

# Linguagem de Programação I

**- Funções -**

**Prof. Ulysses Santos Sousa**  
**[ulyssessousa@ifma.edu.br](mailto:ulyssessousa@ifma.edu.br)**

**Aula 06**

# Roteiro

- **Função**
  - Introdução
  - Definição
  - Chamada
  - Tipo
  - Protótipo
  - Parâmetros
  - Escopo de variáveis
  - Passagem de Parâmetros
  - Funções recursivas

# Introdução

- Até agora, todos os programas foram desenvolvidos utilizando apenas uma função: a função *main()*.
- Em cada programa, o código era único, com início, desenvolvimento e fim.
- Porém, conforme avançamos, deparamo-nos com problemas mais complexos e longos.
- Solução:
  - Dividir tarefas complexas e longas em atividades mais simples e curtas

# Introdução

- **Módulos**

- Utilizados para dividir a resolução de um problema grande em partes menores.
- A técnica de dividir programas em módulos é chamada **dividir para conquistar**.

- **Funções**

- Na Linguagem C os módulos são chamados de funções.
- Evitam que o mesmo código seja escrito várias vezes.
- Como as funções realizam tarefas específicas, isso facilita a correção, o entendimento e a manutenção do código.

# Introdução

- **Funções**

- Os programas em C são escritos geralmente combinando funções pré-definidas, disponíveis na *biblioteca padrão de C*.
  - Exemplos: printf, scanf, pow.
- Permitem que os programas sejam desenvolvidos de maneira mais rápida.
- O programador também pode definir suas próprias funções para serem utilizadas várias vezes no programa.

# Definição de uma função

- Sintaxe geral:

```
tipo nome(declaração dos parâmetros)
{
    instruções;
}
```

# Chamada de uma função

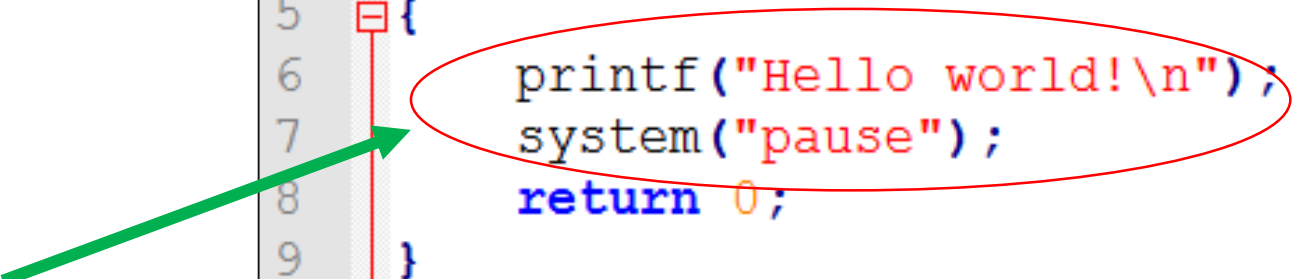
- **Chamada de função**

- Especifica o nome da função e oferece informações de que a função chamada precisa para realizar sua tarefa designada.
- É o meio pelo qual solicitamos que o programa desvie o controle e passe a executar as instruções da função; e que, ao término desta, volte para a posição seguinte à da chamada.

# Chamada de uma função

- Exemplo:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      printf("Hello world!\n");
7      system("pause");
8      return 0;
9  }
```



Chamadas de funções



# Exemplo

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int quadrado(int y); /*protótipo da função*/
5
6  int main()
7  {
8      int x;
9      for (x = 0; x < 10; x++){
10         printf("%d\n", quadrado(x)); /*chamada da função*/
11     }
12     return 0;
13 }
14
15 int quadrado(int y) /*definição da função*/
16 {
17     return y * y;
18 }
```

# Protótipo de uma função

- Geralmente é colocado no início do programa.
- Fornece ao compilador informações necessárias sobre o tipo da função, o número e o tipo dos argumentos.
- Deve preceder a sua definição e a sua chamada.

# Protótipo de uma função

- A declaração de uma função pode ser feita de duas formas:
  - Protótipo externo
    - A declaração da função é feita antes de qualquer função.
    - É realizada uma única vez e é visível para todas as funções do programa.
  - Protótipo interno
    - A declaração é realizada no corpo de todas as funções que a chamam, antes da sua chamada.

