Funções em C - Passagem de parâmetros

Prof. Marcelo Veiga Neves
marcelo.neves@pucrs.br

Funções em C

- "Segmento de programa independente que executa alguma tarefa específica"
- "Conjunto de comandos agrupados em um bloco de código que recebe um nome e através deste pode ser ativado"
- Como já vimos: uma função é obrigatória (*main*)

Definição de funções

- Uma definição de função não pode estar embutida em uma outra
- As funções são independentes umas das outras
- A mesma função pode ser acessada de diferentes partes do programa

Declaração X Definição

- Declaração (funções devem ser declaradas antes)
  - *Protótipo* é o cabeçalho da declaração da função (tipo de retorno + nome + parâmetros)
- Definição
  - Contém todos os comandos da função e inclui a declaração
- Caso uma função não seja definida antes da *main* (ou antes de uma função que a chama) é obrigatório o uso de um protótipo

Exemplo

```c
#include <stdio.h>

void quadrado();          // protótipo

void quadrado()           // sem parâmetros ou retorno
{                         // corpo da função
    int k;

    for (k = 1; k <= 10; k++) {
        printf("%d\n", k*k);
    }
}

int main()
{
    quadrado();           // chamada da função
}
```

Passagem de parâmetros

- Os parâmetros são passados para uma função de acordo com a sua posição
- Os *parâmetros formais* de uma função se comportam como variáveis locais (criados na entrada e destruídos na saída)
- Os argumentos podem ser passados de duas maneiras:
  - Passagem por valor
  - Passagem por referência

Passagem por valor

- Copia o valor do argumento no parâmetro formal
- Alterações feitas nos parâmetros da função não têm efeito nas variáveis usadas para chamá-la

```
int multiplica (int x)
{
    x = x * x;
    return x;
}

int main()
{
    int t = 10;
    printf("%d - %d\n", multiplica(t), t);
}
```

Demonstração

- Execute o exemplo anterior (será necessário incluir o header *stdio.h*) em um dos ambientes abaixo:
  1. https://replit.com/languages/c
  2. https://techiedelight.com/compiler/
  3. https://www.mycompiler.io/new/c
  4. https://onecompiler.com/c
  5. https://www.tutorialspoint.com/online_c_compiler.php
  6. https://www.programiz.com/c-programming/online-compiler/
  7. https://www.onlinegdb.com/online_c_compiler
- Alternativamente, no Linux, utilize um editor de textos qualquer (como *geany*, *vim*, entre outros) para editar o código fonte e salvar o mesmo com a extensão .c
- E utilize o compilador *gcc*, para depois executar o programa:

```
$ gcc programa.c -o programa
$ ./programa
```

Passagem por referência

- É passado não é o valor de uma variável, mas sim uma *referência* para ela
- Alterações feitas nos parâmetros da função afetam as variáveis usadas
- Em C, a passagem por referência é feita com ponteiros[1]
  - Variável que armazena o endereço de memória de outra variável
  - Por enquanto, trataremos apenas do seu uso no contexto da passagem de parâmetros

[1] Na verdade, em C apenas realiza-se passagem de parâmetros por valor. Passagem por referência é emulada passando-se o endereço de uma variável (por valor)

Ponteiros na passagem por referência

- Recebendo e alterando um valor dentro da função

```c
void multiplica(int *x)
{
    *x = *x * *x;
}

int main()
{
    int t = 10;

    multiplica(&t);
    printf("%d\n", t);
    // saída: 100
}
```

- Para "enviar" um ponteiro: usa-se o operador & (endereço)
- Para "receber" e acessar o conteúdo apontado: usa-se o operador * (ponteiro/indireção)

Demonstração

- Execute o exemplo anterior (será necessário incluir o header *stdio.h*) em um dos ambientes abaixo:
  1. https://replit.com/languages/c
  2. https://techiedelight.com/compiler/
  3. https://www.mycompiler.io/new/c
  4. https://onecompiler.com/c
  5. https://www.tutorialspoint.com/online_c_compiler.php
  6. https://www.programiz.com/c-programming/online-compiler/
  7. https://www.onlinegdb.com/online_c_compiler
- Alternativamente, no Linux, utilize um editor de textos qualquer (como *geany*, *vim*, entre outros) para editar o código fonte e salvar o mesmo com a extensão .c
- E utilize o compilador *gcc*, para depois executar o programa:

```
$ gcc programa.c -o programa
$ ./programa
```

Comparando com Java

- Como isso funciona em Java?
  - Não existe o conceito de "passagem por referência"
  - Tudo é *passado por valor*

Exemplos

Retornando mais de um valor

- A passagem por referência também deve ser usada quando é necessário retornar mais de um valor em uma função
- Exemplo: função que troca o valor de duas variáveis entre si
- Observe que a função nesse caso é *void* (não retorna nada por valor)

```
void swap(int *var1, int *var2)
{
    int temp = *var1;
    *var1 = *var2;
    *var2 = temp;
}

int main()
{
    int x = 10, y = 5;
    swap(&x, &y);
    printf("x: %d, y: %d\n", x, y);
}
```

Retornando mais de um valor

- Outro exemplo: função que lê 10 valores inteiros e retorna o maior e menor dentre eles

```
void funcaoMaiorMenor(int *maior, int *menor)
{
    int num, i;
    for (i = 0; i < 10; i++) {
        printf("Digite um numero:");
        scanf("%d", &num);
        if (i == 0) *maior = *menor = num;
        else {
            if (num > *maior) *maior = num;
            else if(num < *menor) *menor = num;
        }
    }
}

int main()
{
    int maior, menor;
    funcaoMaiorMenor(&maior, &menor);
    printf("Maior: %d, menor: %d\n", maior, menor);
    return 0;
}
```

Passagem de vetores para funções

- Em C, vetores são *sempre* passados por *referência*
- Exemplo: função que recebe um vetor com 10 valores, e retorna o maior deles

```
int maior(int vetor[], int tam)
{
    int m = vetor[0];
    int i;

    for (i = 1; i < tam; i++)
        if (vetor[i] > m) m = vetor[i];

    return m;
}

int main()
{
    int numeros[10] = {5, 7, 80, 12, 30, 9, 4, 1, 10, 3};
    int valMaior = maior(numeros, 10);

    printf("Maior: %d\n", valMaior);
}
```

Passagem de vetores para funções

- Em C, vetores são *sempre* passados por *referência*
- Exemplo: função que recebe um vetor com 10 valores, e retorna o maior deles
- No exemplo anterior, a declaração da função *maior* pode ser escrita com a sintaxe de ponteiros, sem diferença alguma no comportamento

```
int maior(int *vetor, int tam)
```

- Como o que está sendo passado é uma *referência*, ao acessarmos uma posição do vetor estaremos acessando o mesmo por um nível de indireção

Referências

- Este material foi construído pelo Prof. Sérgio Johann Filho com base nos slides de aula do prof. Marcelo Cohen