SI100

Algoritmos e Programação de Computadores I

1º Semestre - 2018

Tópico 9 – Parte 2
Modularização de Programas

Unidade 14

Prof. Guilherme Palermo Coelho

guilherme@ft.unicamp.br

Roteiro

- Escopo de Variáveis;
- Passagem de parâmetros por valor e por referência;
- Vetores como parâmetros;
- Exercícios;
- Referências.

Recapitulando

- **Procedimentos**: contêm um conjunto de comandos que são executados quando o procedimento é chamado.
- **Funções**: semelhante aos procedimentos - retornam um valor

```
#include <stdio.h>

int soma(int X, int Y); //Protótipo da função

int main() { // Programa principal
        printf("Resultado = %d\n", soma(15, 25));
        return 0;
}

int soma(int X, int Y)//Definição da função
{
        return (X+Y);
}
```

Escopo de Variáveis

Escopo de Variáveis

- Na linguagem C, duas ou mais variáveis **não podem ser declaradas com o mesmo nome** se fazem parte da **mesma função**;
- Mas elas podem ter o mesmo nome se forem declaradas em **funções diferentes**;
- Neste último caso, elas se comportarão como **variáveis distintas**, apesar de apresentarem o mesmo nome:
  - São chamadas ***variáveis locais:***
    - Só podem ser referenciadas dentro do ***bloco*** em que foram definidas;
    - **Bloco**: conjunto de comandos dentro de "{" "}".
    - Tais variáveis são **criadas** sempre que se entra no bloco e **destruídas** no final do bloco.

Escopo de variáveis – Exemplo 1

```
#include <stdio.h>

void FUNC1() {
        int B = -100; //Só pode ser usada dentro de FUNC1
        printf("B em FUNC1: %d\n", B);
}
void FUNC2() {
        int B = 20; //Só pode ser usada dentro de FUNC2
        printf("B em FUNC2: %d\n", B);
}

int main() {
        int B = -500;
        printf("B em main(): %d\n", B);
        FUNC1();
        FUNC2();
        return 0;
}
```

```
B em main(): -500
B em FUNC1: -100
B em FUNC2: 20
```

Escopo de variáveis – Exemplo 2

```
// Soma de 10 números digitados pelo usuário
#include <stdio.h>

int soma(int x , int y)
{
        int A = x+y;
        return A;
}
int main() {
        int A, x;
        int res = 0;

        for (x=0;x<10;x++) {
           printf("Entre com A: ");
           scanf("%d", &A);
           res = soma(res, A);
        }
        printf("\nO Resultado e: %d\n", res);
        return 0; }
```

```
Entre com A: 1
Entre com A: 1
Entre com A: 1
Entre com A: 1
Entre com A: 1
Entre com A: 1
Entre com A: 1
Entre com A: 1
Entre com A: 1
Entre com A: 1

O Resultado e: 10
```

Escopo de Variáveis – Variáveis Globais

- Além das variáveis locais, existem também na linguagem C as chamadas ***variáveis globais***:
  - São reconhecidas no **programa inteiro** → podem ser usadas em qualquer trecho de código;
  - Elas guardam seus valores durante toda a execução do programa;
  - Devem ser declaradas **fora** de qualquer função do programa;
  - Se em um determinado bloco existir uma ***variável local de mesmo nome***, todas as referências do bloco serão a esta variável local (e não à variável global);
  - O uso de variáveis globais requer **cuidados** quando se está desenvolvendo programas grandes → há chances de se modificar acidentalmente (e incorretamente) o valor da variável global.

Escopo de variáveis – Exemplo 3

```
#include <stdio.h>

int A;  // A é uma variável global

void FUNC3() {
        int A = -100; //A tem escopo local
        printf("A em FUNC3: %d\n", A);
}
void FUNC4() {
        printf("A em FUNC4: %d\n", A);
}

int main() {
        A = 500; // Usa a variável global já declarada
        printf("A em main(): %d\n", A);
        FUNC3();
        FUNC4();
        return 0;
}
```

```
A em main(): 500
A em FUNC3: -100
A em FUNC4: 500
```

Passagem de Parâmetros por Valor e por Referência

Passagem de Parâmetros por Valor

- Os parâmetros passados para as funções dos exemplos vistos até agora consistiam em ***cópias*** dos valores passados na chamada da função → parâmetros passados ***por valor***.
- **Exemplo**:

```
#include <stdio.h>

double fat(int x) {
   double fat = 1.0;
   while (x > 1) {
        fat = fat*x;
        x--;
   }
   return fat;
} //continua
```

```
//continuação

int main() {
   int n = 4;
   printf("n! = %lf\n", fat(n));
   printf("n = %d\n", n);

   return 0;

}
```

```
n! = 24.000000
n = 4
```

