

MC-102 — Aula 16

Funções II

Instituto de Computação – Unicamp

19 de Abril de 2016

Roteiro

- 1 Escopo de Variáveis: variáveis locais e globais
- 2 Exemplo Utilizando Funções
- 3 Vetores, Matrizes e Funções
 - Vetores em funções
 - Vetores multi-dimensionais e funções
- 4 Exercícios

Variáveis locais e variáveis globais

- Uma variável é chamada **local** se ela foi declarada dentro de uma função. Nesse caso ela existe somente dentro da função, e após o término da execução desta, a variável deixa de existir. **Variáveis parâmetros também são variáveis locais**
- Uma variável é chamada **global** se ela for declarada fora de qualquer função. Essa variável é visível em todas as funções. Qualquer função pode alterá-la e ela existe durante toda a execução do programa.

Organização de um Programa

- Em geral um programa é organizado da seguinte forma:

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

Protótipos de funções

Declaração de Variáveis Globais

```
int main(){
    Declaração de variáveis locais
    Comandos;
}
```

```
int fun1(Parâmetros){ //Parâmetros também são variáveis locais
    Declaração de variáveis locais
    Comandos;
}
```

```
int fun2(Parâmetros){ //Parâmetros também são variáveis locais
    Declaração de variáveis locais
    Comandos;
}
```

```
...
...
...
```

Escopo de variáveis

- O **escopo** de uma variável determina de quais partes do código ela pode ser acessada, ou seja, de quais partes do código a variável é visível.
- A regra de escopo em C é bem simples:
 - ▶ As variáveis globais são visíveis por todas as funções.
 - ▶ As variáveis locais são visíveis apenas na função onde foram declaradas.

Escopo de variáveis

```
#include<stdio.h>

void fun1();
int fun2(int local_b);

int global;

int main() {
    int local_main;
    /* Neste ponto são visíveis global e local_main */
}

void fun1() {
    int local_a;
    /* Neste ponto são visíveis global e local_a */
}

int fun2(int local_b){
    int local_c;
    /*Neste ponto são visíveis global, local_b e local_c*/
}
```

Escopo de variáveis

- É possível declarar variáveis locais com o mesmo nome de variáveis globais.
- Nesta situação, a variável local “esconde” a variável global.

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

```
void fun();
```

```
int nota = 10;
```

```
int main(){  
    nota = 20;  
    fun();  
}
```

```
void fun() {  
    int nota;  
    nota = 5;  
    /* Neste ponto nota é a variável local de fun. */  
}
```

Exemplo 1

```
#include <stdio.h>

void fun1();
void fun2();

int x;
int main(){
    x = 1;
    fun1();
    fun2();
    printf("main: %d\n", x);
}

void fun1(){
    x = x + 1;
    printf("fun1: %d\n",x);
}

void fun2(){
    int x = 3;
    printf("fun2: %d\n",x);
}
```

O que será impresso ?

Exemplo 2

```
#include <stdio.h>

void fun1();
void fun2();

int x = 1;
int main(){
    int x=1;
    fun1();
    fun2();
    printf("main: %d\n", x);
}

void fun1(){
    x = x + 1;
    printf("fun1: %d\n",x);
}

void fun2(){
    int x = 4;
    printf("fun2: %d\n",x);
}
```

O que será impresso ?

Exemplo 3

```
#include <stdio.h>

void fun1();
void fun2(int x);

int x = 1;
int main(){
    x=2;
    fun1();
    fun2(x);
    printf("main: %d\n", x);
}

void fun1(){
    x = x + 1;
    printf("fun1: %d\n",x);
}

void fun2(int x){
    x = x + 1 ;
    printf("fun2: %d\n",x);
}
```

O que será impresso ?

Variáveis locais e variáveis globais

- O uso de variáveis globais deve ser evitado pois é uma causa comum de erros:
 - ▶ Partes distintas e funções distintas podem alterar a variável global, causando uma grande interdependência entre estas partes distintas de código.
- A legibilidade do seu código também piora com o uso de variáveis globais:
 - ▶ Ao ler uma função que usa uma variável global é difícil inferir seu valor inicial e portanto qual o resultado da função sobre a variável global.

Exemplo Utilizando Funções

- Em uma das aulas anteriores vimos como testar se um número em **candidato** é primo:

```
divisor = 2;
eprimo=1;
while(divisor<=candidato/2) {
    if(candidato % divisor == 0){
        eprimo=0;
        break;
    }
    divisor++;
}
if(eprimo)
    printf(" %d, ", candidato);
```

Exemplo Utilizando Funções

- Depois usamos este código para imprimir os n primeiros números primos:
- Veja no próximo slide.

