

Funções (Cap.5)

Introdução:

Programa na vida real:

- Muitas instruções;
- Divididos em módulos. Em C, esses módulos são chamados de **função**.

Objetivo das funções:

- Organizar o programa;
- Reutilizar rotinas;
- Evitar repetição do código.

Funções (Cap.5)

Exemplos de funções:

```
•main();  
•printf(...);  
•scanf(...);  
•system(...);  
•fopen(...);  
•fprintf(...);
```

Função Principal

Funções
de Bibliotecas da
linguagem C

Funções (Cap.5)

Exemplo: Biblioteca Matemática

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

Possui as seguintes funções:

sqrt(x) – raiz quadrada
exp(x) – e^x
log(x) – $\ln(x)$
log10(x) – $\log_{10}(x)$
fabs(x) – valor absoluto de x
ceil(x) – arredonda para o menor inteiro maior que x
floor(x) – arredonda para o maior inteiro menor que x
pow(x,y) – x elevado a potência y
fmod(x,y) – resto de x/y como número de ponto flutuante
sin(x) – seno x (em radianos)
cos(x) – cosseno x (em radianos)
tan(x) – tangente x (em radianos)

Funções (Cap.5)

Definição das funções

- Implementação de uma tarefa simples bem definida;
- Nome conciso de acordo com o objetivo.

Componentes de uma função

- Protótipo, Chamada, Argumentos e Retorno.

Estrutura de um Programa:

- inclusão das bibliotecas;
- protótipos (cabeçalho de especificação das funções);
- função principal - main();
- implementação das funções prototipadas.

Funções (Cap.5)

```
#include  
#include
```

```
int square
```

```
main ()  
{  
    int a=12,b;  
    b=square(a);  
    printf("O quadrado de %d eh %d.\n", a, b);  
    system("pause");  
}
```

```
int square(int y)  
{  
    return y*y;  
}
```

Função square

Função Principal

Funções (Cap.5)

Formato de uma função:

```
tipo_de_retorno nome (lista_de_argumentos)  
{  
    Declarações;  
    Instruções  
    Retorno Expressões (return);  
}
```

Alguns Tipos de argumento ou retorno:

```
void → vazio;  
int → inteiro (Padrão. Se o tipo não é explicitado,  
assume-se como int);  
float → ponto flutuante;  
char → caracter.
```

Funções (Cap.5)

Protótipo

Propósito: Informar ao compilador:

- tipo de dado retornado pela função;
- número de argumentos;
- tipo dos argumentos.

```
tipo nome_da_funcao(tipo_arg1,tipo_arg2);
```

```
//Exemplo 01
int  areaQuadrado(int lado);//protótipo

main()
{
.....
}

int  areaQuadrado(int lado)//Implementação
{
    int area = 0;//inicialização
    area= lado*lado;//ou pow(lado,2)
    return  area;
}
```

```
//Exemplo 02
void calculaPrecoProduto(int unidades);//protótipo

main()
{
    .....
}

//Implementação
void calculaPrecoProduto(int unidades)
{
    float preco;

    preco = unidades*1,45;

    printf("O preco eh %d.\n", preco);

}
```

```
//Exemplo 03
float retornaPi();//protótipo

main()
{
    .....
}

float retornaPi();//Implementação
{
    return      3.14;
}
```

Funções (Cap.5)

Palavra-Chave RETURN

Propósito: retornar o valor calculado por uma função

Ações:

- sai da função de forma imediata;
- retorna o valor informado (constante, variável ou expressão).

Atenção: nenhuma instrução que vier após o comando de *return* será executada. Ex.:

```
int    areaQuadrado(int lado)
{
    return lado*lado;
    lado = lado*2; //não será executado
}
```

Funções (Cap.5)

Biblioteca Padrão

- Arquivos de cabeçalho → “`__.h`”
- Contém: protótipos de função, definição de dados

Exemplos:

- **assert.h:** depuração do programa
- **ctype.h:** propriedades de caracteres (ex.: conversão de maiúsculos em minúsculos)
- **errno.h:** mensagem de erro
- **float.h:** limites para o ponto flutuante
- **math.h:** funções matemáticas
- **stdio.h:** funções de entrada e saída
- **stdlib.h:** função de conversão de números em texto, números aleatórios,...
- **string.h:** processamento de cadeias de caracteres (string)
- ... (limites, locale, setjmp, signal, stdarg, stddef, time)

Funções (Cap.5)

Chamadas de Funções

Passagem de Parâmetros em Funções

- **Chamada por valor** – é feita uma cópia do valor dos argumentos e essa cópia é passada à função. Ou seja, as modificações efetuadas na variável dentro da função, não alteram o valor original da variável.
- **Chamada por referência** – utiliza-se uma referência a própria variável na função (passagem do endereço da variável). Será estudada na aula 7 (ponteiros). Logo, modificações dentro da função refletem no valor real da variável em uso.

