

# **INF101202**

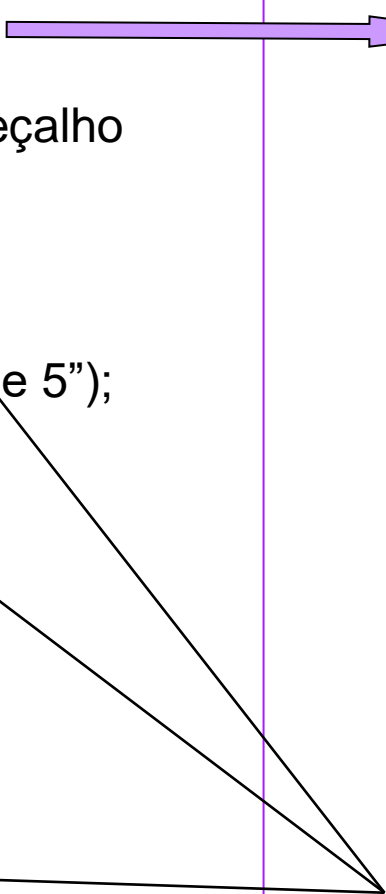
## **Algoritmos e Programação**

### **Modalidade Ead – Turma H**

**Material de apoio: capacitar a programação com  
funções e procedimentos**

O programa a seguir apresenta a saída que está à direita.  
Observe que a parte do código de impressão de 20 asteriscos é repetida 3 vezes no texto do fonte em C (*exemplo de Dantas,L.pag105*).

```
#include<stdio.h>
int main()
{
    int i; // escrita do cabeçalho
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
    printf("Numeros entre 1 e 5");
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
    /* Escrita dos numeros*/
    for (i=1;i<=5;i++)
        printf("%d\n",i);
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
}
```



## Saída:

\*\*\*\*\*

Numeros entre 1 e 5

\*\*\*\*\*

1

2

3

4

5

\*\*\*\*\*

**Código repetido !!!**

Pode-se observar, que o conjunto de instruções:

```
for (i=1;i<20;i++)  
    printf("*");  
    printf("\n");
```

é utilizado **3** vezes.

Melhor se pudéssemos escrevê-lo somente uma vez!!!!

Uma solução: utilizar módulos de código reusáveis !!!!!!!!!!!

E, para a linguagem C, a noção de módulos reusáveis é: **função**

Aplicando o conceito de funções, o programa de impressão de asteriscos se apresenta como:



**Observar:**

**Parte de código com a definição da função**

**Comandos de invocação da função**

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

void linha()
{
    int i;
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
}

int main()
{
    int i;
    linha(); //escreve uma linha de asteriscos
    printf("Numeros entre 1 e 5\n");

    linha(); // escreve uma linha de asteriscos
    for (i=1; i<5; i++)
        printf("%d\n",i);
    linha(); //escreve uma linha de asteriscos
    system("Pause");
    return 0;
}
```

Ação da função:  
escrever  
linha de  
asteriscos

## Observar:

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

```
void linha()
```

```
{
```

```
    int i;
```

```
    for (i=1;i<20;i++)
```

```
        printf("*");
```

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

```
}
```

```
int main()
```

```
{
```

```
    int i;
```

```
    linha(); //escreve uma linha de asteriscos
    printf("Numeros entre 1 e 5\n");
```

```
    linha(); // escreve uma linha de asteriscos
    for (i=1; i<5; i++)
        printf("%d\n",i);
```