Exemplo Utilizando Funções

```
int main(){
    int divisor=0, n=0, eprimo=0, candidato=0, primosImpr=0;
    printf("\n Digite numero de primos a imprimir:");
    scanf("%d",&n);
    if(n>=1){
        printf("2, ");
        primosImpr=1;
        candidato=3;
        while(primosImpr < n){
            divisor = 2;
            eprimo=1;
            while( divisor <= candidato/2 ){
                if(candidato % divisor == 0){
                    eprimo=0;
                    break;
                }
                divisor++;
            }
            if(eprimo){
                printf("%d, ",candidato);
                primosImpr++;
            }
            candidato=candidato+2;//Testa proximo numero
        }
    }
```

Exemplo Utilizando Funções

- Podemos criar uma função que testa se um número é primo ou não (note que isto é exatamente um bloco logicamente bem definido).
- Depois fazemos chamadas para esta função.

Exemplo Utilizando Funções

```
#include <stdio.h>

int ePrimo(int candidato); //retorna 1 se candidato é primo, e 0 caso contrário

int main(){
    int n=0, candidato=0, primosImpr=0;

    printf("Digite numero de primos:");
    scanf("%d",&n);
    if(n >= 1){
        printf("2, ");
        primosImpr = 1;
        candidato = 3;
        while(primosImpr < n){
            if( ePrimo(candidato) ){
                printf("%d, ",candidato);
                primosImpr++;
            }
            candidato=candidato+2;
        }
    }
}
```


Exemplo Utilizando Funções

```
int ePrimo(int candidato){
    int divisor;

    divisor = 2;
    while( divisor <= candidato/2){
        if(candidato % divisor == 0){
            return 0;
        }
        divisor++;
    }
    //Se terminou o laço então candidato é primo
    return 1;
}
```

Vetores em funções

- Vetores também podem ser passados como parâmetros em funções.
- Ao contrário dos tipos simples, vetores têm um comportamento diferente quando usados como parâmetros de funções.
- Quando uma variável simples é passada como parâmetro, seu valor é atribuído para uma nova variável local da função.
- No caso de vetores, **não é criado** um novo vetor!
- Isto significa que os valores de um vetor **são alterados** dentro de uma função!

Vetores em funções

```
#include <stdio.h>

void fun1(int vet[], int tam){
    int i;
    for(i=0;i<tam;i++)
        vet[i]=5;
}

int main(){
    int x[10];
    int i;

    for(i=0;i<10;i++)
        x[i]=8;

    fun1(x,10);
    for(i=0;i<10;i++)
        printf("%d\n",x[i]);
}
```

O que será impresso?

Vetores em funções

- No exemplo anterior note que a função **fun1** recebe o vetor como parâmetro e um inteiro que especifica o seu tamanho.

```
void fun1(int vet[], int tam){
    int i;
    for(i=0;i<tam;i++)
        vet[i]=5;
}
```

- Esta é a forma padrão para se receber um vetor como parâmetro.
- Um vetor possui um tamanho definido, mas em geral usa-se menos posições do que o seu tamanho. Além disso a função pode operar sobre vetores de diferentes tamanhos, bastando informar o tamanho específico de cada vetor na variável **tam**.

Vetores em funções

- Vetores não podem ser devolvidos por funções.

```
#include <stdio.h>

int[] leVet() {
    int i, vet[100];
    for (i = 0; i < 100; i++) {
        printf("Digite um numero:");
        scanf("%d", &vet[i]);
    }
    return vet;
}
```

- O código acima não compila, pois não podemos retornar um **int[]** .

Vetores em funções

- Mas como um vetor é alterado dentro de uma função, podemos criar a seguinte função para leitura de vetores.

```
#include <stdio.h>

void leVet(int vet[], int tam){
    int i;
    for(i = 0; i < tam; i++){
        printf("Digite numero:");
        scanf("%d",&vet[i]);
    }
}
```

- A função abaixo faz a impressão de um vetor.

```
void escreveVet(int vet[], int tam){
    int i;
    for(i=0; i< tam; i++)
        printf("vet [%d] = %d\n",i,vet[i]);
}
```

Vetores em funções

- Podemos usar as funções anteriores no programa abaixo.

```
int main(){
    int vet1[10], vet2[20];

    printf(" ----- Lendo Vetor 1 -----\n");
    leVet(vet1,10);
    printf(" ----- Lendo Vetor 2 -----\n");
    leVet(vet2,20);

    printf(" ----- Imprimindo Vetor 1 -----\n");
    escreveVet(vet1,10);
    printf(" ----- Imprimindo Vetor 2 -----\n");
    escreveVet(vet2,20);

}
```

