MC-102 — Aula 12
Funções II

Instituto de Computação – Unicamp

17 de Abril de 2012

Roteiro

Variáveis locais e variáveis globais

- Uma variável é chamada local se ela foi declarada dentro de uma função. Nesse caso, ela existe somente dentro daquela função e após o término da execução da mesma, 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.


```
#include <stdio.h>
#include ...
Protótipos de funções
Declaração de Variáveis Globais
int main(){
  Declaração de variáveis locais
  Comandos;
}

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

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

...
...
```

Escopo de variáveis

- O escopo de uma variável determina de quais partes do código ela pode ser acessada.
- 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.

```
int nota = 10;
void a() {
  int nota = 5;
  /* Neste ponto nota é a variável local (com valor 5). */
}
```


Escopo de variáveis

```
#include <stdio.h>

void fun1();
void fun2();

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

void fun1(){
  x = 3;
  printf("\n%d",x);
}

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

O que é impresso ?

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]);
}
```


Vetores em funções

- Vetores não podem ser devolvidos por funções.
- Mas mesmo assim podemos fazer algo parecido com isso usando o fato de que vetores são alterados dentro de funções.

```
#include <stdio.h>

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

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:

```
#include <stdio.h>

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

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

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

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

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

}
```


Vetores multi-dimensionais (matrizes) 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_linhas) {
  ...
}
```


Vetores em funções

```
void mostra_matriz(int mat[][10], int n_linhas) {
  int i, j;

  for (i = 0; i < n_linhas; i++) {
    for (j = 0; j < 10; j++)
      printf("%2d ", mat[i][j]);
    printf("\n");
  }
}

int main() {
  int mat[][10] = { { 0,  1,  2,  3,  4,  5,  6,  7,  8,  9},
                    {10, 11, 12, 13, 14, 15, 16, 17, 18, 19},
                    {20, 21, 22, 23, 24, 25, 26, 27, 28, 29},
                    {30, 31, 32, 33, 34, 35, 36, 37, 38, 39},
                    {40, 41, 42, 43, 44, 45, 46, 47, 48, 49},
                    {50, 51, 52, 53, 54, 55, 56, 57, 58, 59},
                    {60, 61, 62, 63, 64, 65, 66, 67, 68, 69},
                    {70, 71, 72, 73, 74, 75, 76, 77, 78, 79}};

  mostra_matriz(mat, 8);
  return 0;
}
```


Vetores multi-dimensionais e funções

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

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

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

```
void mostra_matriz(int mat[5][10], int n_linhas) {
  ...
}
```

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

```
void mostra_matriz(int mat[5][], int n_linhas) {
  //ESTE NÃO FUNCIONA
  ...
}
```

Arquivos-Cabeçalhos ("header", .h)

Diretiva #include

- Inclui arquivos no ponto de chamada
- Texto do arquivo é "expandido" no ponto da chamada
- Sistema operacional é responsável por "encontrar o arquivo"

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


"Nossos" Arquivos-Cabeçalhos

- Arquivos com extensão ".h"
- Atenção: #include "arquivo.h"
- Mas o que são arquivos-cabeçalhos? arquivos que contém protótipos de funções
- Para que servem? para aproveitar uma função em vários programas futuros!

Exemplo:
ePar.c
soma2.c
multiplica2.c
funcoesCabecalho.h
progPrincipal.c

Usando "Nossos" Arquivos-Cabeçalhos

ePar.c
soma2.c
multiplica2.c
funcoesCabecalho.h
progPrincipal.c

```
gcc progPrincipal.c soma2.c multiplica2.c ePar.c -o saida
```

ou

```
gcc -c progPrincipal.c soma2.c multiplica2.c ePar.c
gcc progPrincipal.o soma2.o multiplica2.o ePar.o -o saida
```


Exercício 1

Escreva um programa com as seguintes funções:

- Uma função para preencher uma matriz 4×5 (deve receber como parâmetros a matriz, numLinhas e numColunas);
- Uma função para imprimir esta matriz;
- Uma função para verificar a quantidade de valores na matriz que são maiores do que 20 ou que são menores do que 10;
- Uma função para calcular a média dos valores ímpares presentes na matriz (a média deve ser uma variável global).

Exercício 2

Modifique o Exercício 1 considerando Arquivos-Cabeçalhos.