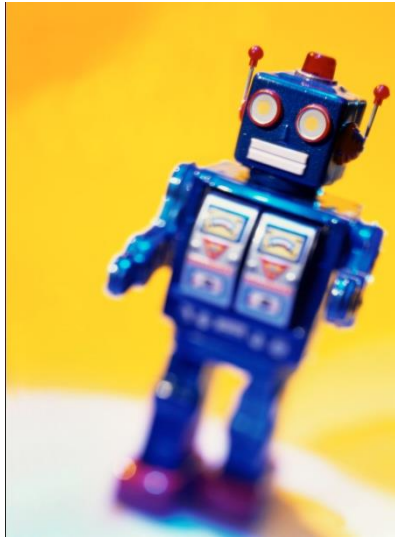


Funções



É uma técnica de reaproveitamento e organização de código.

Funções

- Pode receber alguns parâmetros e devolver um resultado.
- É chamada pelo seu nome, seguido de parênteses contendo os parâmetros:

$f(x, y, \dots)$

Funções

Sintaxe para definição/criação de funções:

```
Tipo_retornado NOME_DA_FUNCAO (PARÂMETROS ){  
  
    BLOCO DE COMANDOS  
  
}
```

Funções

```
Tipo_retornado NOME_DA_FUNCAO (PARÂMETROS ){  
    BLOCO DE COMANDOS  
}
```

Onde:

- **Tipo_retornado** é o tipo de valor que será retornado pela função.
- **NOME_DA_FUNCAO** é como aquele BLOCO DE COMANDOS será conhecido dentro do programa.
 - ✓ Para definir o nome da função, valem as mesmas regras para definir variáveis.
- **PARÂMETROS:** são variáveis separadas por vírgulas.

Funções

```
Tipo_retornado NOME_DA_FUNCAO (PARÂMETROS ){  
    BLOCO DE COMANDOS  
}
```

Onde:

BLOCO DE COMANDOS pode conter :

- sequência de declarações de variáveis e constantes;
- sequência de comandos: comandos condicionais, de repetição, chamada de outras funções, etc.
- Pelo menos um ponto de saída com o comando *return*:
***return* VALOR_A_RETORNAR**

Obs: funções do tipo void não precisam utilizar o comando return.

Exemplo de função

// Função que retorna o valor do fatorial do número dado.

```
int fatorial (int n){  
    int i, fat = 1;  
    for (i =1; i<=n ; i ++)  
        fat = fat * i ;  
    return fat;  
}
```

Como ocorre a chamada da função?

```
int quadrado(int a){  
    return(a*a);  
}
```

