



# TÉCNICAS DE PROGRAMAÇÃO

Aula 06 – Funções por valor e referência




# Objetivos de Aprendizagem

1. Diferenciar função por passagem de parâmetro por valor da função por passagem de parâmetro por referência;
2. Desenvolver programas que utilizam funções como passagem de parâmetro por valor e funções com passagem de parâmetro por referência.




# Escopo de variáveis

- Definição de Escopo
    - O escopo de uma variável é a parte do código do programa onde a variável é visível e, portanto, pode ser utilizada.
    - Com relação ao escopo, as variáveis se dividem em:
      - Globais
      - Locais
- 



# Escopo de variáveis

- Variáveis Globais:
    - São as variáveis declaradas fora dos procedimentos e das funções;
    - São visíveis e podem ser utilizadas em toda a extensão do programa;
  - Variáveis Locais:
    - São as variáveis declaradas dentro dos procedimentos e das funções;
    - São visíveis e podem ser utilizadas apenas dentro do subprograma que as declarou.
- 

# Escopo de variáveis

- Exemplo de Escopo em uma Função:

```
int num1,num2, num3; ← Variáveis globais
int fat(int n) {
    int i, f = 1; ← Variáveis locais a fat
    for (i = n; i > 0; i--)
        f = f * i;
    return f;
}
void main () {
    printf("Digite o numero 1:");
    scanf ("%f", &num1);
    printf("Digite o numero 2:");
    scanf ("%f", &num2);
    if (num1 >= 0 && num2 >= 0)
    {
        num3 = fat(num1) + fat (num2);
        printf("%i",num3);
    }
}
```



# Parâmetros

- Parâmetros são utilizados em computação para possibilitar a construção de subprogramas genéricos.

```
void calcularMedia(float soma, int cont)
{
    float media;
    media = soma / cont;
    printf("Media = %f", media);
}
```





# Parâmetros

- Parâmetros Formais
  - Parâmetros formais são as variáveis declaradas no cabeçalho do subprograma.
- Parâmetros Reais
  - Parâmetros reais são as variáveis passadas no instante da chamada do subprograma.



# Parâmetros

- Exemplo:

Parâmetros Formais



```
int potencia (int base, int expoente) {  
    ...;  
}
```

Parâmetros Reais




```
void main() {  
    int resultado;  
    int a = 2, b = 3;  
    resultado = potencia (a, b);  
    printf("Resultado = %i", resultado);  
}
```





# Passagem de parâmetro por valor

- No instante da chamada do subprograma, o parâmetro formal recebe uma cópia do valor do parâmetro real correspondente.
  - Alterações feitas nos parâmetros formais não refletem nos parâmetros reais correspondentes.
  - Caracteriza-se por ser um mecanismo de entrada de dados para o subprograma.
- 

# Passagem de parâmetro por valor

- Exemplo:

```
int somaDobro(int a, int b) {  
    int soma;  
    a = 2 * a;  
    b = 2 * b;  
    soma = a + b;  
    return soma;  
}  
void main () {  
    int x, y, z;  
    scanf("%i", &x);  
    scanf("%i", &y);  
    z = somaDobro(x, y);  
    printf("A soma do dobro dos numeros %i  
    e %i eh %i", x, y, z);  
}
```

# Usando memória

- C foi projetado para ser uma linguagem de baixo nível que pode acessar facilmente os locais da memória e realizar operações relacionadas à memória.
- Por exemplo, a função **scanf ()** coloca o valor inserido pelo usuário no local ou **endereço** da variável. Isso é feito usando o símbolo **&**.

```
1  int num;  
2  printf ("Digite um número:");  
3  scanf ("% d", & num );  
4  printf ("% d", num);
```

- **&num** é o endereço da variável **num** .



# Usando memória

- Um endereço de memória é fornecido como um número **hexadecimal** .
- **Hexadecimal** , ou **hex** , é um sistema numérico de base 16 que usa dígitos de 0 a 9 e letras de A a F (16 caracteres) para representar um grupo de quatro dígitos binários que podem ter um valor de 0 a 15.
- É muito mais fácil de ler um número hexadecimal de 8 caracteres para 32 bits de memória do que tentar decifrar 32 1s e 0s em binário.

# Usando memória

- O programa a seguir exibe os endereços de memória para as variáveis **i** e **k** :

```
teste de vazio ( int k);

int main () {
    int i = 0;

    printf ("O endereço de i é% x \n", &i);
    teste (i);
    printf ("O endereço de i é% x \n", &i);
    teste (i);

    return 0;
}

void test ( int k) {
    printf ("O endereço de k é% x \n", &k);
}
```

# Usando memória

- Na instrução **printf**, **%x** é o especificador de formato hexadecimal.
- A saída do programa varia de execução para execução, mas é semelhante a:

```
O endereço de i é 846dd754  
O endereço de k é 846dd758  
O endereço de i é 846dd754  
O endereço de k é 846dd758
```

- O endereço de uma variável permanece o mesmo desde o momento em que é declarada até o final de seu escopo.