```
    linha(); //escreve uma linha de asteriscos
    system("Pause");
    return 0;
}
```

Este programa (texto em C) apresenta 2 funções:

***linha*** – responsável pela ação de escrita de asteriscos na tela;

***main*** – responsável por iniciar o processamento e executar todas as instruções presentes.

**Para se usar uma função é necessário:**

**1.a especificação da função ou sua declaração, isto é, o código de instruções que define sua ação;**

**2. a invocação à função através de seu nome, no caso, **linha()****

## Função. O que é?

- é um módulo de código com um objetivo a realizar;
- deve executar uma tarefa específica, bem identificada;
- permite dividir um programa em módulos de código
- é executada através de uma chamada (invocação) realizada por outro módulo ( **main**)

## **Função. Qual objetivo?**

- facilita a solução de problemas complexos, pois divide um problema original em subproblemas (módulos) que são mais fáceis de resolver (achar uma solução) e mais fáceis de transformar em trechos de código mais simples (subprogramas).

**“A arte de programar consiste na arte de organizar e dominar a complexidade dos sistemas”**

**Dijkstra, 1972**

## Função. Uso

Ao se usar as funções ou módulos de programas está-se utilizando da estratégia da

programação modular



## **Função. Vantagens de Uso**

- ◆ Maior controle sobre a complexidade.
- ◆ Estrutura lógica mais clara.
- ◆ Maior facilidade de depuração e teste.
- ◆ Possibilidade de reutilização de código
- ◆ Possibilidade criar bibliotecas de subprogramas.

## em C existem as funções :

a) Pré-definida ou Padrão

Ex.:

```
int numero;
```

```
float raizquadrada;
```

```
(...)
```

```
raizquadrada = sqrt(numero);
```

Funções  
prontas,  
disponíveis nas  
biblioteca padrão  
do C

b) Definida pelo usuário:

Ex.:

```
potencia();
```

Funções que são  
criadas pelo  
programador

## Ex: Sqrt: função pré-definida

**Sqrt** é uma função de tipo **double** e que recebe um parâmetro também tipo **double**.

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

```
(...)
```

```
raizquadrada = sqrt(numero);
```

```
(...)
```

O tipo de **sqrt** é o tipo do valor que é retornado no seu nome e pode ser usado em expressões.

Por exemplo, no trecho acima **sqrt** é chamada dentro de uma atribuição.

Mas as funções não precisam retornar valores em seus nomes, nem obrigatoriamente necessitam de parâmetros para serem executadas.

# Exemplos

```
...  
a = 0;  
do  
{  
    a = a + 1;  
    printf("%10d", 1 + (rand()%6));  
}  
while (a < 10);  
...
```

```
...  
do  
{  
    scanf("%f", &num);  
    printf("cos(%.1f) = %.1f\n", 0.0,  
        cos( num));  
    printf("Outro ? (1 = sim / 0 = nao) ");  
    scanf("%d", &resposta)  
}  
while (resposta == 1);  
...
```

**Exemplos de uso de funções pre-  
definidas, disponíveis na biblioteca  
<math.h>...**

# Declaração e chamada de funções definidas pelo usuário

```
#include<stdio.h>
#include <stdlib.h>
void linha()
{
    int i;
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
}
main()
{
    int i;
    linha(); // escreve uma linha de
              asteriscos
    printf("Numeros entre 1 e 5\n");
    linha(); // escreve uma linha de
              asteriscos
    for (i=1; i<5; i++)
        printf("%d\n",i);
    linha(); // escreve uma linha de asteriscos
    system("Pause");
    return 0;
}
```

**Declaração:** onde se escreve o código da função

**Chamada:** onde a main invoca a ação da função, através do nome, e que antes foi declarada

# Função em C – sintaxe para declaração sem retorno e sem parâmetros

```
void <identificador> (  
{  
    [ <declaração de variáveis locais>; ]  
    < comandos separados por ';' >  
}
```

opcional

Obrigatório o uso da palavra reservada **void**: indica que a função não retorna um valor através de seu nome.

**Ex:**

```
void oqueseraquefaz()  
// exemplo da sintaxe  
{  
    int x;  
    scanf("%d", &x);  
    c = a + b + x;  
    printf("\n %d",c);  
}
```

declaração de variável local x

# Tipo void

**Void** é um termo que indica ausência.

Em linguagem C, é um tipo de dado.

**Funções tipo `void`  
ou  
com parâmetros `void`**

Funções que não retornam valores em seus nomes são declaradas como tipo `void`.

Funções que não utilizam parâmetros são declaradas com parâmetro `void`.



# Chamada para função sem retorno e sem parâmetros

< nome do procedimento >;

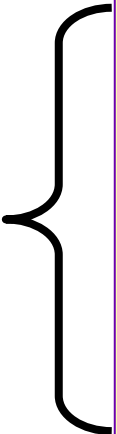
**Ex:**

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


