

Funções

PROGRAMAÇÃO MODULAR

(com uso de subprogramas)



Subprogramas

- Trechos de código independentes, com estrutura semelhante àquela de programas, mas executados somente quando chamados por outro(s) trecho(s) de código.
- Devem executar UMA tarefa específica, muito bem identificada (conforme a programação estruturada).
- Ao ser ativado um subprograma, o fluxo de execução desloca-se do fluxo principal para o subprograma. Concluída a execução do subprograma, o fluxo de execução retorna ao ponto imediatamente após onde ocorreu a chamada do subprograma.



Vantagens do uso de subprogramas:

- **Maior controle sobre a complexidade.**
- **Estrutura lógica mais clara.**
- **Maior facilidade de depuração e teste, já que subprogramas podem ser testados separadamente.**
- **Possibilidade de reutilização de código.**



Funções

Funções são segmentos de programa que executam uma determinada tarefa específica.

Funções (também chamadas de rotinas, ou subprogramas) são a essência da **programação estruturada**.

Ex: `sqrt()`, `strlen()` etc.

O programador também pode escrever suas próprias funções, chamadas de **funções de usuário**, que tem uma estrutura muito semelhante a um programa.



Forma geral da declaração de uma função

```
tipo_da_funcao nome_da_função (lista_de_parâmetros)
{
    //declarações locais
    //comandos
}
```

•**tipo_da_funcao**: o tipo de valor retornado pela função. Se não especificado, por falta a função é considerada como retornando um inteiro.

•**nome_da_função**: nome da função conforme as regras do C

•**lista_de_parâmetros**: tipo de cada parâmetro seguido de seu identificador, com vírgulas entre cada parâmetro. Mesmo se nenhum parâmetro for utilizado, os parênteses são obrigatórios. Os parâmetros da declaração da função são chamados de parâmetros formais.



Exemplos de cabeçalhos de funções

Ex.:

```
soma_valores (int valor1, int valor2); // por falta é inteira
```

```
void imprime_linhas(int num_lin);
```

```
void apresenta_menu ( );
```

```
float conv_dolar_para_reais(float dolar);
```



Forma Geral de Funções - Exemplo

```
float calculaMedia (float notas[10])  
{  
    float soma = 0;  
    for(int i=0; i<10; i++)  
        soma+=notas[i];  
    return soma/10;  
}
```



Funções void

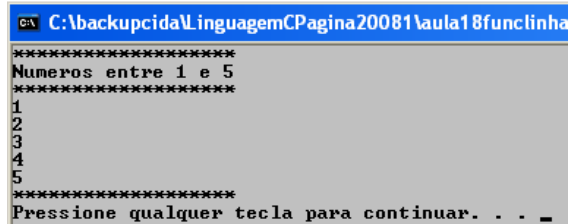
Void é um termo que indica ausência.
Em linguagem C é um tipo de dados.



Programa **escreveint** versão 1: com linha de asteriscos produzida por trecho que se repete no código.

```
//Escrita de numeros inteiros
#include<stdio.h>
#include <stdlib.h>
main( )
{
    int i;
    system("color70")
    //apresentacao do cabecalho
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
    printf("Numeros entre 1 e 5\n");
    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");
    system("pause");
}
```

Execução



```
C:\backupcida\LinguagemCPagina20081\aula18func\linha
*****
Numeros entre 1 e 5
*****
1
2
3
4
5
*****
Pressione qualquer tecla para continuar. . . _
```

A repetição de trechos de código idênticos em um programa pode ser um procedimento fácil e rápido, mas facilmente tende a produzir erros.

Tanto a manutenção quanto a alteração de programas com trechos repetidos tende a ser mais trabalhosa e sujeita a erros.

Com frequência alterações de trechos iguais que se repetem não são realizadas em todas as ocorrências do trecho ou são realizadas de forma incompleta em alguma ocorrência, com resultados bastante danosos.

A solução para esta questão são os **subprogramas**.

A seguir uma versão do programa **escreveint** onde as linhas de asterisco são produzidas pela função **apresente_linha**.