# Protótipo de uma função

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int x;
7      int quadrado(int y); /*protótipo interno*/
8      for (x = 0; x < 10; x++){
9          printf("%d\n", quadrado(x)); /*chamada da função*/
10     }
11     return 0;
12 }
13
14 int quadrado(int y) /*definição da função*/
15 {
16     return y * y;
17 }
```

# Tipo de uma função

- O tipo de uma função é definido pelo tipo de valor que ela retorna por meio do comando *return*.
- Por exemplo:
  - Uma função do tipo *float* retorna um valor do tipo *float*.

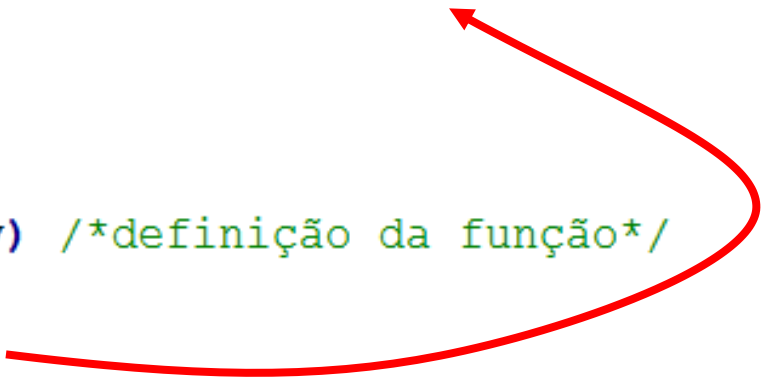
O tipo de uma função é determinado pelo valor que ela retorna via comando ***return***, e não pelo tipo de argumentos que ela recebe.

# Comando *return*

- Tem a função de terminar a função e retornar o controle para a instrução seguinte ao código de chamada.
- Após a palavra *return*, se houver alguma expressão, o valor desta é retornado à função que fez a chamada.
- O uso do comando *return* em funções do tipo *void* é opcional. Se for utilizado, a palavra *return* não deve ser seguida de expressão.

# Comando *return*

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int quadrado(int y); /*protótipo da função*/
5
6  int main()
7  {
8      int x;
9      for (x = 0; x < 10; x++){
10         printf("%d\n", quadrado(x)); /*chamada da função*/
11     }
12     return 0;
13 }
14
15 int quadrado(int y) /*definição da função*/
16 {
17     return y * y;
18 }
```



O valor da função *quadrado* é retornado para a expressão da chamada da função.