```
void escreve_linha()
{
    int c;
    for (c=1; c<=80; c++)
        printf("*");
}
```

```
int main()
{
    escreve_linha();
    system("pause");
}
```

Declaração da  
função



Chamadas do  
procedimento



```
#include<stdio.h>
#include <stdlib.h>
void linha()
{
    int i;
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
}

int main()
{
    int i;
    linha(); // escreve uma linha de asteriscos
    printf("Numeros entre 1 e 5\n");

    linha(); // escreve uma linha de asteriscos
    for (i=1; i<5; i++)
        printf("%d\n",i);

    linha(); // escreve uma linha de asteriscos

    system("Pause");
    return 0;
}
```

Um programa em C deve posuir **SEMPRE** a função **main()** em seu código, mesmo que existam outras funções declaradas, porque a main é que chama ou ativa as funções para serem usadas.

# Variável local:

As variáveis declaradas dentro da função são variáveis locais.

**x, y, c, p** são variáveis locais da função potencia

Os identificadores declarados dentro de uma função possuem escopo de bloco

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

```
void potencia()
{
```

```
    int x,y;
    int c;
    float p =1;
```

```
    printf(" Digite a base\n");
    scanf("%d",&x);
    printf(" Digite o expoente\n");
    scanf("%d",&y);
    for (c=1; c<=y; c++)
        p = p*x;
    printf("x elevado a y: %1.f\n",p);
```

```
}
int main()
{
    printf(" Cálculo da potencia para
    inteiros \n\n");
    potencia();
    system("pause");
    return 0;
}
```

# Escopo:

é a parte de um programa que um identificador pode ser referenciado.

```
...  
void f1()  
{  
    int x=25;  
    printf("\n x local em f1 e depois de entrar em  
f1\n", x);  
    ++X;  
    printf(" x local em f1 e %d antes de sair de  
f1\n", x);  
}  
...  
.....
```

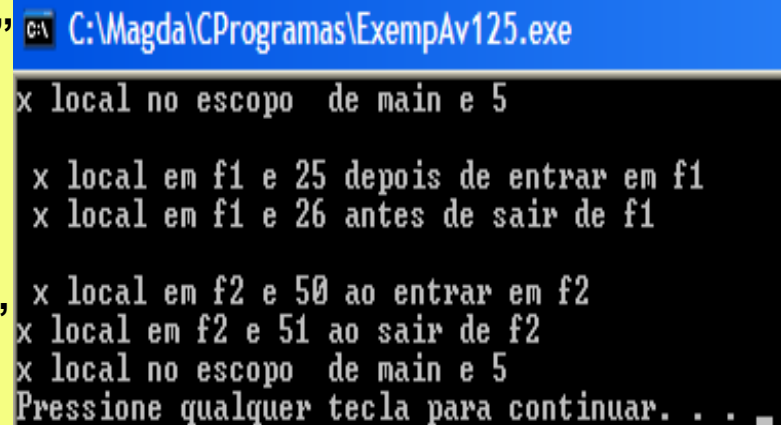
**ESCOPO de x**

x existe a partir de sua declaração int e desaparece após saída pelo } de f1

Escopo de função (cont.): seja o seguinte trecho de programa:

```
...
void f1()
{
    int x=25;
    printf("\n x local em f1 e %d depois de entrar em f1\n",
x);
    ++x;
    printf(" x local em f1 e %d antes de sair de f1\n"
}
void f2()
{
    int x=50;
    printf("\n x local em f2 e %d ao entrar em f2\n",
    ++x;
    printf("x local em f2 e %d ao sair de f2\n", x);
}
int main()
{
    int x=5;
    printf("x local no escopo de main e %d \n", x);
    f1();
    f2();
    printf("x local no escopo de main e %d \n", x);
    .....
```

A saída, após a execução do programa:



```
C:\Magda\Programas\ExempAv125.exe
x local no escopo de main e 5
x local em f1 e 25 depois de entrar em f1
x local em f1 e 26 antes de sair de f1
x local em f2 e 50 ao entrar em f2
x local em f2 e 51 ao sair de f2
x local no escopo de main e 5
Pressione qualquer tecla para continuar. . . _
```

Mostra o escopo de cada x!!

# Exemplo

