

Ministério da Educação  
Universidade Federal de Santa Maria  
Curso Técnico em Informática para Internet Integrado ao Ensino Médio  
Colégio Técnico Industrial de Santa Maria - CTISM



# Algoritmos e Programação

luciana.lourega@ufsm.br





# Funções

- Porque usar funções?
- Para permitir o reaproveitamento de código já construído (por você ou por outros programadores);
- Para evitar que um trecho de código que seja repetido várias vezes dentro de um mesmo programa;
- Com o uso de uma função é preciso alterar apenas **dentro** da função que se deseja;
- Para que os blocos do programa não fiquem grandes demais e, por consequência, mais difíceis de entender;
- Para separar o programa em partes(blocos) que possam ser logicamente compreendidos de forma isolada.

# Funções



- Formato geral de uma função em C:
- **tipo\_da\_funcao NomeDaFuncao (lista\_de\_parametros)**  
**{**  
    **// corpo da função**  
**}**
- A **Lista\_de\_Parametros**, também é chamada de **Lista\_de\_Argumentos**, é opcional.

A hand is pointing at a digital interface with a blue background and hexagonal patterns. Several icons are visible: a shopping cart, a padlock, and a group of people. The word 'Funções' is written in large blue letters.

# Funções

- O tipo da função pode ser qualquer um dos mostrados até agora e representa o tipo do dado que é retornado pela função.
- Caso a função não retorne nenhum valor dizemos que ela é do tipo **void**.
- E caso não seja especificado nenhum tipo, por padrão a função retorna um inteiro.



# Funções

- Entre parênteses estão os parâmetros da função que é constituído pelos nomes das variáveis que se deseja passar para função separados por vírgulas e acompanhados de seus respectivos tipos.
- No caso da função não conter parâmetros a lista de parâmetros será vazia, mas mesmo assim será necessário utilizar os parênteses.



# Funções

- No corpo da função vem o código em C, este código é privativo da função, ou seja, nenhuma outra função poderá acessá-la com nenhum comando, exceto por meio de uma chamada a função.
- Isso quer dizer que o código da função não pode afetar outras partes do programa, a menos que sejam utilizadas variáveis Globais.





# Funções

- Isto porque as variáveis contidas em uma função são locais, só existem naquela função. Essas variáveis são criadas quando entram na função e destruídas ao sair.
- Para entender melhor o uso de funções vamos ver um exemplo abaixo.
- Este programa calcula o fatorial de um número qualquer.



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

```
include<conio.h>
```

```
/*Este programa calcula o fatorial de um número determinado pelo usuário*/
```

```
int CalculaFatorial (int x); /*protótipo da função*/
```

**/\*Com o protótipo é possível que o compilador verifique se existe erros nos tipos de dados entre os argumentos usados para chamar uma função e a definição de seus parâmetros.**

- Além de verificar se a quantidade de argumentos é igual a quantidade de parâmetros, caso contrário, causará erros na execução do programa\*/**



# Funções

A hand is pointing at a digital interface with hexagonal icons. The icons include a shopping cart, a person, and a group of people. The background is blue with a hexagonal pattern.

```
void main(){  
    int Num,Fatorial; /*variáveis locais só podem ser acessadas dentro da função principal.*/  
    printf("Digite o numero o qual deseja saber o fatorial");  
    scanf("%d",&Num);  
    Fatorial=CalculaFatorial(Num);
```

/\*Chama a função que calcula o fatorial. O valor retornado pela função é atribuído para a variável "fatorial", que deve ser do mesmo tipo do que o tipo de dados que a função retorna. E o argumento Num é o parâmetro a ser passado para a função e deve ser do mesmo tipo do argumento x da função \*/

# Funções

- /\*Qualquer modificação no seu valor é feito apenas na variável x, não alterando o valor da variável usada na chamada \*/

```
printf("Fatorial: %d", Fatorial); /*Imprime o valor na tela*/
```

```
getch(); /*Retorna após pressionar uma tecla*/  
}
```

```
int CalculaFatorial(int x){
```

```
/*Esta função recebe como parâmetro o valor de x que é o número que o usuário digitou  
(Num) e retorna o fatorial desse número que é um inteiro, portanto a função retorna  
um dado do tipo inteiro.*/
```



# Funções

- `int cont,Fat;`
  - Essas variáveis são locais pois só podem ser acessadas dentro desta função, aqui elas são criadas e após a execução da função elas são destruídas.

```
Fat=1;  
for(cont=0;cont<x;cont++)  
    Fat=Fat*cont;  
return (Fat); /*Retorna o valor calculado e atribuído a variável Fat*/  
}
```

A hand is pointing at a digital interface with a blue background and hexagonal patterns. Several icons are visible within the hexagons: a shopping cart, a padlock, and a group of three people. The word 'Funções' is written in large blue letters across the top of the image.

