

# FUNÇÕES - Parte I

Modularização de programas, passagem de parâmetros por valor e referência.



# FUNÇÕES

- As linguagens de programação têm à sua disposição várias funções pré-definidas:
- EXEMPLO (Linguagem C)
  - `pow(X,2)` quadrado de X
  - `sqrt (X)` raiz quadrada de X
  - `sin (X)` seno de X

# FUNÇÕES

- As linguagens de programação têm à sua disposição várias funções pré-definidas:
- EXEMPLO (*Linguagem c*)
  - pow (X,2) quadrado de X
  - sqrt (X) raiz quadrada de X
  - sin (X) seno de X



Identificador  
da  
FUNÇÃO

# FUNÇÕES

- As linguagens de programação têm à sua disposição várias funções pré-definidas:
- EXEMPLO (*Linguagem C*)
  - `pow (X,2)` quadrado de X
  - `sqrt (X)` raiz quadrada de X
  - `sin (X)` seno de X



**Parâmetro Formal**

# FUNÇÕES

- As Funções Pré Definidas podem ser usadas diretamente em expressões:
- Exemplo:  
$$h = \text{sqrt}(a + \text{pow}(y, 2) + 2 * \text{sin}(y))$$

# FUNÇÕES

- As Funções Pré Definidas podem ser usadas diretamente em expressões:
- Exemplo:  
h= **sqrt** (a + **pow** (y,2) + 2 \* **sin** (y))



Identificadores das  
FUNÇÕES

# FUNÇÕES

- As Funções Pré Definidas podem ser usadas diretamente em expressões:
- Exemplo:

`h = sqrt (a + pow (y,2) + 2 * sin (y))`

**Parâmetros  
float/double**

# FUNÇÕES

- As Funções Pré Definidas podem ser usadas diretamente em expressões:
- Exemplo:  
h= sqrt (a + pow(y,2) + 2 \* sin (y))