Programa **escreveint** versão 2: com uma função para apresentar linhas de asteriscos.

```
//escrita de numeros inteiros
#include<stdio.h>
#include <stdlib.h>
void apresente_linha(void);
main( )
{
    int i;
    system("color 70");
    //apresentacao do cabecalho
    apresente_linha( );
    printf("Numeros entre 1 e 5\n");
    apresente_linha( );
    // Escrita dos numeros
    for (i=1;i<=5;i++)
        printf("%d\n",i);
    apresente_linha( );
    system("pause");
}

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

Chamadas à função **apresente_linha** substituem trechos repetidos

Execução

```
GA C:\backupcida\linguagemCPagina20081\aula18funci
*****
Numeros entre 1 e 5
*****
1
2
3
4
5
*****
Pressione qualquer tecla para continuar. . .
```



Chamadas à função **apresente_linha** substituem trechos repetidos

```
//escrita de numeros inteiros
#include<stdio.h>
#include <stdlib.h>
void apresente_linha(void);
main( )
{
    int i;
    system("color 70");
    //apresentacao do cabecalho
    apresente_linha( );
    printf("Numeros entre 1 e 5\n");
    apresente_linha( );
    // Escrita dos numeros
    for (i=1;i<=5;i++)
        printf("%d\n",i);
    apresente_linha( );
    system("pause");
}

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

Protótipo da função **apresente_linha**

Chamadas da função **apresente_linha**

Declaração da função

apresente_linha:
escreve uma linha de asteriscos.

Cabeçalho da função `apresente_linha`

`void apresente_linha (void)`



Indica que a função não retorna valor no seu nome.



Indica que a função não tem parâmetros.

A função **`apresente_linha`** realiza sua tarefa sem receber nenhum valor do mundo externo à função, via parâmetros, e sem retornar nenhum valor no seu nome.

Seu tipo é `void` e seus parâmetros são `void`.



Execução de uma função

Em tempo de execução, ao ser encontrada uma chamada de uma função, a execução é desviada para o trecho de código da função.

A função é ativada e os itens locais à função são criados (os parâmetros por valor e os itens declarados internamente à função, como variáveis e constantes).

A função é executada até que seu término seja atingido.

Concluída a execução da função, todos os elementos locais à função que foram criados no momento de sua ativação são destruídos e a execução retorna ao fluxo principal, ao ponto imediatamente seguinte àquele onde ocorreu a chamada da função.



Variáveis locais

Os parâmetros que aparecem no cabeçalho das funções e as variáveis e constantes declaradas internamente a funções são locais à função.

Na função `apresente_linha`, o `i` é uma variável local a essa função.

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



Variáveis e constantes locais

Importante:

Recomenda-se fazer todas as declarações de uma função no seu início.

As variáveis e constantes declaradas em uma função são ditas locais à função porque:

- só podem ser referenciadas por comandos que estão dentro da função em que foram declaradas;
- existem apenas enquanto a função em que foram declaradas está sendo executada. São criadas quando a função é ativada e são destruídas quando a função encerra.



Criação e destruição de variáveis locais a uma função:

Programa principal

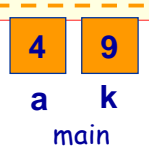
```
int a = 4, k = 9;
```

```
...  
execute x
```

```
...
```

```
...  
execute x
```

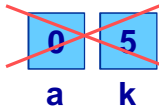
```
...
```



Subprograma X

```
int a, k
```

```
...  
a = 0; // local  
k = 5; // local  
...
```



Variáveis locais:

- uma função (inclusive a *main*) tem acesso somente às variáveis locais.
- não altera valor de variáveis de outras funções.