# Funções

- O comando `return` usado no final da função é o responsável por retornar um valor para a instrução que o chamou, neste caso para a variável `Fatorial`. Que por sua vez é impressa na tela.
- Exemplo:
- `#include <stdio.h>`
- `#include <conio.h>`
- `float sqr (float num);`

# Funções



```
int main ()
{
    float num,sq;
    printf ("Entre com um numero: ");
    scanf ("%f",&num);
    sq=sqr(num);
    printf ("\n\nO numero original e: %f\n",num);
    printf ("O seu quadrado vale: %f\n",sq);
    getch();
}

float sqr (float num)
{
    num=num*num;

    return num;
}
```



# Funções

- Parâmetros
  - A fim de tornar mais amplo o uso de uma função, a linguagem C permite o uso de parâmetros.
  - Estes parâmetros possibilitam que se definida sobre quais dados a função deve operar.
  - Parâmetros são valores que as funções recebem da função que a chamou. Portanto, os parâmetros permitem que uma função passe valores para outra.





# Funções

- Parâmetros podem ser passados para as funções de duas maneiras:
  - Passagem de parâmetro por valor
  - Passagem de parâmetro por referência.
- Na passagem por valor uma cópia do valor do argumento é passado para a função.
- No exemplo a seguir as variáveis  $x$  e  $t$  não ocupam a mesma posição de memória.

# Funções

- Modificações feitas em x não alteram o valor de t, da mesma forma que modificações feitas em t não alteram o valor de n.

```
sqr (int x){  
    x=x*x;  
    return(x);  
}
```

```
main (){  
    int t=10;  
    printf(“%d %d”,sqr(t)t); //saída do programa 100 e 10.  
}
```

A hand is pointing at a digital interface with a blue background and hexagonal patterns. Several icons are visible: a shopping cart, a padlock, and a group of people. The word "Funções" is written in large blue letters.

# Funções

- `int x=5, y=3, z;`
- `z = adicao ( x , y );`
- O que fizemos nesse caso foi chamar a função **adicao** passando os valores de **x** e **y**, que significam **5** e **3** respectivamente, e não as próprias variáveis.

# Funções

```
int addition (int a, int b)
```

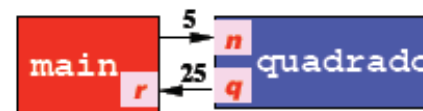
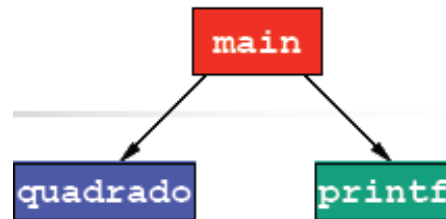
```
z = addition ( 5 , 3 );
```

- Dessa maneira, quando a função **adicao** está sendo chamada, o valor de suas variáveis **a** e **b** tornam-se **5** e **3** respectivamente, mas qualquer modificação de **a** ou **b** dentro da função **adicao** não afetará os valores de **x** e **y** fora dela, porque as variáveis **x** e **y** não foram passadas para a função, somente seus valores.

# Funções

```
#include <stdio.h>
int quadrado(int);
main()
{
    int r;
    r = quadrado(5);
    printf("Aqui vai: %d",r);
}

int quadrado(int n)
{
    int q;
    q = n * n;
    return q;
}
```





# Funções

- **Na passagem de parâmetro por referência:**
  - Ocorre quando alterações nos parâmetros formais, dentro da função, alteram os valores dos parâmetros que foram passados para a função.
  - Este nome vem do fato de que, neste tipo de chamada, não se passa para a função os valores das variáveis, mas sim suas referências (a função usa as referências para alterar os valores das variáveis fora da função).



A hand is shown pointing at a digital interface. The interface features a blue background with a pattern of hexagons. Some hexagons contain white icons: a shopping cart, a padlock, and a group of three people. The word "Funções" is written in large, bold, blue letters across the middle of the image.

# Funções

- Quando queremos alterar as variáveis que são passadas para uma função, nós podemos declarar seus parâmetros formais como sendo *ponteiros*.
- Os ponteiros são a "referência" que precisamos para poder alterar a variável fora da função.
- O único inconveniente é que, quando usarmos a função, teremos de lembrar de colocar um **&** na frente das variáveis que estivermos passando para a função. Veja um exemplo:

# Funções

- Exemplo

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

```
void Swap (int *a, int *b); //protótipo da função
```

```
int main (void)
```

```
{
```

```
int num1,num2;
```

```
num1=100;
```

```
num2=200;
```

```
Swap (&num1,&num2);
```

```
printf ("\n\nEles agora valem %d %d\n",num1,num2);
```

```
getch();}
```

# Funções

```
void Swap (int *a,int *b)  
{
```

```
    int temp;
```

```
    temp=*a;
```

```
    *a=*b;
```

```
    *b=temp;
```

```
}
```

```
//temp=100 *a=200 *b=100
```

A hand is pointing at a digital interface with a blue background and hexagonal patterns. Several icons are visible: a shopping cart, a person, and a group of people. The word "Funções" is written in large blue letters.

# Funções

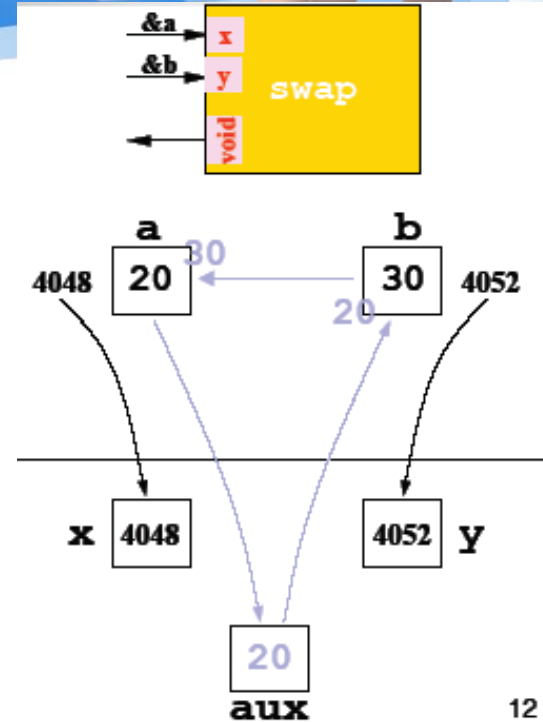
- Não é muito difícil. O que está acontecendo é que passamos para a função Swap o endereço das variáveis num1 e num2.
- Estes endereços são copiados nos ponteiros a e b. Através do operador \* estamos acessando o conteúdo apontado pelos ponteiros e modificando-o.
- Mas, quem é este conteúdo? Nada mais que os valores armazenados em num1 e num2, que, portanto, estão sendo modificados!

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

```
//void swap(int*, int*);
```

```
void swap(int *x, int *y)
{
    int aux;
    aux=*x; *x=*y; *y=aux;
    return;
}
```

```
main()
{
    int a=20, b=30;
    swap(&a,&b);
    printf("a=%d e b=%d",a,b);
    getch();
    return(0);
}
```





# Funções

- **Vetores e Matrizes como parâmetro:**
  - Vetores e matrizes são sempre passados por referência na linguagem C.
  - Quando declaramos um vetor em C, a variável correspondente a esse vetor é uma referência à região de memória onde o mesmo é armazenado.
  - Ao aplicarmos um índice a esse vetor, é que estamos nos referindo a um elemento.



# Funções



```
void ordena(int v[],int n) {  
    int i,k;  
    do  
    {  
        k = 0;  
        for(i=0; i < n-1; i++)  
            if(v[i] > v[i+1])  
            {  
                troca(&v[i],&v[i+1]);  
                k++; }  
    }  
    while(k!=0);  
}
```

# Funções

A hand is shown pointing at a digital interface. The interface features a blue background with a pattern of hexagons. Some hexagons contain white icons: a person, a shopping cart, and a group of people. The word 'Funções' is written in large, bold, blue letters across the top of the image.