# Parâmetros de uma função

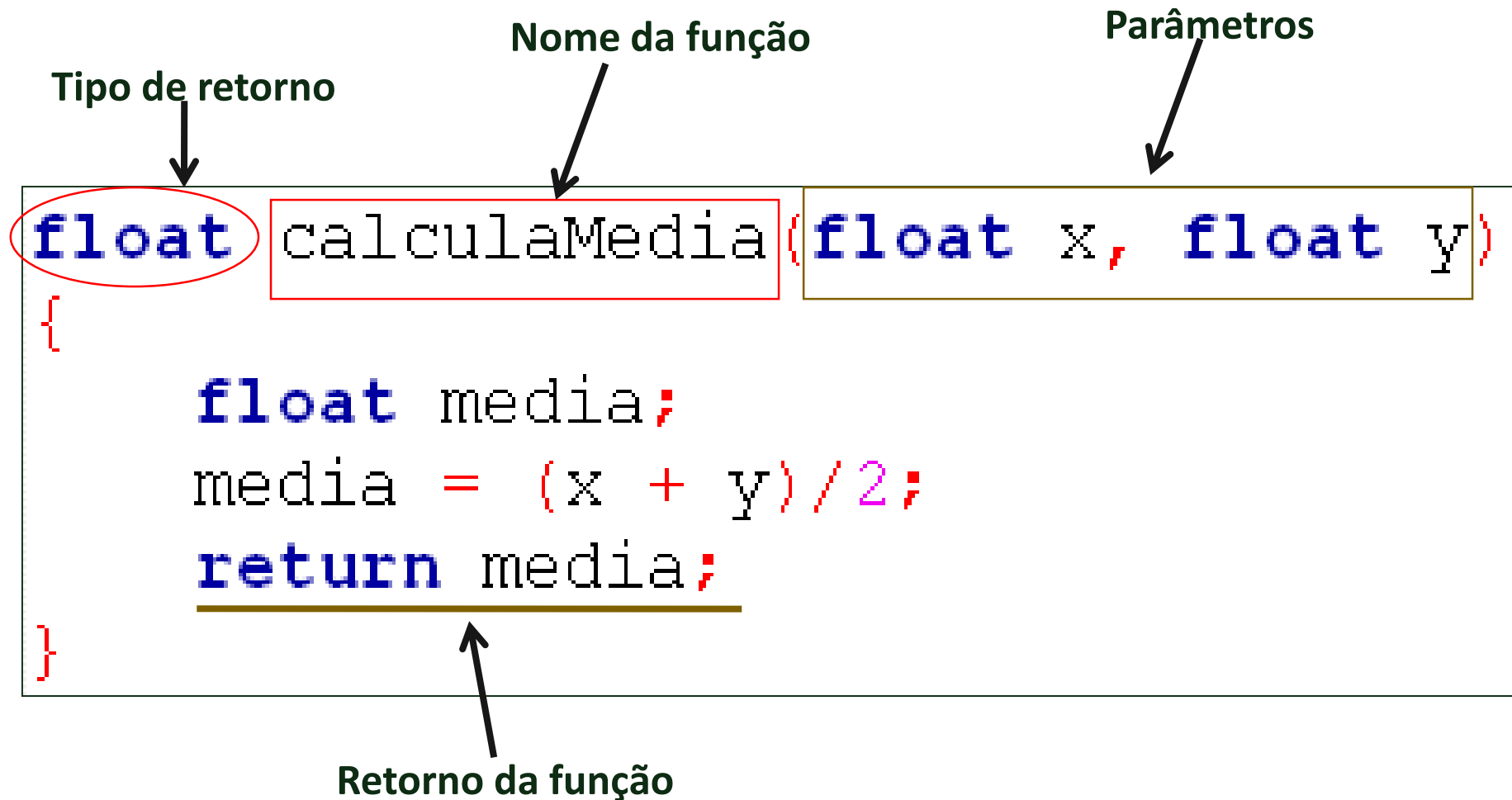
- **Parâmetros**

- Servem para permitir que a uma função seja “executada” para conjuntos de dados diferentes, os quais lhe serão fornecidos no momento de sua ativação.
- O conjunto de parâmetros pode ser vazio, neste caso, a ativação da função executa sempre a mesma ação.
- Os dados que substituirão os parâmetros quando a função for executada são chamados de *argumentos*.



# Parâmetros de uma função

- Exemplo:



# Parâmetros de uma função

- Exemplo (cont.)

Função principal  
do programa

```
int main()  
{  
    float numero1, numero2, resultado;  
    scanf("%f", &numero1);  
    scanf("%f", &numero2);  
    resultado = calculaMedia(numero1, numero2);  
    printf("%f", resultado);  
    return 0;  
}
```

Chamada da função *calculaMedia*

# Parâmetros de uma função

- **Observações**

- O fato do valor retornado ser do tipo *float* não implica que os parâmetros, necessariamente serão do tipo *float*.
- Na função *calculaMedia*, *x* e *y* são parâmetros que recebem os valores dos argumentos *numero1* e *numero2* da função *main()*.
- O número de parâmetros deve ser igual ao número de argumentos.
- Existem funções que não possuem parâmetros, logo não é necessário passar argumentos no momento da sua chamada.

# Escopo de Variáveis

- **Variáveis globais**

- São declaradas fora de qualquer função e podem ser manipuladas em qualquer função.
- Diz-se que seu escopo é *global*.
- Vantagem:
  - Fácil utilização, pois podem ser utilizadas em qualquer função.
- Desvantagem:
  - É mais difícil de realizar manutenções no programa.

# Escopo de Variáveis

- **Variáveis locais**

- São declaradas dentro da função a qual pertencem;
- Só podem ser acessadas dentro da função em que foram criadas;
- Diz-se que seu escopo é **local**;
- O escopo da variável local prevalece sobre o escopo da variável global.

# Escopo de Variáveis

- **Variáveis Locais**

- Vantagem:

- *Independência*: Uma alteração em uma variável local só afetará a própria função, tornando muito mais fácil ao programador garantir que manutenções no programa não tenham efeitos colaterais indesejados em outras funções.

# Escopo de Variáveis

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

int _somatorio;