Criação e destruição de variáveis locais a uma função:

Programa principal

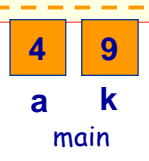
```
int a = 4, k = 9;
```

```
...  
execute x
```

```
...
```

```
...  
execute x
```

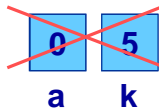
```
...
```



Subprograma X

```
int a, k
```

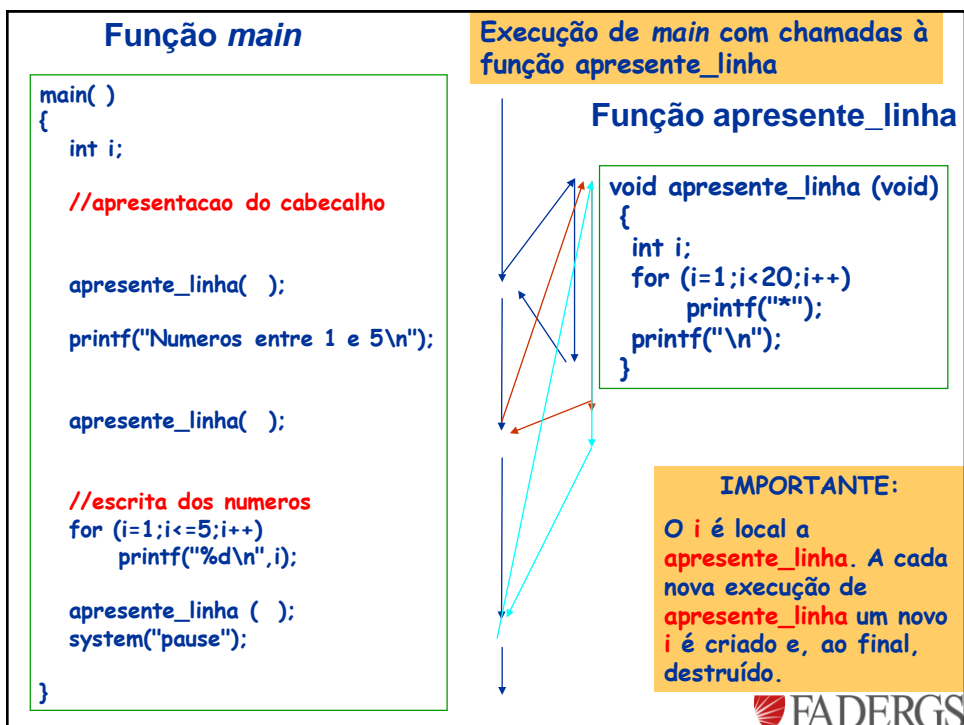
```
...  
a = 0; // local  
k = 5; // local  
...
```



ATENÇÃO:

Uma função (inclusive a *main*) tem acesso somente às suas variáveis locais.





Funções com tipo não void e com parâmetros

sqrt: função pré-definida

- A seguir um programa que extrai a raiz quadrada de um número indeterminado de valores informados.
- Para extrair a raiz quadrada dos valores é usada a função pré-definida **sqrt**, da biblioteca **math.h**.



sqrt: função pré-definida (cont.)

```
//extrai a raiz quadrada de valores informados
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
main ( )
{
    int seguir;
    double valor;
    do
    {
        printf("\nValor para extrair raiz: ");
        scanf("%lf", &valor);
        printf ("\nRaiz quadrada de %6.2lf = %6.2lf\n", valor, sqrt(valor));
        printf("\nMais um valor, digite 1, para parar, digite 0: ");
        scanf("%d", &seguir);
    }
    while (seguir);
    system("pause");
}
```



sqrt: função pré-definida (cont.)

A função `sqrt` é do tipo `double`, isso significa que quando ela é chamada, no lugar de sua chamada retorna um valor `double`.

Para executar essa função é necessário fornecer um parâmetro, o valor para o qual se deseja que a raiz quadrada seja calculada.

No exemplo, está armazenado na variável `valor`.



calc_produto: função definida pelo usuário

- A seguir um programa que calcula o produto de um número indeterminado de pares de valores informados.
- Para calcular os produtos é usada a função definida pelo usuário `calc_produto`.



produto: função definida pelo usuário (cont.)