```
#include <stdlib.h>
#include <stdio.h>
void potencia()
```

```
{
    int x,y;
    int c;
    float p =1;
    printf(" Digite a base\n");
    scanf("%d",&x);
    printf(" Digite o expoente\n");
    scanf("%d",&y);
    for (c=1; c<=y; c++)
        p = p*x;
    printf("x elevado a y: %1.f\n",p);
}
```

```
int main()
{
    int c =0;
    printf(" Cálculo da potencia para inteiros \n\n");
    do
    {
        potencia();
        c = c +1;
    }
    while ( c <=10);
    system("pause");
    return 0;
}
```

Variáveis locais a potencia: x,y,p,c  
somente potencia as “conhece”

Apesar do mesmo nome (**c**), elas são diferentes e não são confundidas. O c de potencia só existe, enquanto potencia estiver em execução

Variável local do main:  
somente **main** a “conhece”

# Variáveis Locais

- São declaradas **dentro de um bloco ou função**.
- Só são reconhecidas e só podem ser referenciadas por **comandos** que estão **dentro do bloco** (ou função) no qual elas foram declaradas.
- **Existem** apenas enquanto o **bloco** de código (ou função) em que foram declaradas **está sendo executado**.

Na função **potencia**, **c** é uma variável local à esta função. Não existe para a função **main** que chama potencia.

## A divisão dos poderes entre a função e o programa principal:

A execução de uma função só acontece se uma outra a chama, ou seja, se a **main** a invoca através de seu nome.

Após a execução de todos os comandos da função chamada, a execução segue na main, na instrução seguinte à chamada.

A seguir um exemplo deste fluxo de execução:





```

int main()
{
    int a[MAX], k;
    //preencher a com valores dos índices
    for (k=0; k<MAX; k++)
        a[k] = k;

    //imprimir a
    for (k=0; k<MAX; k++)
        printf("%d\n",a[k]);

    //somar 2 em cada elemento de a
    for (k=0; k<MAX; k++)
        a[k] = a[k] + 2;

    //imprimir a
    for (k=0; k<MAX; k++)
        printf("%d\n",a[k]);

    //zerar os elementos pares de a
    for (k=0; k<MAX; k++)
        if (a[k] % 2 == 0)
            a[k] = 0;

    //imprimir a
    for (k=0; k<MAX; k++)
        printf("%d\n",a[k]);

    ...
    system("pause");
}

```

1. Identifique primeiramente o que o código a esquerda pretende realizar.
2. Dado o programa no quadro a esquerda, vamos substituir o quadrado lilás que envolve o trecho de código pela expressão:  
**execute imprimevetor**
3. Esta expressão será a **chamada** à função  
imprimevetor.

Assim, teremos o quadro seguinte:

```

int main()
{
    int a[MAX], k;
    //preencher com valores dos índices
    for (k=0; k<MAX; k++)
        a[k] = k;

    //imprimir a
    for (k=0; k<MAX; k++)
        printf("%d\n",a[k]);

    //somar 2 em cada elemento de a
    for (k=0; k<MAX; k++)
        a[k] = a[k] + 2;

    //imprimir a
    for (k=0; k<MAX; k++)
        printf("%d\n",a[k]);

    //zerar os pares
    for (k=0; k<MAX; k++)
        if (a[k] % 2 == 0)
            a[k] = 0;

    //imprimir a
    for (k=0; k<MAX; k++)
        printf("%d\n",a[k]);

    ...
    system("pause");
}

```

```

int main()
{
    int a[MAX], k;
    //preencher com valores dos índices
    for (k=0; k<MAX; k++)
        a[k] = k;

    //imprimir a
    execute imprimevetor

    //somar 2 em cada elemento de a
    for (k=0; k<MAX; k++)
        a[k] = a[k] + 2;

    //imprimir a
    execute imprimevetor

    //zerar os pares de a
    for (k=0; k<MAX; k++)
        if (a[k] % 2 == 0)
            a[k] = 0;