Parâmetros  
double/float

Estes parâmetros  
devem ser valores  
conhecidos

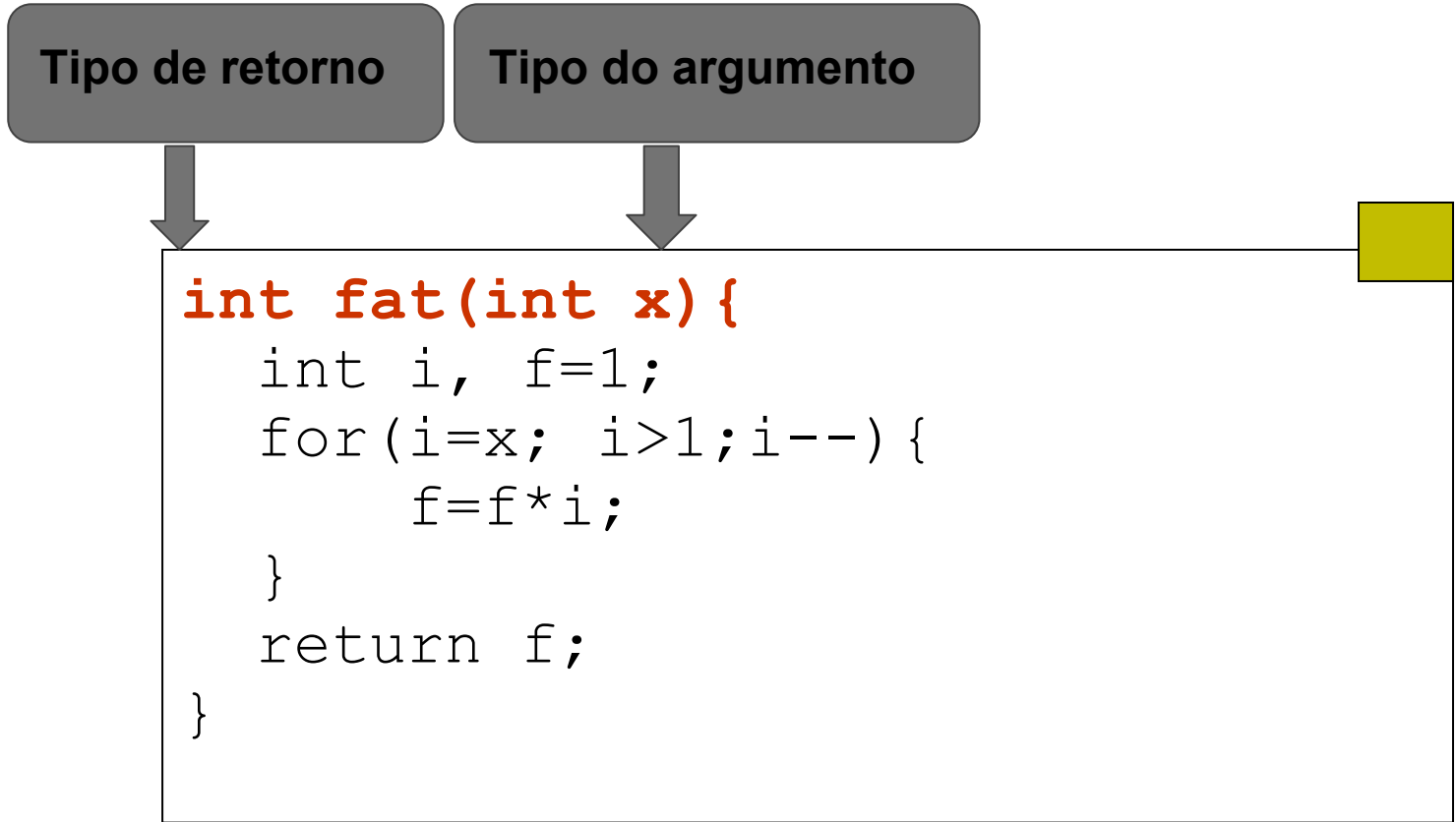


# FUNÇÕES

Se houver necessidade o programador pode **definir** suas próprias **FUNÇÕES**

Tipo de retorno

Tipo do argumento



The diagram consists of two grey rounded rectangular boxes at the top. The left box contains the text 'Tipo de retorno' and the right box contains 'Tipo do argumento'. Two grey arrows point downwards from these boxes to a larger white rectangular box with a black border. This white box contains a C++ function signature and body. The return type 'int' is highlighted in red. A yellow square is positioned at the top right corner of the white box, with a horizontal line extending from its left side to the right edge of the box.

```
int fat(int x){  
    int i, f=1;  
    for(i=x; i>1;i--){  
        f=f*i;  
    }  
    return f;  
}
```

# Return

- As funções retornam um resultado que deve ser do mesmo tipo para o qual a função foi declarada anteriormente.

**<tipo\_retornado>** <nome\_função>( <lista\_dos\_parametros> )

```
int fatorial (int n)
```

- O comando **return** é responsável por encerrar a execução da função e retornar o valor daquele tipo.

```
return product;
```

# Return

- Se um tipo não é especificado para uma função, o tipo `int` será o *default* da função.

```
int all_add( int a, int b)
{
    int c;
    ....
    return (a+b+c);
}
```



```
all_add( int a, int b)
{
    int c;
    ....
    return (a+b+c);
}
```

# Return

- O valor retornado pela função é convertido, se necessário, para o tipo adequado.

```
float add( int a, int b)
{
    int soma;
    soma = a+b;
    return soma;
}
```

# Return

- Recomenda-se limitar a função para que tenha um único **return** visando facilitar a compreensão da função pelo programador.
- Todavia, o uso de mais que um **return** também pode tornar o código mais legível.
- Desta forma, a quantidade de **return** em uma função deve facilitar o entendimento e a manutenção do código.

# Return

- Exemplo:

```
double absolute_value(double x)
{
    if(x>0.0)
        return x;
    else
        return -1*x;
}
```

# UTILIZAÇÃO DA FUNÇÃO FATORIAL - Exemplo

- Dados dois números N e K, calcular a Combinação

$$C_{N,K} = N! / K!(N-K)!$$



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

int fat (int x){
    int i,f=1;
    for(i=x;i>1;i--){
        f=f*i;
    }
    return f;
}

int main(){
    int n,k;
    float c;
    printf("FORNECA O VALOR DE N: "); scanf("%d",&n);
    printf("FORNECA O VALOR DE K: "); scanf("%d",&k);
    c= (float) fat (n) / (fat (k) * fat (n-k));
    printf("C(%d,%d)=%.2f",n,k,c);

return 0;
}
```

# Declarações de funções

- Um protótipo da função indica ao compilador o número e o tipo de argumentos que devem ser passados para a função e o tipo de valor que deve ser retornado pela função.

**<tipo\_retornado>** <nome\_função>(<lista\_dos\_parametros>);

- A lista dos parâmetros apresenta os tipos separados por vírgula, onde os identificadores são opcionais.

float func(int, float); ⇔ float func(int x, float y);

# Declarações de funções

- Se uma função, por exemplo, `func(x)` é chamada antes de sua declaração, definição ou protótipo, o compilador assume a declaração abaixo como default

```
int func();
```

- A maioria dos compiladores precisa conhecer os tipos de retorno e os parâmetros, antes que o programa principal faça uma chamada à sub-rotina.

# Declarações de funções

- Programas maiores teriam grande quantidade de linhas inicialmente dedicadas às funções e procedimentos até se chegar à função `main()`.
- Além disso, ficaria cada vez mais difícil manter as funções na ordem correta de acordo com sua chamada pelo programa principal.
- A linguagem C permite que os protótipos de funções sejam declarados antes do programa principal.

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

int fat (int);

int main(){
    int n,k;
    float c;
    printf("FORNECA O VALOR DE N: "); scanf("%d",&n);
    printf("FORNECA O VALOR DE K: "); scanf("%d",&k);
    c= (float) fat(n)/(fat(k)*fat(n-k));
    printf("C(%d,%d)=%.2f",n,k,c);

return 0;
}

int fat (int x){
    int i,f=1;
    for(i=x;i>1;i--){
        f=f*i;
    }
    return f;
}
```