//calcula produtos de pares de valores informados

```
#include <stdio.h>
#include <stdlib.h>
int calc_produto(int, int);
main ( )
{
    int seguir;
    int oper1, oper2, produto;
    do
    {
        printf("\nOperando 1: ");
        scanf("%d", &oper1);
        printf("\nOperando 2: ");
        scanf("%d", &oper2);
        printf("\nProduto = %d\n", calc_produto(oper1, oper2));
        printf("\nPara continuar, digite 1, para parar, digite 0. ");
        scanf("%d", &seguir);
    }
    while (seguir);
    system("pause");
}
```

```
int calc_produto(int valor1, int valor2)
{
    return valor1 * valor2;
}
```



produto: função definida pelo usuário (cont.)

A função `calc_produto` é do tipo `int`, isso significa que quando ela é chamada, no lugar de sua chamada retorna um valor `int`.

Para executar essa função é necessário fornecer dois parâmetros, os dois valores para cálculo do produto, `oper1` e `oper2`.



Comando return()

retorno de valor e fim lógico da função

O comando return **atribui valor** a função.

Ao ser executado, **encerra a execução** da função.

Ao ser executado o return na função calc_produto, um valor é atribuído à função e ela encerra sua execução.

No ponto onde ocorreu a chamada de calc_produto, um valor passa a estar disponível para processamento.



Comando return()

retorno de valor e fim lógico da função

Se uma função é declarada com tipo diferente de void (int, char, float etc.) significa que ela pretende explorar a possibilidade de retorno de um valor em seu nome, e então pode ser usada em expressões.

Uma função que retorna um valor em seu nome deve conter pelo menos uma ocorrência do comando return, uma vez que é pela execução de um comando return que um valor é atribuído ao nome de uma função.



Quando uma função encerra sua execução?

Uma função encerra sua execução quando:

- o fim do seu código é atingido;
- ou
- um comando *return* é encontrado e executado.



Vários comandos return podem existir em uma função?

Sim, embora não seja recomendável.

Segundo os princípios da programação estruturada seguidos na disciplina, cada função deve ter um único ponto de entrada e um único ponto de saída.

Se vários returns existirem em uma função, tem-se múltiplos pontos de saída possíveis.

Mas a função só conclui quando o primeiro *return* é ativado.



Funções em C

As funções devem ser declaradas de modo a serem o mais independentes possível do mundo externo a elas. Nos códigos das funções devem ser usados, sempre que possível, tão somente os parâmetros declarados no cabeçalho da função (se existirem) e os demais itens locais à função.

Em grande medida em C a preocupação com a independência das funções é facilitada pelo fato **dos parâmetros de chamada e dos parâmetros da declaração da função ocuparem espaços de memória distintos e só existir a chamada passagem de parâmetro por valor entre eles.**

Passagem de parâmetro por valor: os parâmetros de chamada e os parâmetros formais (da declaração da função) só se conectam no momento da chamada da função e então o que há é apenas a transferência de valores entre os parâmetros respectivos.



Exemplo de passagem de parâmetro por valor

A seguir duas versões de um programa que recebe um valor e através de uma função soma dez a este valor. Como a passagem de parâmetros para as funções em C é por valor, e o valor alterado não é mostrado no interior da função, a alteração do valor é perdida.

Observar nas duas versões do programa a seguir que como os mundos da função e o mundo externo a ela são mundos distintos, não faz diferença usar nomes iguais (valor e valor) ou diferentes (valor e num) nos parâmetros correspondentes usados na *main* e na função.



Exemplo de passagem de parâmetro por valor (cont.)

```
#include <stdio.h>
#include <stdlib.h>
void soma_dez_a_valor(int);
main ( )
{
    int valor;
    system("color 71");
    printf("\nValor inteiro: ");
    scanf("%d", &valor);
    printf("\nNa Main: valor antes da chamada da funcao: %d\n", valor);
    soma_dez_a_valor(valor);
    printf("\nNa Main: valor apos chamada da funcao: %d\n", valor);
    system("pause");
}

void soma_dez_a_valor(int valor)
{
    valor = valor + 10;
    printf("\nNa Funcao: valor dentro da funcao: %d\n", valor);
}
```

```
Valor inteiro: 12
Na Main: valor antes da chamada da funcao: 12
Na Funcao: valor dentro da funcao: 22
Na Main: valor apos chamada da funcao: 12
Pressione qualquer tecla para continuar. . .
```