Funções (Cap.5)

Exemplo de chamada por valor

```
#include <stdio.h>
#include <stdlib.h>
int square (int)

main(){
    int a=12,b;
    printf("O valor inicial de a eh %d.\n", a);
    b=square(a);
    printf("O quadrado de %d eh %d.\n", a, b);
    printf("O valor atual de a eh %d.\n", a);
    system("pause");
}

int square(int y)
{
    return y*y;
}
```

O valor inicial de a eh 12.
O quadrado de 12 eh 144.
O valor atual de a eh 12.
Pressione qualquer tecla para continuar. . .

Passa-se o valor de "a" para a função.

"b" recebe o valor de "y" ao quadrado.

O valor de "a" não é alterado.

O valor de "a" é atribuído a "y".

Retorna-se o valor de "y" ao quadrado.

Funções (Cap.5)

Classes de Armazenamento:

Variáveis Globais:

Declaradas fora de qualquer função.
Qualquer função que venha após a declaração dela pode utilizá-la.

Atenção: Variáveis declaradas dentro da função *main* não são consideradas Globais.

Variáveis Locais:

Declaradas dentro de uma função. Sua declaração só tem validade dentro da função.

Funções (Cap.5)

Variáveis Globais:

```
float margemLucro = 0.1;

main ()
{
    float    CustoProduto = 5, valorVenda = 0;
    valorVenda = custoProduto + custoProduto*margemLucro;

    printf("O valor de Venda e de %d \n", valorVenda);
    system("pause");
}
```

Variável Global

Visível na
função *main*

Funções (Cap.5)

Variáveis Globais:

```
float calculaValorVenda(float CustoProduto);  
float margemLucro = 0.1; ← Variável Global  
  
main ()  
{  
    float      CustoProduto = 5, valorVenda = 0;  
  
    valorVenda = calculaValorVenda(CustoProduto);  
  
    printf("O valor de Venda e de %d \n", valorVenda);  
    system("pause");  
}  
  
float calculaValorVenda(float CustoProduto)  
{  
    return CustoProduto*margemLucro; ← Visível também dentro de uma função  
}
```

Funções (Cap.5)

Variáveis Locais

```
float calculaValorVenda(float CustoProd  
main ()  
{  
    float      CustoProduto = 5, valorVenda = 0;  
    valorVenda = calculaValorVenda(CustoProduto);  
  
    printf("A margem de lucro e de %d \n", margemLucro); //ERRADO  
    printf("O valor de Venda e de %d \n", valorVenda);  
    system("pause");  
}  
  
float calculaValorVenda(float CustoProduto)  
{  
    float      margemLucro = 0.1; ← Existe apenas dentro da função  
    return CustoProduto*margemLucro;  
}
```

Funções (Cap.5)

Variáveis Locais

```
float calculaValorVenda(float CustoProduto);
```

```
main ()  
{
```

```
    float    CustoProduto = 5, valorVenda = 0;
```

```
    valorVenda = calculaValorVenda(CustoProduto);
```

```
    float    margemLucro = 0.1; ←
```

```
    printf("A margem de lucro e de %d \n", margemLucro);
```

```
    printf("O valor de Venda e de %d \n", valorVenda);
```

```
    system("pause");
```

```
}
```

```
float calculaValorVenda(float CustoProduto)
```

```
{
```

```
    return CustoProduto*margemLucro;
```

```
}
```

A variável é local, e existe apenas dentro da função *main*

Logo, a variável não existe para esta função

Funções (Cap.5)

Especificadores de Classe de Armazenamento:

São usados para complementar a declaração de uma variável:

- **extern** variável explicitamente global
- **static** variável explicitamente estática
- **register** variável de acesso rápido
- **auto** variável explicitamente local

Esses especificadores serão usados para informar ao compilador de que forma a variável deve ser armazenada.

Forma geral:

```
especificador tipo nome_da_variável;
```

Funções (Cap.5)

Especificadores de Classe de Armazenamento:

auto (*default*) – a variável só existe dentro do bloco em que foi declarada. É criada no início do bloco de instruções ou de uma função, e então destruída no final;

Ex:

```
auto float x, y;
```

Esta forma economiza memória, pois essa classe de variável só existe enquanto for necessária. É o tipo padrão do C.