int somar(int x, int y)
{
    _somatorio = x + y;
}

int main()
{
    int a, b;
    scanf("%d", &a);
    scanf("%d", &b);
    somar(a, b);
    printf("Resultado = %d", _somatorio);
    return 0;
}
```

**Variável global**

**Variáveis locais**

# Passagem de Parâmetros

- **Por valor:**

- Na passagem de parâmetros por valor, uma cópia da variável é feita durante a ativação da função.
- Os valores dos argumentos são atribuídos aos parâmetros de entrada na função ativada.
- Possíveis alterações que podem ser feitas nos valores das variáveis passadas para a função ativada, não irão refletir na função ativadora.



# Passagem de Parâmetros

- **Por valor:**

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void trocaValores(int x, int y)
5 {
6     int aux;
7     aux = x;
8     x = y;
9     y = aux;
10    printf("Valor das variaveis dentro da funcao:\n");
11    printf("x = %d e y = %d\n", x, y);
12 }
13
14 int main()
15 {
16     int x, y;
17     x = 10;
18     y = 5;
19     printf("Antes da chamada da funcao:\n");
20     printf("x = %d e y = %d\n", x, y);
21     trocaValores(x, y);
22     printf("Depois da chamada da funcao:\n");
23     printf("x = %d e y = %d\n", x, y);
24     return 0;
25 }
```

# Passagem de Parâmetros

- **Por referência**

- Neste caso, o endereço da variável, e não uma cópia do valor, é passado durante a ativação do módulo.
- Apesar de ter outro identificador, os valores manipulados na função ativada (parâmetros) estão no mesmo espaço de memória dos argumentos na função anterior e são, efetivamente, a mesma variável.
- Todas as alterações feitas sobre esta função ativada serão refletidas na função ativadora.

# Passagem de Parâmetros

- **Por referência**

- Na linguagem C, utiliza-se o operador & para passar o endereço da variável na chamada da função.
- Na função, os parâmetros são declarados como *ponteiro*, utilizando o operador \*.

# Passagem de Parâmetros

- **Por referência:**

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void trocaValores(int *x, int *y)
5 {
6     int aux;
7     aux = *x;
8     *x = *y;
9     *y = aux;
10 }
11
12 int main()
13 {
14     int x, y;
15     x = 10;
16     y = 5;
17     printf("Antes da chamada da funcao:\n");
18     printf("x = %d e y = %d\n", x, y);
19     trocaValores(&x, &y);
20     printf("Depois da chamada da funcao:\n");
21     printf("x = %d e y = %d\n", x, y);
22     return 0;
23 }
```

# Vetores como argumentos de funções

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define T 3
4  float media(float[], int); /*protótipo*/
5
6  int main()
7  {
8      int i;
9      float notas[T];
10     for(i = 0; i < T; i++){
11         printf("Digite a nota %d: ", i + 1);
12         scanf("%f", &notas[i]);
13     }
14     printf("A media do aluno eh: %.2f\n", media(notas, T));
15     return 0;
16 }
17
18 float media(float notas[], int tamanho)
19 {
20     int i;
21     float m = 0.0;
22     for(i = 0; i < tamanho; i++)
23         m += notas[i];
24     return m/tamanho;
25 }
```

# Vetores como argumentos de funções

- Não é necessário informar o tamanho do vetor no protótipo e no parâmetro da função. Apenas os colchetes.
- Na chamada não é necessário informar o tamanho do vetor e os colchetes.
  - O nome de um vetor, desacompanhado dos colchetes, representa o endereço de memória em que o vetor foi armazenado.
  - Ou seja, `notas` é o endereço de `notas[0]`, que é uma variável do tipo *float*.
- Vetores são passados para funções por referência.

# Matrizes de duas dimensões como argumento de funções

- O método de passagem do endereço é semelhante ao da passagem de um vetor, não importando quantas dimensões tenha a matriz:

```
media(notas, numeroAlunos);
```

- Entretanto, na declaração do protótipo da função deve-se informar o comprimento da matriz na segunda dimensão:

```
media(notas[][3], numeroAlunos);
```

# Funções Recursivas

- Uma função é recursiva quando ela faz uma chamada ou ativa a si mesma.
- Na Matemática, algumas funções clássicas podem ser estabelecidas de tal forma que suas definições utilizem, de modo decorrente, a própria função que se está definindo.
- Exemplo:
  - Cálculo da fatorial.