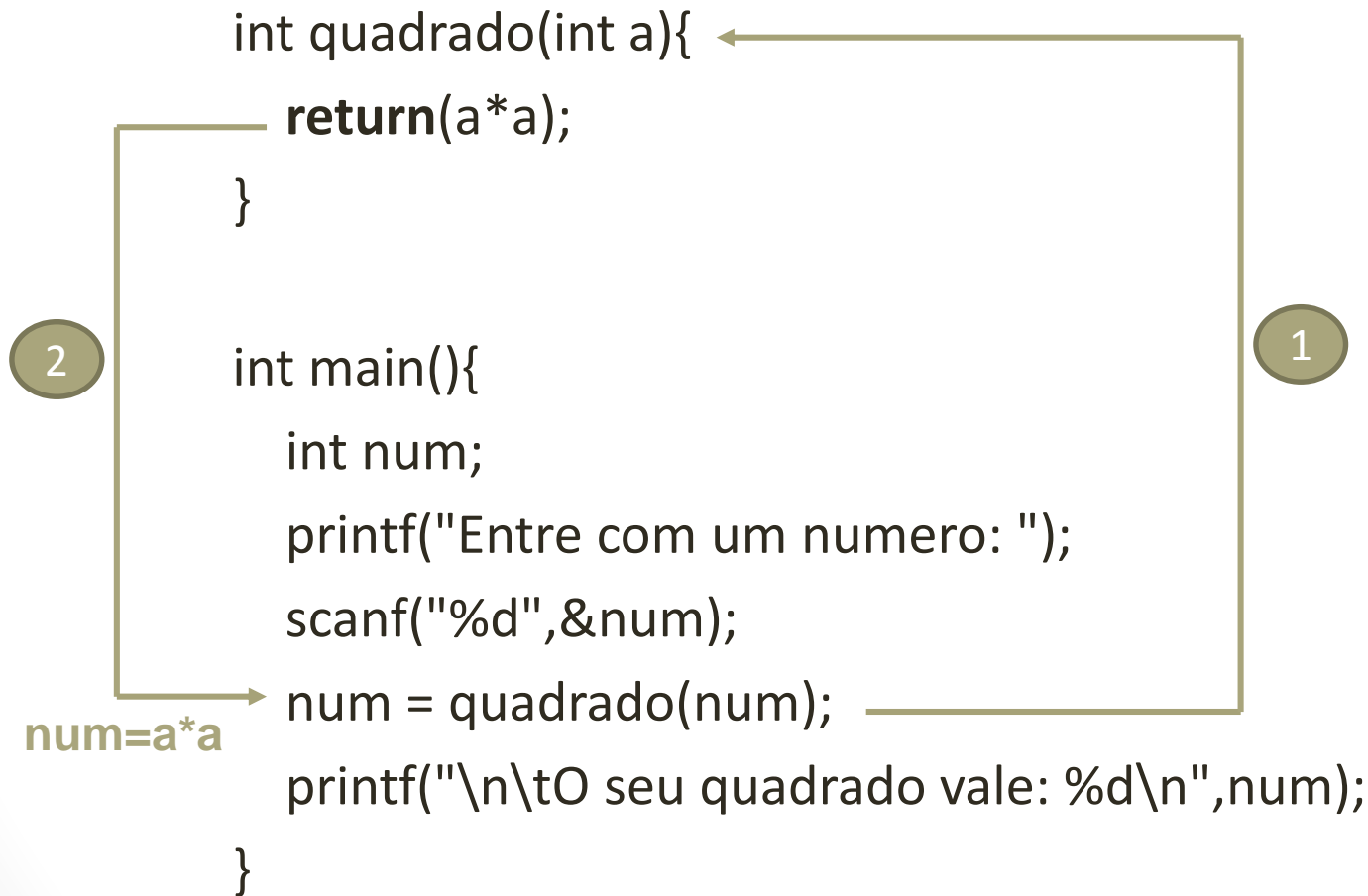
a=num

```
int main(){  
    int num;  
    printf("Entre com um numero: ");  
    scanf("%d",&num);  
    num = quadrado(num);  
    printf("\n\tO seu quadrado vale: %d\n",num);  
}
```

1

Chama a função quadrado()

Como ocorre o retorno da função?



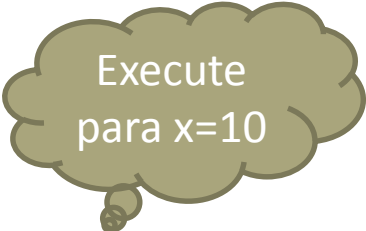
Local de declaração de uma função

Toda função deve ser definida ou declarada antes de ser utilizada, ou seja, antes do programa principal (**main**)

```
#include <stdio.h>

int fatorial (int n){
    int i, fat = 1;
    for (i =1; i<=n ; i ++)
        fat = fat * i ;
    return fat;
}

int main(){
    int x ;
    printf("\nDigite um numero inteiro positivo: " );
    scanf("%d", &x) ;
    int fat = fatorial (x);
    printf("\tO fatorial de %d eh : %.2f\n", x,fat);
}
```



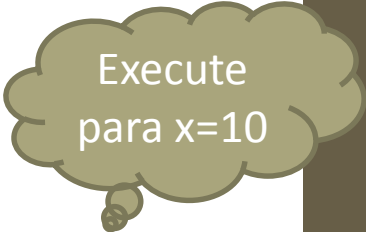
Execute
para x=10

Local de declaração de uma função

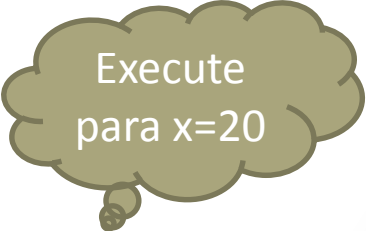
```
#include <stdio.h>

int fatorial (int n){
    int i, fat = 1;
    for (i =1; i<=n ; i ++ )
        fat = fat * i ;
    return fat;
}

int main(){
    int x ;
    printf("\nDigite um numero inteiro positivo: " );
    scanf("%d", &x) ;
    int fat = fatorial (x);
    printf("\tO fatorial de %d eh : %.2f\n", x,fat);
}
```



Execute
para x=10



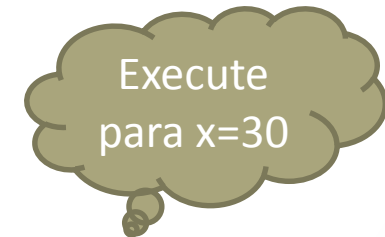
Execute
para x=20

Local de declaração de uma função

```
#include <stdio.h>

float fatorial (int n){
    int i;
    float fat = 1;
    for (i =1; i<=n ; i ++ )
        fat = fat * i ;
    return fat;
}

int main(){
    int x ;
    printf("\nDigite um numero inteiro positivo: " );
    scanf("%d", &x) ;
    float fat = fatorial (x);
    printf("\tO fatorial de %d eh : %.2f\n", x,fat);
}
```

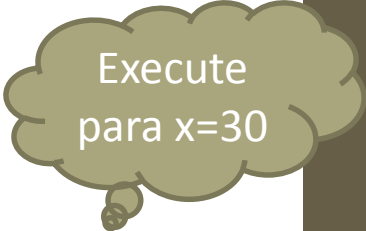


Local de declaração de uma função

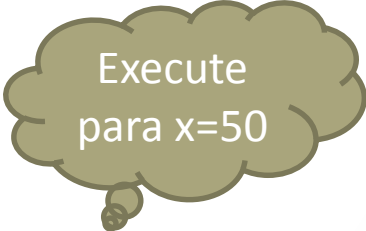
```
#include <stdio.h>

float fatorial (int n){
    int i;
    float fat = 1;
    for (i =1; i<=n ; i ++){
        fat = fat * i ;
    }
    return fat;
}

int main(){
    int x ;
    printf("\nDigite um numero inteiro positivo: " );
    scanf("%d", &x) ;
    float fat = fatorial (x);
    printf("\tO fatorial de %d eh : %.2f\n", x,fat);
}
```



Execute
para x=30



Execute
para x=50

Exemplo de função

```
#include <stdio.h>
double fatorial (int n){
    int i;
    double fat = 1;
    for (i =1; i<=n ; i ++){
        fat = fat * i ;
    }
    return fat;
}
```

```
// OUTRO PROGRAMA PRINCIPAL
int main (void){
    double fat;
    for (int i=1; i<=50; i++){
        fat = fatorial (i);
        printf ("O fatorial de %d eh : %.2f\n", i,fat);
    }
}
```

Exemplo de função

```
#include <stdio.h>
double fatorial (int n){ //FUNÇÃO
    int i;
    double fat = 1;
    for (i =1; i<=n ; i ++ )
        fat = fat * i ;
    return fat;
}
int main (void){ //PROGRAMA PRINCIPAL
    double fat;
    for (int i=1; i<=150; i++){
        fat = fatorial (i);
        printf ("O fatorial de %d eh : %.2f\n", i,fat);
    }
}
```

Local de declaração de uma função

- Pode-se também declarar uma função depois da cláusula **main**. Nesse caso, é preciso declarar antes o protótipo da função:

tipo_retornado nome _função (lista de parâmetros);

- O protótipo de uma função é uma declaração de função que omite o corpo mas especifica o seu nome, tipo de retorno e lista de parâmetros.
 - Não é preciso incluir os nomes das variáveis passadas como parâmetros. Apenas os seus tipos já são suficientes.

Exemplo com protótipo da função

```
double fatorial (int); //Isto é um protótipo da função
```

```
int main (void){ //PROGRAMA PRINCIPAL  
    double fat;  
    for (int i=1; i<=150; i++){  
        fat = fatorial (i);  
        printf ("O fatorial de %d eh : %.2f\n", i,fat);  
    }  
}
```

```
double fatorial (int n){ //FUNÇÃO  
    int i;  
    double fat = 1;  
    for (i =1; i<=n ; i ++)  
        fat = fat * i ;  
    return fat;  
}
```


EXEMPLO

Dados 3 números inteiros, mostrar o maior.

```
int maior(int x, int y, int z){  
    int maior = x;  
    if (y > maior) maior = y;  
    if (z > maior) maior = z;  
    return maior;  
}
```

EXEMPLO

```
int maior(int x, int y, int z){  
    int maior = x;  
    if (y > maior) maior = y;  
    if (z > maior) maior = z;  
    return maior;  
}
```

```
// PROGRAMA PRINCIPAL
```

```
int main (void){  
    int num1, num2, num3, resp=1;  
    while (resp){  
        printf("\nDigite o primeiro numero: "); scanf("%i", &num1);  
        printf("Digite o primeiro numero: "); scanf("%i", &num2);  
        printf("Digite o primeiro numero: "); scanf("%i", &num3);  
        printf("\tMaior: %d \n", maior(num1, num2, num3));  
        printf("Digite 1 para repetir ou 0 para sair: "); scanf ("%i", &resp);  
    }  
}
```

EXERCÍCIO

Dado o programa principal abaixo, montar a função primo.

```
int main (void){
    int num, resp=1;
    printf("Digite um numero maior que 0: "); scanf("%i", &num);
    while (num>0){
        if (primo(num))
            printf("\tEste numero eh primo.");
        else printf("\tEste numero nao eh primo.");
        printf("\n\nDigite numero maior que 0: "); scanf("%i", &num);
    }
}
```

```
int primo(int num){  
    if (num==1) return 0;  
    int i = 1, primo = 1;  
    while ((primo) && (i < (num/2))){  
        i += 1;  
        if (num%i == 0)  
            primo = 0;  
    }  
    return primo;  
}
```

```
int main (void){  
    int num, resp=1;  
    printf("Digite um numero maior que 0: "); scanf("%i", &num);  
    while (num>0){  
        if (primo(num))  
            printf("\tEste numero eh primo.");  
        else printf("\tEste numero nao eh primo.");  
        printf("\n\nDigite numero maior que 0: "); scanf("%i", &num);  
    }  
}
```

Escopo de variáveis

Tanto as variáveis definidas no corpo da função, quanto os argumentos da função são acessíveis apenas no bloco de código da própria função.

```
int v=10, w= 1; // variáveis globais
```

```
int main(void){  
    verifica(100, 150);  
    cancela(20,30);  
}
```

Escopo de variáveis

```
int v=10, w= 1; // variaveis globais
```

```
void verifica(int a, int b){
```

```
    int k=20;
```

```
    // Neste bloco de código, V, W, A, B e K são acessíveis.
```

```
    printf("V, W = %d, %d\n", v, w); //variáveis globais
```

```
    printf("A, B = %d, %d\n", a, b); //variáveis locais
```

```
    printf(" K= %d\n", k); //variável local
```

```
}
```

```
int main(void){
```

```
    verifica(100, 150);
```

```
    //cancela(20,30);
```

```
}
```

Escopo de variáveis

```
int v=10, w= 1; // variaveis globais
```

```
void verifica(int a, int b){  
    int k=20;  
    // Neste bloco de código, V, W, A, B e K são acessíveis.  
    printf("V, W = %d, %d\n", v, w); //variáveis globais  
    printf("A, B = %d, %d\n", a, b); //variáveis locais  
    printf(" K= %d\n", k);          //variável local  
}
```

```
int main(void){  
    verifica(100, 150);  
    //cancela(20,30);  
}
```

V, W = 10, 1
A, B = 100, 150
K = 20

Escopo de variáveis

```
int v=10, w= 1; // variaveis globais
```

```
void cancela(int x, int y){  
    int k=0;  
    // Neste bloco de código, V, W, X, Y e K são acessíveis,  
    // mas note que *este* K não tem nenhuma relação  
    // com o K da função verifica() anterior.  
    printf("V, W = %d, %d\n", v, w); //variáveis globais  
    printf("X, Y = %d, %d\n", x, y); //variáveis locais  
    printf(" K= %d\n", k);    //variável local  
}
```

```
int main(void){  
    //verifica(100, 150);  
    cancela(20,30);  
}
```