    //imprimir a
    execute imprimevetor

    ...
    system("pause");
}

```

## A divisão dos poderes entre a função e o programa principal:

Dada a declaração da função imprimevetor e após a chamada da função pela **main**, pode-se verificar no quadro do slide a seguir, como o processamento das instruções se desloca entre a execução dos comandos da função, retornando ao programa principal ao término daquelas.

A ação do processamento será representada por setas:



onde linha preta mostra a ação do main até a execução até da primeira chamada de imprimevetor;

A linha **vermelha** mostra a 2ª. chamada e a execução da função de novamente;

e a linha **azul** mostra a terceira chamada.

## A divisão dos poderes entre a função e o programa principal:

### Programa principal

```
int main()
{
    int a[MAX], k;
    //preencher com valores dos índices
    for (k=0; k<MAX; k++)
        a[k] = k;
    //imprimir a
    execute imprimevetor

    //somar 2 em cada elemento
    for (k=0; k<MAX; k++)
        a[k] = a[k] + 2;

    //imprimir a
    execute imprimevetor

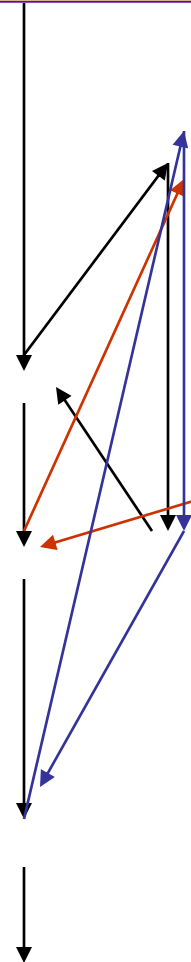
    //zerar os pares
    for (k=0; k<MAX; k++)
        if (a[k] % 2 == 0)
            a[k] = 0;

    //imprimir a
    execute imprimevetor
    ...
    system("pause");
}
```

### função

#### imprimevetor;