Funções (Cap.5)

Especificadores de Classe de Armazenamento:

register – mantém o conteúdo em um dos registradores de hardware de alta velocidade, mas, caso seja necessário utilizar o registrador, a variável poderá ser destruída;

Ex:

```
register float x, y;
```

Funções (Cap.5)

Especificadores de Classe de Armazenamento:

extern – são variáveis globais, sendo que essa classe é o caso default das variáveis declaradas fora das funções, ao usar uma variável global em uma função, declara-se a variável usando o termo **extern**.

Ex:

```
float x = 0, y = 1;
main ()
{
    extern float x, y;
    ...
}
```

Funções (Cap.5)

Especificadores de Classe de Armazenamento:

static – são variáveis locais que mantêm o valor quando a função é encerrada. Dessa forma, caso a função seja reutilizada, a função utilizará o valor antigo da variável.

Ex:

```
static int cont;
```

Funções (Cap.5)

Recursividade

A recursão ocorre quando uma função chama a si mesma. A função é *recursiva* se um comando no corpo da função a chama.

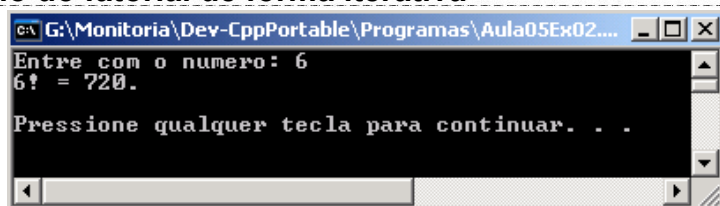
O cálculo do fatorial de um número pode ser feito de forma recursiva.

Funções (Cap.5)

Cálculo do fatorial de forma iterativa

```
main()
{
    int fatorial=1, contador, numero;

    printf("Entre com o numero: ");
    scanf("%d", &numero);
    for(contador=numero; contador>=1; contador--)
        fatorial*=contador;
    printf("%d! = %d.\n\n", numero, fatorial);
    system("pause");
}
```

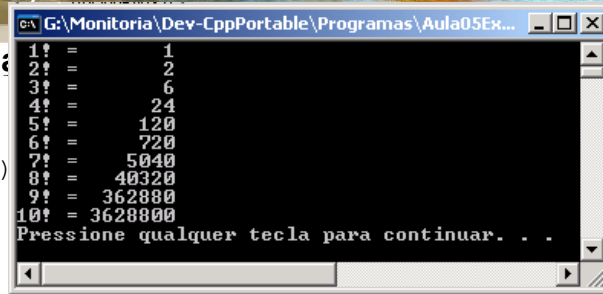


Funções (Cap.5)

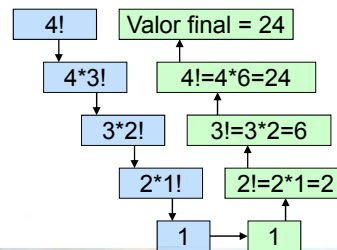
Cálculo do fatorial

```
long fatorial(long n)
{
    if (n > 1) return n * fatorial(n-1);
    else return 1;
}

main()
{
    int cont;
    for(cont=1; cont<=10; cont++)
        printf("%2d! = %7d\n", cont, fatorial(cont));
    system("pause");
}
```



```
G:\Monitoria\Dev-CppPortable\Programas\Aula05Ex...
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
Pressione qualquer tecla para continuar. . .
```



Funções (Cap.5)

Erros comuns em programação:

Erro no protótipo da função:

Retornar um tipo de valor diferente do tipo declarado.

```
void half (float x);
{
    ...
    return (x/2)
}
```

Neste caso a função não deveria retornar nenhum valor, provocando um erro de sintaxe.

Observação: caso o tipo de retorno não seja declarado o compilador irá considerar o retorno do tipo inteiro.

Funções (Cap.5)

Erros comuns em programação

Erro ao definir os parâmetros da função:

Ao declarar parâmetros da função de mesmo tipo é comum declarar: `(float x, y)` em vez de `(float x, float y)`. Isto causa erro de sintaxe pelo fato de `(float x, y)` não ser o mesmo que `(float x, int y)` na definição dos parâmetros da função.

```
int função(float x, y)≠int função(float x, float y)  
int função(float x, y)=int função(float x, int y)
```

Funções (Cap.5)

Exercícios (livro)

- 1) Exemplo de Fatorial (pag. 142)
- 2) Fibonacci (pag. 145)
- 3) 5.8 (pag. 158)
- 4) 5.17
- 5) 5.24 (item a,b)
- 6) 5.39
- 7) 5.47

Funções (Cap.5)

Desafio: Exercício 5.39 do Livro Texto, pg. 162.
Torres de Hanói