# Void

- As sub-rotinas na linguagem C podem ser utilizadas todas como funções.
- A palavra reservada **void** na declaração de uma sub-rotina, indica que se trata de uma função que não retorna nenhum valor.
- O uso de **void** ao invés de uma lista de parâmetros, indica que a função não utiliza argumentos (lista de parâmetros).
- Os trechos abaixo são equivalentes.  
$$\text{void func()} \Leftrightarrow \text{void f(void)}$$

# Void

- Exemplos:

```
void nadafaz(void) { }
```

```
void wrt_endereço(void){  
    printf(“%s\n%s\n%s\n%s\n%s\n\n”,  
        “          *****”,  
        “          ***  SANTA CLAUS  *”,  
        “          ***  NORTH POLE  *”,  
        “          ***  EARTH          *”,  
        “          *****”);  
}
```

# Programa para verificar se um número é primo

```
#define<stdio.h>
#define<stdlib.h>
```

```
int primo(int);
```

```
int main(){
```

```
    int n;
```

```
    printf("forneca o numero: "); scanf("%d",&n);
```

```
    if (primo(n))
```

```
        printf("%d eh primo",n);
```

```
    else
```

```
        printf("%d nao eh primo",n);
```

```
    return 0;
```

```
}
```

```
int primo(int n){
```

```
    int c, i;
```

```
    i=2;
```

```
    c=1;
```

```
    while((i<n) && (c)){
```

```
        if(n%i == 0)
```

```
            c=0
```

```
        else
```

```
            i = i+1;
```

```
    return c;
```

```
}
```

Declaração da função

Definição  
da  
Função



# Passagem de parâmetros por referência em funções

- O endereço de memória da variável é fornecido à função e não uma cópia do valor da variável.
- Qualquer alteração executada pela função ocorre na posição de memória fornecida.
- Portanto, as alterações permanecem quando a função é encerrada.
- Ponteiros, são usados nas passagens por referência.

# Passagem de parâmetros por referência em funções

## ■ Vetores e matrizes

- São passados sempre por referência.
- Os colchetes, após o nome do vetor passado a uma função, indicam que o parâmetro é um vetor. Não precisa especificar o tamanho.

```
int soma_vetor(int vetor[], int elementos)
```

# Passagem de parâmetros por referência em funções

## ▪ Vetores e matrizes

- Exemplo: Formas possíveis de se declarar a função func() para receber o vetor int vet[100].

```
func(int x[100]);
```

```
func (int x[]);
```

```
func(int *x);
```

# Passagem de parâmetros por referência em funções

## ▪ Vetores e matrizes

- Ao passar uma matriz bidimensional, se for preciso acessar entradas específicas, o número de colunas precisa ser fornecido.


```
int soma_matriz(int matriz[][5], int linhas)
```


- Se não há necessidade de acessar entradas específicas da matriz, ela poderá ser tratada como um vetor. Nesse caso, o número de elementos da matriz deverá ser fornecido.

```

1  #include<stdio.h>
2  #define LN  2
3  #define CL  5
4
5  int soma_vetor(int[], int);
6  int soma_matriz(int , int colunas,int matriz[][colunas]);
7
8  void main(){
9      int vetor[CL]={10, 20, 30, 40, 50};
10
11     int bidim[LN][CL]={{10,20,30,40,50},
12                        {60,70,80,90,100} };
13     printf("Soma dos valores do vetor:%d\n", soma_vetor(vetor,CL));
14     printf("Soma dos valores da matriz:%d\n\n", soma_matriz(LN,CL,bidim));
15     printf("Soma dos valores do matriz:%d\n", soma_vetor(bidim, LN*CL));
16 }
17
18 int soma_vetor(int vetor[], int elementos){
19     int i,soma;
20
21     soma=0;
22     for(i=0; i<elementos; i++)
23         soma += vetor[i];
24
25     return soma;
26 }
27
28 int soma_matriz(int linhas, int colunas,int matriz[][colunas]){
29     int i,j,soma;
30
31     soma=0;
32     for(i=0; i<linhas; i++)
33         for(j=0; j<colunas; j++)
34             soma += matriz[i][j];
35
36     return soma;
37 }

```





**Soma dos valores do vetor:150**  
**Soma dos valores da matriz:550**  
**Soma dos valores do matriz:550**

# Exercícios

1-Escreva um programa que calcule e imprima o produto de duas matrizes geradas aleatoriamente com valores inteiros, onde:

- Uma função deve ser responsável pela geração aleatória dos valores. O usuário fornece o intervalo dos valores [Min;Max].
- Uma função deve ser responsável pelo cálculo do produto.
- Uma função deve ser responsável pela impressão das matrizes.

2-Escreva um programa para executar as tarefas a partir do menu descrito abaixo:

1- $x^y$ ;

2-x é primo?

3-x!

4-MMC(x,y)

Digite a opção desejada: