13     a = 10;
14     b = 20;
15     printf("Valor de a e b em main = %d e %d\n", a,b);
16     calculo( &a, &b ); // o & (E comercial) indica envio por referência
17     printf("Valor de a e b em main = %d e %d\n", a,b);
18 }
```

O & (E comercial) significa que o endereço de memória da variável foi enviado como parâmetro à função, envio por referência.

A tela abaixo demonstra a execução do exemplo 13.

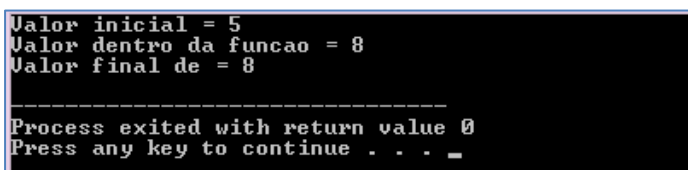
```
Valor de a e b em main = 10 e 20
Valor de a e b em calculo = 17 e 23
Valor de a e b em main = 17 e 23

-----
Process exited with return value 0
Press any key to continue . . .
```

## Exemplo 14 – passagem de parâmetro à função por referência (uso de ponteiros)

```
1 //bibliotecas
2 #include <stdio.h>
3
4 //cabeçalhos de funções
5 void muda_valor(int *);
6
7 main()
8 {
9     int cont = 5;
10    printf("Valor inicial = %d\n", cont);
11    muda_valor(&cont); // o & (E comercial) indica envio por referência
12    printf("Valor final de = %d\n", cont);
13 }
14 void muda_valor(int *a) // o asterisco indica passagem por referência (ponteiro)
15 {
16     *a += 3;
17     printf("Valor dentro da funcao = %d\n", *a);
18 }
```

A tela abaixo demonstra a execução do exemplo 14.



```
Valor inicial = 5
Valor dentro da funcao = 8
Valor final de = 8

-----
Process exited with return value 0
Press any key to continue . . . _
```

- ☞ Para alterar os valores das variáveis dentro da função, devemos passar o endereço da variável para a função. Dentro da função, devemos usar ponteiro.
- ☞ Em C/C++, por padrão, a passagem de parâmetros é sempre feita por valor, exceto quando se usa vetor/matriz (array).

## EXERCÍCIOS (com parâmetro por referência - ponteiro)

- 24) Faça um programa para calcular a média aritmética de 4 notas. Declare as variáveis que conterão as notas em main (float, local). Crie uma função onde serão digitadas as 4 notas. Crie outra função para fazer o cálculo da média. O resultado do cálculo da média deve ser apresentado na tela com uma casa decimal em main.
- 25) Faça um programa que declare 2 variáveis inteiras *a* e *b* locais em *main*, cujos valores serão digitados pelo usuário. Crie uma função com ponteiros que inverta os conteúdos das variáveis na memória, de forma que *a* passe a conter o valor de *b* e vice-versa. Apresente na tela em *main* a inversão. Exemplo: *a*=10 e *b*=20, inicialmente; no final: *a*=20 e *b*=10.

- 26) Faça um programa que contenha uma função para cálculo da equação do segundo grau,  $ax^2 + bx + c$ . Leia os valores dos coeficientes  $a$ ,  $b$  e  $c$ , faça a checagem necessária para que seja equação do segundo grau. Chame a função para calcular o delta e as raízes  $x_1$  e  $x_2$ . Envie como parâmetros as variáveis dos coeficientes, por valor, e duas outras variáveis para as raízes, por referência. Mostre em *main* as raízes calculadas na função.
- 27) Faça um programa com uma função para cálculo de fatorial. Leia uma variável (int, local) que terá o correspondente fatorial calculado na função, mas não utilize *return* para retornar o resultado do cálculo, a própria variável enviada deverá conter o resultado que será impresso em *main*.

## 4 BIBLIOTECA conio.c

A seguir, algumas funções da biblioteca *conio.c*.

### 4.1 FUNÇÃO gotoxy()

Para posicionamento de cursor na tela, nas coordenadas x (coluna) e y (linha) do plano cartesiano, iniciando em coluna e linha um.

Exemplo 15 – desenha moldura

```
1 #include <stdio.h>
2 #include <conio.c> //necessária para gotoxy()
3 void moldura (int ci, int li, int cf, int lf, char car)
4 {
5     for (int c=ci; c<=cf; c++)
6     {
7         gotoxy(c,li);printf("%c", car);
8         gotoxy(c,lf);printf("%c", car);
9     }
10    for (int l=li; l<=lf; l++)
11    {
12        gotoxy(ci,l);printf("%c", car);
13        gotoxy(cf,l);printf("%c", car);
14    }
15 }
16 main ()
17 {
18     moldura(1,1,80,24,'*');
19     moldura(10,5,70,19,'@');
20     getche();
21 }
```

### 4.2 FUNÇÃO Sleep()

Para retardar a execução de instruções do programa. Opera em milissegundos. Observe que o 'S' de Sleep é maiúsculo.

Exemplo 16 – retarda a impressão de cada caractere em 200 milissegundos

```
1 #include <stdio.h>
2 #include <conio.c> //necessária para Sleep()
3 void mostra ()
4 {
5     for (int c='a'; c<='h'; c++)
```