V, W = 10 1
X, Y = 20 30
K = 0

EXERCÍCIO 1

Faça uma função que receba por parâmetro um valor inteiro e positivo N e retorne o valor de S, dado por:

$$S = 1 + 1/1! + 1/2! + 1/3! + \dots + 1/N!$$

Exercício 1

```
int main(void){
    int num;
    float soma;
    printf("\nDigite um numero positivo: "); scanf("%d", &num);
    while(num<0){
        printf("Digite um numero positivo: ");
        scanf("%d", &num);
    }
    soma = serie(num); // Definir essa função
    printf("Soma da serie: %6.4f\n", soma);
}
```

Exercício 1

```
float serie(int n){ //calcula a soma de uma serie
    float x,soma = 1.0;
    for (int k=1; k<n; k++){
        x = fatorial(k); // definir essa funcao
        soma = soma + 1/x;
    }
    return soma;
}
```

```
int main(void){
    int num;
    float soma;
    printf("\nDigite um numero positivo: "); scanf("%d", &num);
    while(num<0){
        printf("Digite um numero positivo: ");
        scanf("%d", &num);
    }
    soma = serie(num); // Definir essa função
    printf("Soma da serie: %6.4f\n", soma);
}
```

Exercício 1

```
float serie(int n){  
    //calcula a soma de uma serie  
    float x,soma = 1.0;  
    for (int k=1; k<n; k++){  
        x = fatorial(k);  
        soma = soma + 1/x;  
    }  
    return soma;  
}
```

```
float fatorial (int n){  
    int i;  
    float fat = 1;  
    for (i =1; i<=n ; i ++)  
        fat = fat * i ;  
    return fat;  
}
```

```
int main(void){  
    int num;  
    float soma;  
    printf("\nDigite um numero positivo: "); scanf("%d", &num);  
    while(num<0){  
        printf("Digite um numero positivo: ");  
        scanf("%d", &num);  
    }  
    soma = serie(num);  
    printf("Soma da serie: %6.4f\n", soma);  
}
```

Tipos de passagem de parâmetros

Os parâmetros de uma função são um mecanismo para se passar informação de um trecho de código para dentro da função.

- Tipos de passagem de parâmetros:
 - Por valor
 - Por referência

Passagem por Valor

- Uma cópia do dado é feita e passada para a função.
- Esse tipo de passagem de parâmetro é o padrão para todos os tipos básicos pré-definidos em C (**int, char, float e double**).
- Mesmo que o valor de uma variável mude dentro da função, nada acontece com o valor de fora da função.

Passagem por Valor

```
int v=10; //variavel global
```

```
int w= 1; //variavel global
```

```
void verifica(int v, int w){
```

```
    v= v+1;
```

```
    w = w*2;
```

```
    printf("V, W = %d, %d\n", v, w); //variáveis locais
```

```
}
```

```
int main(void){
```

```
    verifica(v, w);
```

```
    printf("V, W = %d, %d\n", v, w); //variáveis globais
```

```
}
```

V, W = 11, 2 // printf da função verifica

V, W = 10, 1 // printf da função main

Passagem por Referência

- Se o valor da variável mudar dentro da função essa mudança se reflete fora da função.
- Existem casos em que isso é necessário, como o caso da função **scanf**.
 - ✓ Sempre que desejamos ler algo do teclado, passamos para a função **scanf** o nome da variável onde o dado será armazenado. Essa variável tem seu valor modificado dentro da função **scanf** e seu valor pode ser acessado no programa principal.
- Na passagem de parâmetros por referência não se passa para a função os valores das variáveis, mas sim os endereços das variáveis na memória.


Passagem por Referência

- Para passar um parâmetro por referência, usa-se o operador “*” na frente do nome do parâmetro durante a declaração da função.

void verifica(int *v, int *w)

- Toda vez que a variável passada por referência for usada dentro da função, o operador “*” deverá ser usado na frente do nome da variável.