```
{
    for (k=0; k<MAX; k++)
        printf("%d\n",a[k]);
}
```



# Execução de uma função

Em tempo de execução, ao ser encontrada uma chamada de uma função, a execução é desviada do ponto onde ocorreu a chamada para a função:

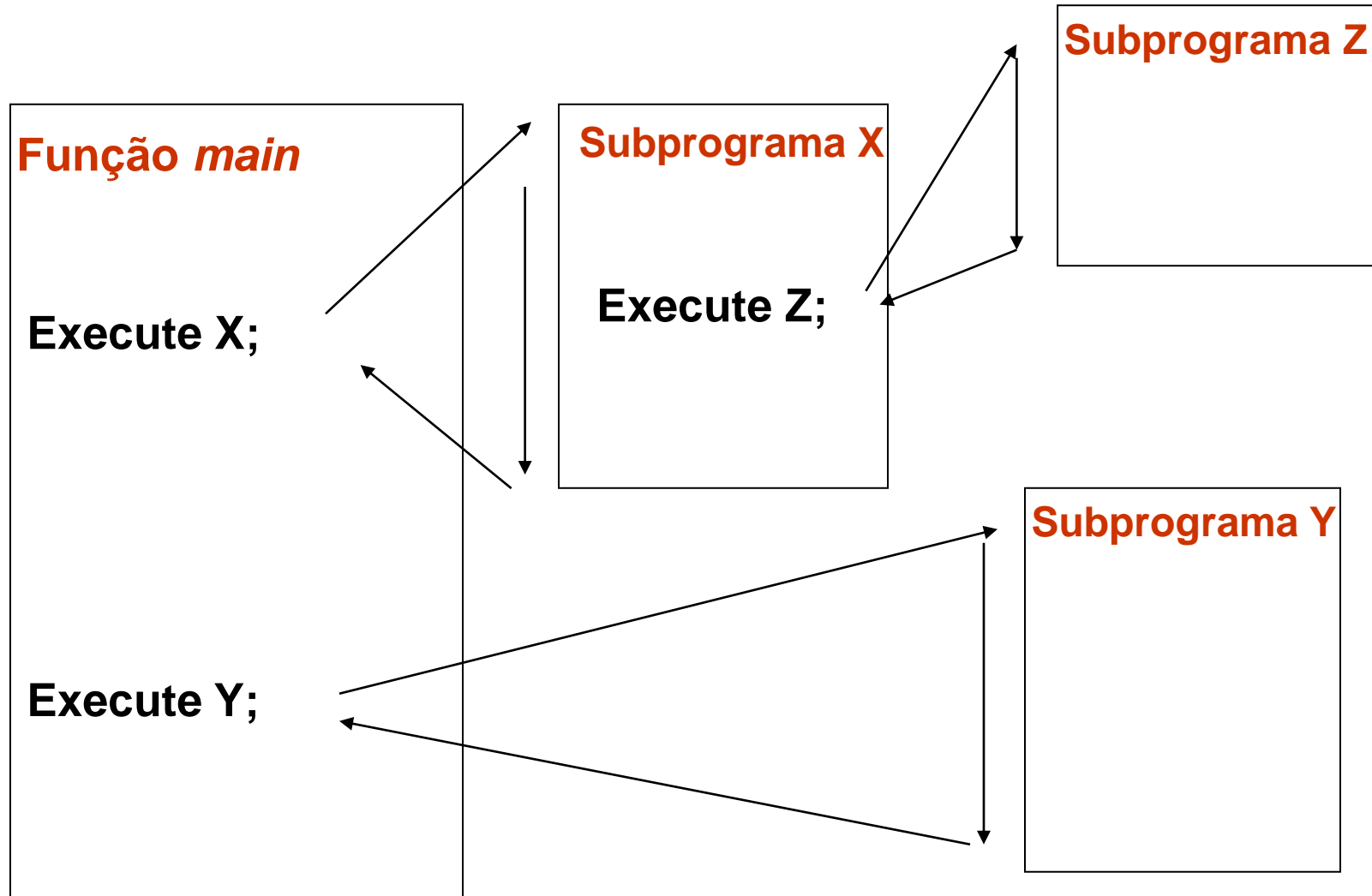
1. A função é ativada.
2. As variáveis nela declaradas (locais) são criadas.
3. A função é executada.

Concluída a execução da função, as variáveis locais à função são destruídas e a execução é retomada, no código que chamou a função, no ponto imediatamente seguinte àquele da chamada.

# Aninhamento de funções é possível?

Em C, é possível chamar uma função de dentro de outra função, **mas não é possível declarar uma função dentro de outra função!**

# Vários níveis de chamadas



# É possível chamar uma função antes da sua declaração?

Não, em C, funções assim como variáveis, etc. só podem ser usados (chamados) após sua declaração. Isso significaria no caso das funções ter que declará-las todas antes da função *main*.

Mas para as funções existe uma pré-declaração chamada *protótipo*, que será vista adiante, que permite contornar essa exigência.



# Variáveis Globais

- São declaradas **fora** das funções (inclusive da *main*).
- São reconhecidas pelo **programa inteiro**.
- Podem ser usadas em qualquer **trecho de código**.
- Existem durante **toda a execução** do programa.

# Variáveis Globais

Em programação estruturada  
devem, sempre que possível, serem  
evitadas! !!

## ATENÇÃO:

**Variáveis locais têm prioridade sobre as variáveis globais: se as variáveis locais tiverem nomes iguais àqueles de variáveis globais, os subprogramas terão acesso apenas às variáveis locais.**

## Programa

## Subprograma X

```
int a, b, k;  
int main ( )
```

```
...  
x ();
```

```
...
```

```
...  
x ();
```

```
...
```

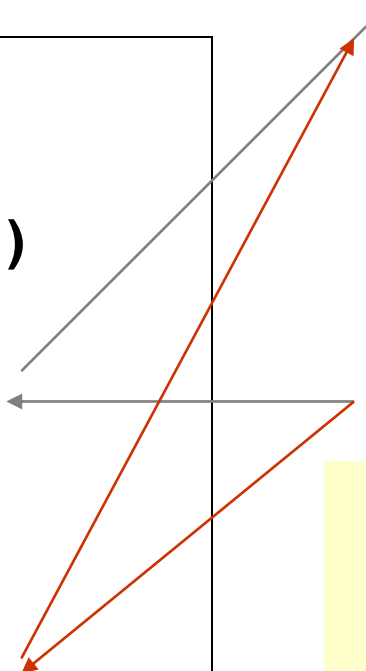
```
int m, k
```

```
...
```

```
a = 0;
```

```
k = 5;
```

```
...
```



Variáveis locais com nomes iguais a variáveis globais: o subprograma tem acesso somente à variável local.

Não enxerga a variável global.

Variável **a**: global.

Variável **k**: há duas, uma global e outra local.

Quando a função **x** está ativa, só o **k** local é percebido pela função.

## Execução

0	4	9
a	b	k