Vetores multi-dimensionais e funções

- Ao passar um **vetor simples** como parâmetro, não é necessário fornecer o seu tamanho na declaração da função.
- Quando o **vetor é multi-dimensional** a possibilidade de não informar o tamanho na declaração se restringe à primeira dimensão apenas.

```
void mostra_matriz(int mat[][10], int n) {  
    ...  
}
```


Vetores multi-dimensionais e funções

- Pode-se criar uma função deixando de indicar a primeira dimensão:

```
void mostra_matriz(int mat[][10], int n) {  
    ...  
}
```

- Ou pode-se criar uma função indicando todas as dimensões:

```
void mostra_matriz(int mat[10][10], int n) {  
    ...  
}
```

- Mas não pode-se deixar de indicar outras dimensões (exceto a primeira):

```
void mostra_matriz(int mat[10][], int n) {  
    //ESTE NÃO FUNCIONA  
    ...  
}
```

Vetores multi-dimensionais e funções

- É comum definirmos uma constante com o tamanho máximo de matrizes e vetores multi-dimensionais, e passarmos os tamanhos efetivamente utilizados como parâmetros para funções que operam sobre matrizes ou vetores-multidimensionais.

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

void imprimeMatriz(int mat[MAX][MAX], int lin, int col) {
    int i, j;

    for (i = 0; i < lin; i++) {
        for (j = 0; j < col; j++)
            printf("%d\t", mat[i][j]);
        printf("\n");
    }
}
```

Vetores multi-dimensionais em funções

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

void imprimeMatriz(int mat[MAX][MAX], int lin, int col) {
    int i, j;

    for (i = 0; i < lin; i++) {
        for (j = 0; j < col; j++)
            printf("%d\t", mat[i][j]);
        printf("\n");
    }
}

int main() {
    int mat[MAX][MAX] = { { 0, 1, 2, 3, 4, 5},
                          {10, 11, 12, 13, 14, 15},
                          {20, 21, 22, 23, 24, 25},
                          {30, 31, 32, 33, 34, 35},
                          {40, 41, 42, 43, 44, 45},
                          {50, 51, 52, 53, 54, 55},
                          {60, 61, 62, 63, 64, 65},
                          {70, 71, 72, 73, 74, 75}};

    imprimeMatriz(mat, 8, 6);
    return 0;
}
```

Vetores multi-dimensionais em funções

- Lembre-se que vetores (multi-dimensionais ou não) são alterados quando passados como parâmetro em uma função.

```
void teste(int mat[MAX][MAX], int lin, int col) {
    int i, j;

    for (i = 0; i < lin; i++) {
        for (j = 0; j < col; j++){
            mat[i][j] = -1;
        }
    }
}

int main() {
    int mat[MAX][MAX] = { { 0, 1},
                          { 2, 3 } };

    teste(mat, 2, 2);
    return 0;
}
```

- Qual o conteúdo de **mat** após a execução da função **teste**?

Exercício

- Escreva uma função em C para computar a raiz quadrada de um número positivo. Use a idéia abaixo, baseada no método de aproximações sucessivas de Newton. A função deverá retornar o valor da vigésima aproximação.

Seja Y um número, sua raiz quadrada é raiz da equação

$$f(x) = x^2 - Y.$$

A primeira aproximação é $x_1 = Y/2$. A $(n + 1)$ -ésima aproximação é

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Exercício

- Escreva uma função em C que recebe como parâmetros duas matrizes quadradas $n \times n$ e computa a soma destas ($n \leq 100$). O protótipo da função deve ser:

```
void somaMat(double mat1[100][100], double mat2[100][100],  
             double matRes[100][100], int n);
```

- As matrizes **mat1** e **mat2** devem ser somadas e o resultado atribuído à **matRes**. O parâmetro **n** indica as dimensões das matrizes.

Exercício

- Escreva uma função em C que recebe como parâmetros duas matrizes quadradas $n \times n$ e computa a multiplicação destas ($n \leq 100$). O protótipo da função deve ser:

```
void multiplicaMat(double mat1[100][100], double mat2[100][100],  
                  double matRes[100][100], int n);
```

- As matrizes **mat1** e **mat2** devem ser multiplicadas e o resultado atribuído à **matRes**. O parâmetro **n** indica as dimensões das matrizes.