```
6      {
7          printf("%c\n\n", c);
8          Sleep(200); //em milissegundos; 's' é maiúsculo.
9      }
10 }
11 main ()
12 {
13     mostra();
14     getche();
15 }
```

### 4.3 FUNÇÃO textcolor()

Exibe os caracteres com a cor de texto escolhida.

As cores são representadas por números, que variam de 0 a 15 (Quadro 1), ou por constantes simbólicas, em letras maiúsculas, com os nomes das cores em inglês.

Código da cor	Cor
0	Preto
1	Azul
2	Verde
3	Verde-água
4	Vermelho
5	Roxo
6	Amarelo
7	Branco
8	Cinza
9	Azul-claro
10	Verde-claro
11	Verde-água-claro
12	Vermelho-claro
13	Lilás
14	Amarelo-claro
15	Branco-brilhante

Quadro 1 – Lista de códigos de cores



Exemplo 17.a – imprime CTI em letras coloridas usando códigos numéricos

```
1 #include <stdio.h>
2 #include <conio.c>
3 void troca_cor ()
4 {
5     textcolor(13); //lilás
6     printf("C\n");
7     textcolor(4); //vermelho
8     printf("T\n");
9     textcolor(6); //amarelo
10    printf("I\n");
11 }
12 main ()
13 {
14     troca_cor();
15     getche();
16 }
```

Exemplo 17.b – imprime CTI em letras coloridas usando constantes simbólicas

```
1 #include <stdio.h>
2 #include <conio.c>
3 void troca_cor ()
4 {
5     textcolor(GREEN);
6     printf("C\n");
7     textcolor(RED);
8     printf("T\n");
9     textcolor(YELLOW);
10    printf("I\n");
11 }
12 main ()
13 {
14     troca_cor();
15     getche();
16 }
```

#### 4.4 FUNÇÃO `textbackground()`

Exibe os caracteres com a cor de fundo escolhida.

As cores são representadas por números, que variam de 0 a 15 (Quadro 1, acima), ou por constantes simbólicas, em letras maiúsculas, com os nomes das cores em inglês.

Exemplo 18 – imprime CTI – UNESP com fundo azul-claro, letras branco-brilhante

```
1 #include <stdio.h>
2 #include <conio.c>
3 void troca_cor ()
4 {
5     textbackground(9); //azul-claro
6     textcolor(15); //branco-brilhante
7     printf("CTI - UNESP");
8 }
9 main ()
10 {
11     troca_cor();
12     getche();
13 }
```

#### 4.5 FUNÇÃO clrscr()

Limpa a tela ou pinta da cor definida em *textbackground()*, apagando qualquer texto visível.

Exemplo 19 – limpeza de tela com troca de cores de fundo

```
1 #include <stdio.h>
2 #include <conio.c>
3 void apaga_tela()
4 {
5     textbackground(9); //azul-claro
6     textcolor(15); //branco-brilhante
7     printf("limpeza de tela em 3 segundos...");
8     Sleep(3000);
9     textbackground(0); //preto
10    clrscr();
11 }
12 main ()
13 {
14     textbackground(14); //amarelo-claro
15     clrscr(); //limpa a tela pintando da cor definida acima
16     printf("Biblioteca conio.c\n");
17     Sleep(2000);
18     apaga_tela();
19     getche();
20 }
```

## 4.6 FUNÇÃO `clreol()`

Apaga o conteúdo da linha atual, a partir da posição do cursor até a margem direita da tela (até o final da linha), mantendo o cursor em sua posição inicial.

Exemplo 20 – limpeza de linha

```
1 #include <stdio.h>
2 #include <conio.c>
3 void texto(char *txt)
4 {
5     gotoxy(15,10);printf("%s",txt);
6     Sleep(2000);
7     gotoxy(20,10);clreol();
8 }
9 main ()
10 {
11     texto("Biblioteca conio.c - Linguagem C - FPD");
12     getche();
13 }
```

## 5 REFERÊNCIAS BIBLIOGRÁFICAS

CPLUSPLUS.COM. **The C++ resources network**. Disponível em  
<<http://www.cplusplus.com/doc/tutorial/>>. Acesso em jul/2011.

JAMSA, K. **Microsoft C, dicas, segredos e truques**. São Paulo: Makron Books, 1992, 701p.

MARSHALL, A. D. **Programming in C - UNIX system calls and subroutines using C**. Disponível em <<http://www.cs.cf.ac.uk/Dave/C/CE.html>>. Acesso em jul/2011.

MIZRAHI, V. V. **Treinamento em linguagem C, curso completo**. São Paulo: McGraw-Hill, 1990. (Módulo 1). 241p.

SCHILDT, H., **C Completo e Total**, 3ª ed. São Paulo: Makron Books Ltda, 1996. 827 p.

\_\_\_\_\_. São Paulo: McGraw-Hill, 1990. (Módulo 2). 273p.

Universidade Federal de Minas Gerais (UFMG). **Curso de Linguagem C**, Disponível em  
<<http://www.mtm.ufsc.br/~azeredo/cursoC/index.html>>. Acesso em jul/2011.