Exemplo de passagem de parâmetro por valor (cont.)

```
#include <stdio.h>
#include <stdlib.h>
void soma_dez_a_valor(int);
main ( )
{
    int valor;
    system("color 71");
    printf("\nValor inteiro: ");
    scanf("%d", &valor);
    printf("\nNa Main: valor antes da chamada da funcao: %d\n", valor);
    soma_dez_a_valor(valor);
    printf("\nNa Main: valor apos chamada da funcao: %d\n", valor);
    system("pause");
}

void soma_dez_a_valor(int num)
{
    num = num + 10;
    printf("\nNa Funcao: valor dentro da funcao: %d\n", num);
}
```

```
Valor inteiro: 34
Na Main: valor antes da chamada da funcao: 34
Na Funcao: valor dentro da funcao: 44
Na Main: valor apos chamada da funcao: 34
Pressione qualquer tecla para continuar. . .
```



Exemplo de passagem de parâmetro por valor (cont.)

```
#include <stdio.h>
#include <stdlib.h>
void soma_dez_a_valor(int);
int numero1=2;
main ( )
{
    int valor;
    system("color 71");
    printf("\nValor inteiro: ");
    scanf("%d", &valor);
    printf("\nNa Main: valor antes da chamada da funcao: %d\n", valor);
    printf("\nNa Main: o numero 1 antes da chamada da funcao: %d\n", numero1);
    soma_dez_a_valor(valor);
    printf("\nNa Main: valor apos chamada da funcao: %d\n", valor);
    printf("\nNa Main: o numero 1 apos chamada da funcao: %d\n", numero1);
    system("pause");
}

void soma_dez_a_valor(int valor)
{
    valor = valor + 10;
    printf("\nNa Funcao: valor dentro da funcao: %d\n", valor);
    numero1 = numero1+20;
    printf("\nNa Funcao: o numero dentro da funcao: %d\n", numero1);
}
```

```
Valor inteiro: 12
Na Main: valor antes da chamada da funcao: 12
Na Main: o numero 1 antes da chamada da funcao: 2
Na Funcao: valor dentro da funcao: 22
Na Funcao: o numero dentro da funcao: 22
Na Main: valor apos chamada da funcao: 12
Na Main: o numero 1 apos chamada da funcao: 22
Pressione qualquer tecla para continuar. . . _
```



produto: função definida pelo usuário

```
//calcula produtos de pares de valores informados
#include <stdio.h>
#include <stdlib.h>
int calc_produto(int, int);
main ( )
{
    int seguir;
    int oper1, oper2, produto;
    do
    {
        printf("\nOperando 1: ");
        scanf("%d", &oper1);
        printf("\nOperando 2: ");
        scanf("%d", &oper2);
        printf("\nProduto = %d\n", calc_produto(oper1, oper2));
        printf("\nPara continuar, digite 1, para parar, digite 0: ");
        scanf("%d", &seguir);
    }
    while (seguir);
    system("pause");
}

int calc_produto(int valor1, int valor2)
{
    return valor1 * valor2;
}
```



Relembrando: passagem de parâmetros

Ao ser ativada a função `calc_produto`, `valor1` e `valor2` são criadas.

E os valores existentes nesse momento em `oper1` e `oper2` são transferidos para `valor1` e `valor2`.

A conexão entre `oper1` e `valor1` e `oper2` e `valor2` só existe no momento que a função é ativada.

Fora o momento da ativação as funções `calc_produto` e `main` são mundos independentes.

```
int calc_produto(int valor1, int valor2)
```

Chamada na função main:

```
calc_produto(oper1, oper2);
```

Quaisquer modificações de `valor1` e `valor2` que aconteçam a partir da chamada de `calc_produto` só são conhecidas e percebidas dentro da função `calc_produto`.

Atenção:

`valor1` e `valor2` existem na função `calc_produto`.

`oper1` e `oper2` existem na função `main`.



O quê é necessário para usar-se uma função em Linguagem C?

- A declaração da função.
- A chamada da função.
- Dependendo do caso, o protótipo da função.



Declaração da função: cabeçalho e corpo da função, com o código que produz o(s) resultado(s) esperado(s).

