Passagem de Parâmetros e Recursividade

Módulo 8 Aula 5

Linguagem C, o Curso Definitivo WR Kits

Autor: Dr. Eng. Wagner Rambo

Passagem de Parâmetros

Em C podemos passar parâmetros para funções por valor. Quando a passagem é realizada por valor, estamos enviando na realidade uma cópia do valor da variável para a função. Veja um exemplo no Box 1, onde temos uma função que retorna a multiplicação do dobro dos parâmetros passados para ela.

```
int mult_dobro(int a, int b);

main()
{
  int x, y;

  printf("Valor de x: ");
  scanf("%d",&x);
  printf("Valor de y: ");
  scanf("%d",&y);

  printf("(2x%d)x(2x%d) = %d\n",x,y,mult_dobro(x,y));

}

int mult_dobro(int a, int b)
{
  a*=2;
  b*=2;

  return a*b;

}
```

Box 1 - Exemplo de passagem de parâmetro por valor.

A função *mult_dobro* multiplica o dobro dos parâmetros passados para ela e retorna o resultado, quando passamos x e y, os seus valores são copiados para a e b. Poderíamos escrever toda a expressão em uma única linha, mas salientamos a multiplicação de a e b por 2 para ficar evidente que podemos atualizar os seus conteúdos dentro da função. Ao final do processo da função, a e b são destruídas. Repare que apesar de alterarmos a e b, as variáveis da função principal x e y mantiverem seus valores originais. Em outras palavras, uma função não irá alterar o conteúdo das variáveis passadas para ela, mas criar cópias para trabalhar os seus valores.

Utilizando ponteiros, podemos de certa forma "burlar" essa regra. Vamos supor que necessitamos de um procedimento para alterar o conteúdo de duas variáveis em nosso código principal. A ideia é que obter o quadrado da primeira e o cubo da segunda. Vejamos como fica este procedimento no Box 2.

```
void quad_cube(int *a, int *b)
{
  int temp_a = *a,
      temp_b = *b;

  *a = temp_a*temp_a;
  *b = temp_b*temp_b*temp_b;

} /* end quad_cube */
```

Box 2 - Função com passagem de cópia de endereço.

O Box 3 consiste em um programa para teste do procedimento *quad_cube*.

```
main()
{
  int x, y;

  printf("Valor de x: ");
  scanf("%d",&x);
  printf("Valor de y: ");
  scanf("%d",&y);

  printf("x=%d\n",x);
  printf("y=%d\n",y);

  quad_cube(&x,&y);

  printf("quadrado  de x=%d\n",x);
  printf("cubo      de y=%d\n",y);

} /* end main */
```

Box 3 - Teste do procedimento quad_cube.

Como pode ser visto, nós passamos uma cópia do endereço das variáveis x e y para o procedimento, de modo que consigamos alterar os seus conteúdos através dos ponteiros. Em algumas obras esse processo será chamado de passagem por referência. No entanto, ainda temos de certo modo uma passagem por valor, pois é enviada uma cópia do endereço das variáveis para a função.

Quando passamos vetores para funções, estamos realizando um método semelhante, uma vez que na realidade passamos o endereço do primeiro elemento do vetor. Utilizamos apenas o próprio nome do vetor na passagem pois consiste em uma abstração para o endereço do primeiro elemento, como vimos em aulas anteriores.

Passagem de Parâmetros na Linha de Comando

Uma das características do C é permitir a criação de programas que permitam a passagem de parâmetros através das linhas de comando do sistema operacional. A linha de comando enviada ao programa já deve ser reconhecida no instante em que a execução inicia, por este motivo, a função *main* deverá receber as linhas de comando. Os dois parâmetros que são passados para *main* podem ser conferidos no Box 4.

```
main(int argc, char *argv[])
```

Box 4 – Parâmetros passados para a função main.

O primeiro parâmetro, *argc*, é um inteiro que indica quantos argumentos foram passados na linha de comando, onde o nome do programa está incluso. E *argv* é um vetor que conterá as *strings* que poderão ser passadas na linha de comando. Para ilustrar esta aplicação, confira um programa exemplo no Box 5, que calcula o quadrado de todos os números passados na linha de comando e mostra o resultado em uma simples tabela.

```
main(int argc, char *argv[])
{
  int i,num, quad;              /* iterações, número, quadrado */

  printf("num quad\n");         /* cabeçalho da tabela */

  for(i=1; i<argc; i++)         /* inicia em 1, ignora o nome */
  {
    num = atoi(argv[i]);        /* converte string para inteiro */
    quad = num*num;

    printf("%2d %3d\n", num, quad);

  } /* end for */
}
```

Box 5 - Calcula o quadrado de todos os números passados na linha de comando.

Para testar este programa, cole o executável no disco C:\, renomeie para *quad* digita por exemplo a linha:

```
quad 2 7 12 18
```

O resultado será:

```
num quad
 2   4
 7  49
12 144
18 324
```

Exercício proposto: escreva um programa que você digite ohm seguido dos parâmetros de tensão e corrente, e seja calculado o valor da resistência, lembrando que

$$R = \frac{V}{I}$$

Exemplo: você digita

```
ohm  12  1  15  2  5  0.001
```

E o resultado será impresso no seguinte formato

```
Resistores: 12.0 | 7.5 | 5000.0 | Ohms.
```

Dica: utilize a função *atof()* para converter *string* para *double*.

Recursividade

Chamamos de recursividade, ou recursão, a capacidade que uma função tem de chamar a si própria. Na Linguagem C podemos tranquilamente trabalhar com funções recursivas. Na recursividade direta, a função chama a si própria. Na recursividade indireta, a função A chama a função B que volta a chama a função A. Um exemplo de função recursiva é a *downto0* apresentada no Box 6, que imprime na tela os números de n até 0.

```
void downto0(int n)
{
  if(n<0) return;

  printf("%d\n",n);
  downto0(n-1);

}
```

Box 6 - Função recursiva para imprimir números de n até 0.

Podemos trabalhar com funções recursivas na manipulação de *strings*, como por exemplo, a função apresentada no Box 7, que calcula o tamanho de uma *string*.

```
int strhowbig(const char *s)
{
  if(*s=='\0')
    return 0;
  else
    return 1 + strhowbig(s+1);

}
```

Box 7 - Função recursiva para calcular o tamanho da string.

Exercício proposto: projete uma função recursiva para imprimir os números de 1 a n, sendo n o número passado como parâmetro para a função.

Exemplo: o parâmetro passado é 5. A saída da função será 1, 2, 3, 4, 5.

Bibliografia: DAMAS, Luís; Linguagem C, décima edição.

Disponível em: https://amzn.to/3nGdIbN