```
C:\backupcida\LinguagemCProgramas\
Na main 1: a = 7 b = 4 k = 9
Em x: a = 0 m = -1 k = 5
Na main 2: a = 0 b = 4 k = 9
Em x: a = 0 m = -1 k = 5
Na main 3: a = 0 b = 4 k = 9
Pressione qualquer tecla para con
```

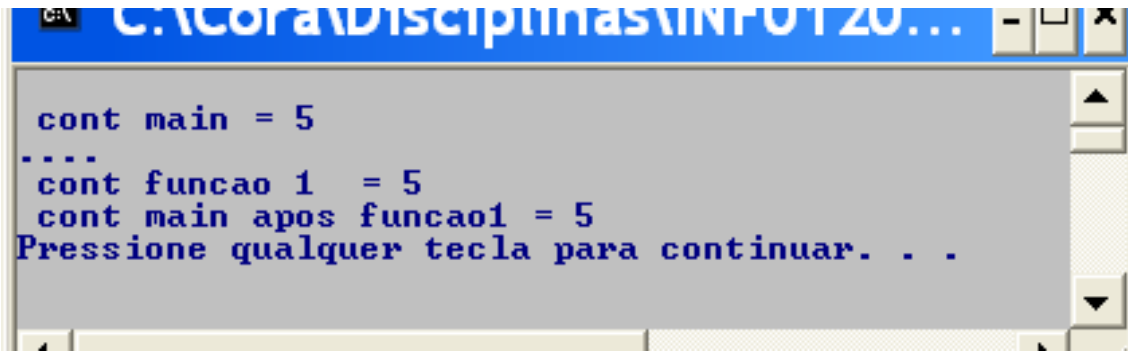
O valor de **m** é aleatório, já que **m** nunca recebe valor.

# Variáveis Locais e Globais – exemplo 1

```
#include <stdio.h>
#include <stdlib.h>
int cont=5; // variável global, maior prioridade!
void funcao1(void)
{
    int i;
    for (i=1; i < cont; i++)
        printf(".");
    printf("\n cont funcao 1  = %d", cont);
}

int main( )
{
    int x;
    system("color 71");
    printf("\n cont main = %d \n", cont);
    funcao1();
    printf("\n cont main apos funcao1 = %d \n", cont);
    system("pause");
    return(0);
}
```

**Função1 e main**  
utilizam a variável  
**cont** global!



```
C:\Corso Disciplinas\INFO120...
cont main = 5
.....
cont funcao 1  = 5
cont main apos funcao1 = 5
Pressione qualquer tecla para continuar. . .
```

## Variáveis Locais e Globais – exemplo 2

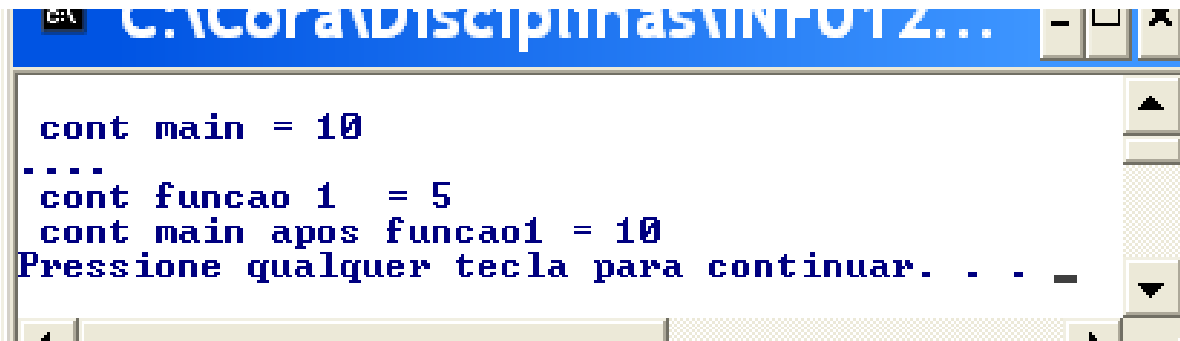
```
#include <stdio.h>
#include <stdlib.h>
int cont=5; // variavel global, maior prioridade!
void funcao1(void)
{
    int i;
    for (i=1; i < cont; i++)
        printf(".");
    printf("\n cont funcao 1 = %d", cont);
}