# O que é um Ponteiro?

- Os ponteiros são muito importantes na programação C porque permitem que você trabalhe facilmente com locais de memória.
- Eles são fundamentais para arrays, strings e outras estruturas de dados e algoritmos.
- Um ponteiro é uma variável que contém o endereço de outra variável. Em outras palavras, ele "aponta" para o local atribuído a uma variável e pode acessar indiretamente a variável.

# O que é um Ponteiro?

- Os ponteiros são declarados usando o símbolo \* e assumem a forma:
  - **pointer\_type\* identificador;**
- **pointer\_type** é o tipo de dado para o qual o ponteiro estará apontando. O tipo de dado do ponteiro real é um número hexadecimal, mas ao declarar um ponteiro, você deve indicar para qual tipo de dado ele estará apontando.
- O asterisco \* declara um ponteiro e deve aparecer próximo ao identificador usado para a variável do ponteiro.



# O que é um Ponteiro?

- O programa a seguir demonstra variáveis, ponteiros e endereços:

```
int j = 63;  
int * p = NULL;  
p = &j;  
  
printf("O endereço de j é% x \n", &j);  
printf("p contém o endereço% x \n", p);  
printf("O valor de j é% d \n", j);  
printf("p está apontando para o valor% d \n", * p);
```

```
O endereço de j é ff3652cc  
p contém o endereço ff3652cc  
O valor de j é 63  
p está apontando para o valor 63
```

# Operador de Desreferência

- Há várias coisas a serem observadas sobre este programa:
  - Os ponteiros devem ser inicializados como **NULL** até que sejam atribuídos a um local válido.
  - Os ponteiros podem ser atribuídos ao endereço de uma variável usando o sinal **e comercial &**.
  - Para ver o que um ponteiro está apontando, use **\*** novamente, como em **\*p** . Nesse caso, o **\*** é chamado de operador de indireção ou **desreferência** . O processo é denominado **desreferenciamento** .



# Ponteiro para Ponteiro

- Alguns algoritmos usam **um ponteiro para um ponteiro** . Este tipo de declaração de variável usa **\*\*** , e pode ser atribuído ao endereço de outro ponteiro , como em:

```
int x = 12;  
int *p = NULL  
int **ptr = NULL;  
p = &x;  
ptr = &p;
```



# Ponteiros em expressões

- Ponteiros podem ser usados em **expressões** como qualquer variável.
- Os operadores aritméticos podem ser aplicados a qualquer coisa para a qual o ponteiro esteja apontando.

# Ponteiros em Expressões

- Por exemplo:

```
int x = 5;  
int y;  
int * p = NULL;  
p = & x;  
  
y = * p + 2; /* y é atribuído a 7 */  
y += * p; /* y é atribuído a 12 */  
* p = y; /* x é atribuído 12 */  
(* p) ++; /* x é incrementado para 13 */  
  
printf ("p está apontando para o valor% d \n", * p);
```

- Observe que os parênteses são necessários para o operador ++ aumentar o valor que está sendo apontado. O mesmo é verdade ao usar o operador -.



# Parâmetros por Referência

- Uma array (vetor ou matriz) não pode ser passada por valor para uma função.
- Entretanto, um nome de array é um ponteiro , então apenas passar um nome de array para uma função é passar um ponteiro para o array .

# Exemplo

```
int add_up ( int * a , int num_elements);

int main () { ordens
  int [5] = {100, 220, 37, 16, 98};

  printf ("O total de pedidos é% d \ n", add_up (pedidos, 5));

  return 0;
}

int add_up ( int * a , int num_elements) {
  int total = 0;
  int k;

  para (k = 0; k < núm_elementos; k ++) {
    total + = a [k];
  }

  retorno (total);
}
```



# Funções que retornam um array

- Assim como um ponteiro para uma array pode ser passado para uma função, um ponteiro para uma array pode ser retornado.



# Exemplo

```
int * get_evens ();

int main () {
    int * a;
    int k;

    a = get_evens (); /* obtém os primeiros 5 números pares */
    para (k = 0; k <5; k ++)
        printf ("%d \n", a [k]);

    return 0;
}

int * get_evens () {
    static int nums [5];
    int k;
    int par = 0;

    para (k = 0; k <5; k ++) {
        nums [k] = par += 2;
    }

    return (nums);
}
```



# Funções que retornam um array

- Observe que um ponteiro , não uma matriz , é declarado para armazenar o valor retornado pela função.
- Observe também que quando uma variável local está sendo passada para fora de uma função, você precisa declará-la como **estática** na função.
- Lembre-se de que  $a[k]$  é igual a  $* (a + k)$  .