***v = *v + 1;**



Ponteiro para
endereço de memória

- Na chamada da função é necessário utilizar o operador “&” na frente do nome da variável que será passada por referência.

verifica(&v, &w);

Passagem por Referência

```
int v=10; //variavel global
```

```
int w= 1; //variavel global
```

```
void verifica(int *v, int *w){
```

```
    *v= *v+1;
```

```
    *w = *w*2;
```

```
    printf("V, W = %d, %d\n", *v, *w); //variáveis locais
```

```
}
```

```
int main(void){
```

```
    verifica(&v, &w);
```

```
    printf("V, W = %d, %d\n", v, w); //variáveis globais
```

```
}
```

V, W = 11, 2 // printf da função verifica

V, W = 11, 2 // printf da função main

Passagem por Valor

```
void troca(int a, int b){
    int temp;
    temp = a;
    a = b;
    b = temp;
    printf ("Dentro : %d e %d\n", a, b);
}
int main (void){
    int x = 2;
    int y = 3;
    printf("Antes : %d e %d\n", x , y);
    troca(x, y);
    printf("Depois : %d e %d\n ", x , y );
}
```

Antes: 2 e 3
Dentro: 3 e 2
Depois: 2 e 3

Passagem por Referência

```
void troca(int *a, int *b){
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
    printf ("Dentro : %d e %d\n", *a, *b);
}
int main (void){
    int x = 2;
    int y = 3;
    printf("Antes : %d e %d\n", x , y);
    troca(&x, &y);
    printf("Depois : %d e %d\n ", x , y );
}
```

Antes: 2 e 3
Dentro: 3 e 2
Depois: 3 e 2

Exercício 2

O número 3025 possui a seguinte característica especial:

$$30 + 25 = 55$$

$$55^2 = 3025$$

Faça uma **função** que receba um número inteiro de quatro dígitos e retorne o resultado do cálculo acima para avaliar se possui a característica acima.

Exercício 2

```
float carac(int n){  
    // verifica a seguinte caracteristica:  
    // 3025 --> 30 + 25 = 55 --> 55^2 = 3025  
    int p1, p2, soma;  
    p1= n/100;  
    p2=n%100;  
    soma = p1 + p2;  
    return pow(soma,2);  
}
```

Passagem por Valor

Exercício 2

```
float carac(int n){  
    // verifica a seguinte caracteristica:  
    // 3025 --> 30 + 25 = 55 --> 55^2 = 3025  
    int p1, p2, soma;  
    p1= n/100;  
    p2=n%100;  
    soma = p1 + p2;  
    return pow(soma,2);  
}
```

Passagem por Valor

```
int main (void){  
    int num;  
    printf("\n Digite um numero com 4 digitos: ");  
    scanf("%d",&num);  
    while ((num<1000) || (num> 9999)){  
        printf("\n Digite um numero com 4 digitos: ");  
        scanf("%d",&num);  
    }  
    if (num == carac(num))  
        printf("%d possui a caracteristica.\n", num);  
    else printf("%d NAO possui a caracteristica.\n", num);  
}
```

Exercício 3

Considerando o exercício 2, altere o programa principal para procurar e mostrar todos os números inteiros com 4 dígitos que possuem esta característica especial.

Exercício 4

Escreva um procedimento, chamado MM, que recebe dois parâmetros, A e B, e devolve o menor dos dois em A e o maior dos dois em B.

Caso sejam passados valores repetidos, a ordem da resposta entre eles não importa.

Exercício 4

```
void mm(int *a, int *b){  
    if (*a<=*b)  
        return;  
    else{  
        int aux = *a;  
        *a = *b;  
        *b= aux;  
        return;  
    }  
}
```

Exercício 4

```
void mm(int *a, int *b){  
    if (*a<=*b)  
        return;  
    else{  
        int aux = *a;  
        *a = *b;  
        *b= aux;  
        return;  
    }  
}
```

```
int main(void)  
{  
    int a, b;  
    printf("Digite o valor de A: ");  
    scanf ("%i", &a);  
    printf("Digite o valor de B: ");  
    scanf ("%i", &b);  
    mm(&a, &b);  
    printf("A = %d, B = %d\n", a, b);  
}
```