Se for função com tipo diferente de void, deve ter pelo menos um return, para atribuir valor à função.

Chamada da função: no ponto onde se deseja que a função seja executada, deve ser escrito o nome da função seguido de um par de parênteses, tendo no interior o nome dos parâmetros, se houver.

Protótipo: as funções têm que ser declaradas antes de serem usadas. Para deixar a função *main* em destaque, é melhor declarar as funções definidas pelo usuário após a *main*. Então, para funções, o sistema aceita que primeiro só se indique o tipo, nome da função e tipos dos parâmetros, se houver, ou seja, o **protótipo da função**, e depois em algum ponto do código adiante, se declare a função de forma completa.



```
//calcula produtos de pares de valores informados
```

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

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

```
int calc_produto(int, int);
```

```
main ( )
```

```
{
```

```
    int seguir;
```

```
    int oper1, oper2, produto;
```

```
    do
```

```
    {
```

```
        printf("\nOperando 1: ");
```

```
        scanf("%d", &oper1);
```

```
        printf("\nOperando 2: ");
```

```
        scanf("%d", &oper2);
```

```
        printf("\nProduto = %d\n", calc_produto(oper1, oper2));
```

```
        printf("\nMais um valor, digite 1, para parar, digite 0: ");
```

```
        scanf("%d", &seguir);
```

```
    }
```

```
    while (seguir);
```

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

```
}
```

```
int calc_produto(int valor1, int valor2)
```

```
{
```

```
    return valor1 * valor2;
```

```
}
```

Protótipo

Chamada da função

Declaração da função



Cuidados no uso de funções com parâmetros

Em funções com parâmetros, cuidar que o número e o tipo dos parâmetros sejam coincidentes no protótipo (se usado), na declaração e na chamada.

Em C, os parâmetros independentemente de seus nomes são emparelhados na declaração e chamada por ordem de declaração, da esquerda para a direita.

Ex.:

```
int calc_produto(int, int);  
int calc_produto(int valor1, int valor2)  
calc_produto(oper1, oper2)
```



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!**



Exercício

Calcular a média aritmética de dois números informados pelo usuário.

Escreva o código de uma função que calcule a média aritmética de dois valores informados como parâmetros.

Escreva um programa que use esta função.



```
#include<stdio.h>
#include <stdlib.h>
float media(float, float);
main(){

    float v1,v2,m;
    printf ("Informe os numeros: ");
    scanf("%f %f",&v1,&v2);
    m=media(v1,v2);
    printf("a media dos numeros e': %.4f\n",m);
    system("pause");
}
// declaracao da funcao media
float media(float n1, float n2){
    return((n1+n2)/2);
}
```



Linguagem C

Exercício:

Faça um programa que receba a idade de 5 pessoas e mostre mensagem informando "maior de idade" e "menor de idade" para cada pessoa. Considere a idade a partir de 18 anos, inclusive, como maior de idade.

Fazer uma função para o teste da idade.



```
main()
{
    int x,idade;
    for(x=1;x<=5;x++){
        printf("\nDigite a idade ==> ");
        scanf("%d", &idade);
        if(idade >= 18){
            printf("\nmaior de idade\n"); }
        else{
            printf("\nmenor de idade\n");
        }
    }
    getch();
}
```



Exercício: Calcular a média da nota de 5 alunos e verificar quantos conseguiram nota acima da média.

Fazer uma função, com passagem de valor, com o o cálculo da média e outra para a contagem de quantos estão acima da média.

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int contA=0;
    double media, n1, n2, n3, n4, n5;
    printf('Digite as notas');
    scanf ("%lf %lf %lf %lf %lf", &n1, &n2, &n3, &n4, &n5);
    media = ((n1+ n2 + n3 + n4 + n5)/5);
    if (n1>media) contA=contA+1;
    if (n2>media) contA=contA+1;
    if (n3>media) contA=contA+1;
    if (n4>media) contA=contA+1;
    if (n5>media) contA=contA+1;
    printf ("%lf %d", media, contA);
    system("pause");
}
```



O conteúdo dos slides, desta aula, foram retirados de diversas obras e autores.