Passagem de Parâmetros por Referência

- Apesar da passagem de parâmetros por valor já nos dar uma boa flexibilidade, existem situações em que é desejável alterar o valor das variáveis passadas como parâmetro de forma que isto se ***reflita também fora do escopo da função***.
- Nestes casos, é possível passar os parâmetros para uma função através da chamada ***passagem por referência***.
  - A função cria uma **cópia do *endereço*** da variável (um ***apontador*** para a posição de memória em que ela está armazenada) ao invés de uma cópia do seu conteúdo.
  - As modificações são feitas na posição de memória em que a variável está guardada, se refletindo em todos os blocos do programa que têm acesso a esta variável.

Passagem de Parâmetros por Referência

- **Exemplo**: passagem de parâmetros por referência.

```
#include <stdio.h>

void troca (int *x, int *y) {
        int aux;
        aux=*x;
        *x = *y;
        *y = aux;
}
int main() {
        int a = 5;
        int b = 10;
        troca(&a, &b);
        printf("a = %d | b = %d\n", a, b);
        return 0;
}
```

x e y são apontadores para posições de memória que guardam valores do tipo int.

aux recebe o conteúdo da posição de memória de endereço guardado em x.

a posição de memória cujo endereço está em x recebe o valor da posição de memória cujo endereço está em y.

São passados os endereços de a e b.

```
a = 10 | b = 5
```

Passagem de Parâmetros por Referência

- A passagem de parâmetros por referência utiliza o conceito de ***apontadores*** (ou ***ponteiros***) da linguagem C.
- **Apontador em C:** variável que armazena o ***endereço*** de uma posição de memória, ou seja, o ***endereço de outra variável***.
  - Cada ponteiro tem um ***tipo*** associado (int, float, struct pessoa ...);
  - São declarados com um '*' antes do nome:
    - **Ex.**: int *x; //x é um ponteiro para variáveis int
  - Armazenam endereços passados com o comando &:
    - **Ex.**: x = &a; //x recebe o endereço da variável a;
  - O conteúdo da posição de memória apontada por um ponteiro x é acessado por *x:
    - **Ex.**: *x = 10; //pos. de mem. com endereço em x recebe 10;

Passagem de Parâmetros por Referência

- **Exemplo**: apontadores.

```
#include <stdio.h>

int main() {
        int X = 10;      //X contém o valor 10
        int *p, *q;      //p e q contêm lixo

        p = &X;          //p recebe o end. de X (aponta para X)
        q = p;           //q também aponta para X;

        *q = *p + 10; //conteúdo do end. q recebe o conteúdo
                      //do end. p somado com o valor 10.

        printf("X: %d, *p: %d, *q: %d\n", X, *p, *q);
        printf("&X: %p, p: %p, q: %p\n", &X, p, q);
        return 0;}
```

```
X: 20, *p: 20, *q: 20
&X: 0x7fff561e7b14, p: 0x7fff561e7b14, q: 0x7fff561e7b14
```

VETORES COMO PARÂMETROS

Vetores como parâmetros

- Vetores também podem ser passados como parâmetros para funções → eles ***sempre*** são passados por **referência**;
  - Se os valores de cada posição do vetor forem alterados dentro da função, esta alteração **se refletirá fora da função**.

- Em C, é possível passar vetores para uma função de três maneiras diferentes (todas funcionam da **mesma forma**):
  - <tipo> funcao(**int v[10]**,...); //forma comum, como declarado
  - <tipo> funcao(**int v[]**,...); //tamanho do vetor omitido
  - <tipo> funcao(**int *v**, ...); //apontador para a primeira posição //do vetor

Vetores como parâmetros

- **Exemplo**: passagem de vetor como parâmetro;

```
#include <stdio.h>
#define TAM 10

void MostraVet1(int v[TAM]) {
        int j;
        for(j=0; j<TAM; j++)
                printf("%d ",v[j]);
}

void MostraVet2(int v[], int n) {
        int j;
        for(j=0; j<n; j++)
                printf("%d ",v[j]);
}
//continua...
```