int main( )
{
    int cont=10, x; // cont local a main
    system("color 71");
    printf("\n cont main = %d \n", cont);
    funcao1( );
    printf("\n cont main apos funcao1 = %d \n", cont);
    system("pause");
    return(0);
}
```

Primeira declaração de **cont** é global.

Em **função1** é o **cont** global que vale, já que não existe **cont** local.

Na função **main**, há um **cont** declarado, internamente à **main**, então é ele quem vale.



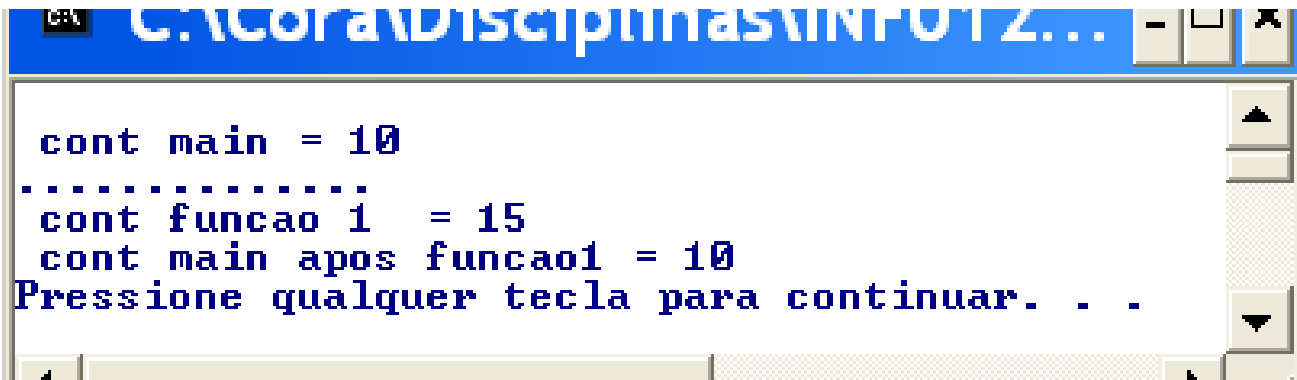
```
C:\Corad\Disciplinas\INF012...
cont main = 10
.....
cont funcao 1 = 5
cont main apos funcao1 = 10
Pressione qualquer tecla para continuar. . . -
```

## Variáveis Locais e Globais – exemplo 3

```
#include <stdio.h>
#include <stdlib.h>
int cont=5; // variavel global
void funcao1(void)
{
    int cont=15, i; // cont local a funcao1
    for (i=1; i < cont; i++)
        printf(".");
    printf("\n cont funcao 1 = %d", cont);
}

int main( )
{
    int cont=10, x; // cont local a main
    system("color 71");
    printf("\n cont main = %d \n", cont);
    funcao1( );
    printf("\n cont main apos funcao1 = %d \n", cont);
    system("pause");
    return(0);
}
```

Como tanto **função1**, quanto **main**, tem uma variável local **cont** declarada, ninguém enxerga o **cont** global!



```
C:\Corso\Disciplinas\INF012...
cont main = 10
.....
cont funcao 1 = 15
cont main apos funcao1 = 10
Pressione qualquer tecla para continuar. . .
```

# Importantíssimo!!!

Em funções, utilizar sobretudo **variáveis locais e parâmetros** (se for o caso).

**EVITAR AO MÁXIMO O USO DE VARIÁVEIS GLOBAIS!!!!**

# Funções

Na verdade:

- A Linguagem C se baseia completamente em funções, sendo elas os “blocos de construção do programa”;
- o corpo principal do programa ( main) nada mais é do que a função por onde o programa começa;
- Geralmente, programas em C são várias mini-funções, ao contrário de um único bloco grande;