- No exemplo acima, " $v[]$ " denota um vetor, cujo tamanho não é fixado na função, já que o mesmo depende do parâmetro efetivo.
- O segundo parâmetro, " $n$ " indica o tamanho do parâmetro efetivo e deve ser preenchido ao se chamar a função.
- O exemplo abaixo mostra alguns usos da função "ordena".

# Funções



```
... int a[10];  
... ordena(a,10);
```

//função que retorna o somatorio dos elementos de um vetor de n elementos

```
int somatorio(int v[], int n)  
{  
    int i,soma=0;  
    for(i=0;i<n;i++){  
        soma=soma +v[i];  
    }  
    printf("soma:\n%d",soma);  
    return soma;  
}
```

# Funções



```
int main()
{
    int v[10],result,i;
    for(i=0;i<10;i++)
        v[i]=i;
    result= somatorio(v,10);
    getch();
}
```

# Funções

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

```
void imprime (int vet[])
```

```
{
```

```
    int i;
```

```
    for (i=0; i < 5; i++)
```

```
        printf ("%d ", vet [i]);
```

```
        printf ("\n\n");
```

```
}
```

Não é preciso informar o tamanho do índice...

...mas deve-se tomar cuidado na hora de manipular ao vetor, pois caso o programa "tente" acessar um índice que não existe, o resultado será indesejado.

```
int main (void)
```

```
{
```

```
    int matriz1 [5] = {1, 2, 3, 4, 5};
```

```
    imprime_1(matriz1);
```

```
}
```

# Funções



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

```
void funcao(int vetor[]){
    int soma = 0;

    for (int i = 0; i < 10; i++){
        soma += vetor[i];
        printf("soma = %d", soma);
    }
}
```



# Funções

- ```
int main(void){  
    int vetor[10];  
  
    //apenas inicializacao do conteudo  
    for (int i = 0; i < 10; i++)  
        vetor[i] = i;  
  
    //chama sua função passando o vetor como parâmetro  
  
    //note que não é necessário o uso explícito de um ponteiro  
  
    funcao(vetor);  
}
```

# Funções

A hand is shown pointing at a digital interface. The interface features a blue background with a pattern of hexagons. Some hexagons contain white icons: a shopping cart, a group of people, and a person with a checkmark. The hand is positioned on the right side of the frame, with the index finger pointing towards the center.

- Quando vamos passar um vetor como argumento de uma função, podemos declarar a função de três maneiras equivalentes. Seja o vetor:

```
int matr[x] [50];
```

- e que queiramos passá-la como argumento de uma função **func()**. Podemos declarar **func()** das três maneiras seguintes:

```
void func (int matr[x][50]);
```

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

```
void func (int *matr[x]);
```

A hand is shown pointing at a digital interface. The interface features a blue background with a pattern of hexagons. Some hexagons contain white icons: a shopping cart, a padlock, and a group of three people. The word "Funções" is written in large, bold, blue letters across the middle of the image.

# Funções

- Nos três casos, teremos dentro de **func()** um **int\*** chamado **matrix**.
- Ao passarmos um vetor para uma função, na realidade estamos passando um ponteiro.
- Neste ponteiro é armazenado o endereço do primeiro elemento do vetor.



# Funções

- Passagem de matriz como parâmetro da função:
  - *tipo retorno nome(tipo m[][dim2],...)*
- *dim2* deve ser fornecido para que o compilador possa calcular o deslocamento em bytes em relação ao endereço do primeiro elemento para uma determinada posição.
- *//função que calcula a soma de duas matrizes de n linhas e m colunas.*

# Funções




```
void soma(int A[][10],int B[][10], int C[][10], int n,int m)
{
    int i,j;
    for(i=0;i<n;i++)
        for (j=0;j<m;j++)
            C[i][j]= A[i][j]+B[i][j];
}
```

# Funções



- ```
void funcao(int matriz[][10])  
{  
  
    int i;  
    int j;  
  
    //zerando a matriz  
    for (i = 0; i < 10; i++)  
        for(j = 0; j < 10; j++)  
            matriz[i][j] = 0;  
}
```



```
int main()
{
    int matriz[10][10];
    int i;
    int j;
```

//chama sua função passando a matriz como parâmetro

//note que não é necessário o uso explícito de um ponteiro

```
funcao (matriz);

    for (i = 0; i < 10; i++){
        for(j = 0; j < 10; j++)
            printf("%d ", matriz[i][j]);
        printf("\n");
    }
    getch();}
```