# Funções Recursivas

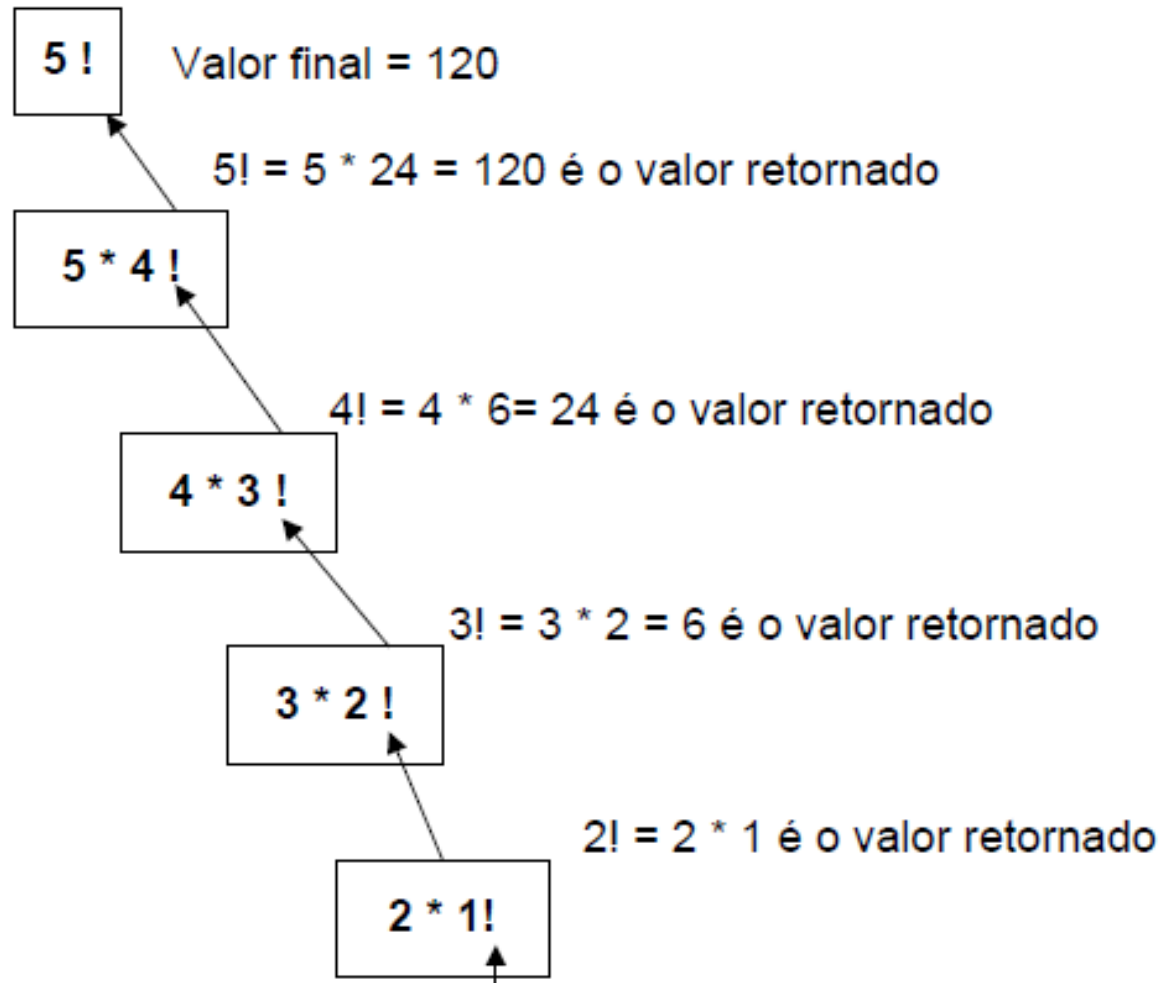
- Quando há uma recursão, um ambiente local é criado para cada chamada.
- As variáveis de chamadas recursivas são independentes entre si, como se estivéssemos chamando funções diferentes.

# Funções Recursivas

## • Exemplo 1: Fatorial

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int fatorial(int x)
5 {
6     if (x > 0)
7         return x * fatorial(x - 1);
8     return 1;
9 }
10 int main()
11 {
12     int numero, resultado;
13     printf("Digite o numero para o calculo da fatorial:\n");
14     scanf("%d", &numero);
15     resultado = fatorial(numero);
16     printf("Resultado: %d\n", resultado);
17     return 0;
18 }
```

# Funções Recursivas



# Funções Recursivas

- Exemplo 2: Sequência de Fibonacci

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int fibonacci(int n)
5 {
6     if (n > 2)
7         return (fibonacci(n - 1) + fibonacci(n - 2));
8     return 1;
9 }
10
11 int main()
12 {
13     int n;
14     printf("Digite um numero: ");
15     scanf("%d", &n);
16     printf("O termo %d da serie de Fibonacci e: %d\n", n, fibonacci(n));
17     return 0;
18 }
```

# Referências

- **MIZRAHI, V. V. Treinamento em Linguagem C. 2ª Edição. São Paulo: Person Prentice Hall, 2008.**
- **SCHILDT, H. C, Completo e Total. 3ª Edição revista e atualizada; Tradução e revisão técnica: Roberto Carlos Mayer. São Paulo: Makron Books, 1996.**