Vetores como parâmetros

- **Exemplo**: passagem de vetor como parâmetro;

```
//continuação
void MostraVet3(int *v, int n) {
        int j;
        for(j=0; j<n; j++)
                printf("%d ",v[j]);
}
int main() {
        int vetor[TAM];
        int i;
        for (i=0;i<TAM;i++)
                vetor[i] = i*i;
        MostraVet1(vetor); printf("\n");
        MostraVet2(vetor, TAM); printf("\n");
        MostraVet3(vetor, TAM); printf("\n");
        return 0;
}
```

```
0 1 4 9 16 25 36 49 64 81
0 1 4 9 16 25 36 49 64 81
0 1 4 9 16 25 36 49 64 81
```

Vetores como parâmetros

- **ATENÇÃO**: quando os parâmetros são matrizes, **apenas** o tamanho da **primeira** dimensão pode ser omitido;
  - Os demais devem estar explícitos no protótipo (e na definição) da função;
  - **Exemplos**:
    - void MostraMatriz(int **m[][40]**,...);
    - void MostraMatriz(**int m[10][40]**,...);
    - void Mostra3D(**int m[][10][20]**,...);
    - void Mostra4D(**int m[][10][20][30]**,...);
  - **Exemplo ERRADO:**
    - void MostraMatriz(**int m[][]**,...); //**Errado**! **Cuidado**!

EXERCÍCIOS

Exercícios

1. Escreva um procedimento **media** que receba duas notas (tipo *float*) e um caractere. Caso este caractere seja 'A', o procedimento deve calcular a ***média aritmética*** das duas notas. Caso seja 'B', a ***média geométrica*** (raiz quadrada do produto das duas notas) deve ser calculada. Escreva também um programa que leia os valores das duas notas e do caractere, chame o procedimento **media** e imprima o resultado na tela. No procedimento **media**, o valor calculado deve ser armazenado e retornado em um dos **parâmetros**.

2. Escreva um procedimento que receba dois vetores de inteiros A e B, de tamanho 10, e armazene em B o cubo de cada elemento contido em A. O programa principal deve ler os dez valores de A e exibir os valores armazenados em B após a chamada do procedimento.

Exercícios

3. Faça duas funções que implementem a soma e a multiplicação de dois números complexos *z* e *w*. Estas operações são definidas por:

$$z + w = (a + bi) + (c + di) = (a + c) + (b + d)i$$

$$z \times w = (a + bi) \times (c + di) = (ac - bd) + (ad + bc)i$$

Sendo assim, cada função deve receber quatro parâmetros: a parte real e a imaginária de *z* e a parte real e a imaginária de *w*. O resultado deve estar nos dois primeiros parâmetros (substituir o valor de *z*).

Exercícios

4. Sem implementar o programa abaixo, mostre o que ele imprime na tela:

```
#include <stdio .h>
void f(int a);

int main() {
        int a;
        a = 1;
        printf(”1: a=%d \n”, a);
        f(a);
        printf(”3: a=%d \n”, a);
        return 0 ;
}

void f(int a) {
        a = 2;
        printf(”2: a=%d \n”, a);
}
```

Exercícios

5. Modifique a função ***void f(int a)*** e a sua chamada no programa principal do Ex. 4 para que a resposta agora seja:

   1: a=1

   2: a=2

   3: a=2

6. Faça um procedimento que receba um **vetor de inteiros** de **tamanho N** e retorne o menor e o maior elemento do vetor. O procedimento **não deve** ler nada do teclado, nem mostrar nada na tela. Também não deve receber nenhum parâmetro adicional. Depois, faça um ***main()*** para testar o procedimento (no programa principal, o vetor pode ser definido no próprio código, mas a saída deve ser na tela).

Exercícios

- **Observação**:
  - Os seguintes exercícios devem ser entregues via SuSy.
    - Exercício 2;
    - Exercício 3;
    - Exercício 6.
  - Veja os enunciados atualizados no site do sistema:
    - https://susy.ic.unicamp.br:9999/si100a (Turma A);
    - https://susy.ic.unicamp.br:9999/si100b (Turma B).

Referências

Referências

- MIZRAHI, V. V., *Treinamento em Linguagem C – Curso Completo – Módulo 1*. 2a Edição, Pearson Makron Books, 2005.
- SCHILDT, H. *C Completo e Total*. Makron Books, 1996.
- ASCENCIO, A. F. G. & DE CAMPOS, E. A. V., *Fundamentos da Programação de Computadores – Algoritmos, Pascal e C/C++*. Pearson Prentice Hall, 2003.