Ponteiros e Funções

Vetores de ponteiros

Podemos construir vetores de ponteiros como declaramos vetores de qualquer outro tipo. Uma declaração de um vetor de ponteiros inteiros poderia ser:

```
int *pmatrx [10];
```

No caso acima, **pmatrx** é um vetor que armazena 10 ponteiros para inteiros.

Endereços de elementos de vetores

Nesta seção vamos apenas ressaltar que a notação

&nome_da_variável[índice]

é válida e retorna o endereço do ponto do vetor indexado por índice. Isto seria equivalente a nome_da_variável + indice. É interessante notar que, como consequência, o ponteiro **nome_da_variável** tem o endereço **&nome_da_variável[0]**, que indica onde na memória está guardado o valor do primeiro elemento do vetor.

A Função

Funções são as estruturas que permitem ao usuário separar seus programas em blocos. Se não as tivéssemos, os programas teriam que ser curtos e de pequena complexidade. Para fazermos programas grandes e complexos temos de construí-los bloco a bloco.

Uma função no C tem a seguinte forma geral:

tipo_de_retorno nome_da_função (declaração_de_parâmetros)
{
corpo_da_função
}

O tipo-de-retorno é o tipo de variável que a função vai retornar. O default é o tipo **int**, ou seja, uma função para qual não declaramos o tipo de retorno é considerada como retornando um inteiro. A declaração de parâmetros é uma lista com a seguinte forma geral:

tipo nome1, tipo nome2, ... , tipo nomeN

Repare que o tipo deve ser especificado para cada uma das N variáveis de entrada. É na declaração de parâmetros que informamos ao compilador quais serão as entradas da função (assim como informamos a saída no tipo-de-retorno).

O corpo da função é a sua alma. É nele que as entradas são processadas, saídas são geradas ou outras coisas são feitas.

O Comando return

O comando **return** tem a seguinte forma geral:

return valor_de_retorno; ou *return;*

Digamos que uma função está sendo executada. Quando se chega a uma declaração **return** a função é encerrada imediatamente e, se o valor de retorno é informado, a função retorna este valor. É importante lembrar que o valor de retorno fornecido tem que ser compatível com o tipo de retorno declarado para a função.

Uma função pode ter mais de uma declaração **return**. Isto se torna claro quando pensamos que a função é terminada quando o programa chega à primeira declaração **return**. Abaixo estão dois exemplos de uso do **return**:

```
#include <stdio.h>

int Square (int a)

{

        return (a*a);

}

int main ()

{

        int num;

        printf ("Entre com um numero: ");
```

```
        scanf ("%d",&num);

        num=Square(num);

        printf ("\n\nO seu quadrado vale: %d\n",num);

        return 0;

}
```

Vetores como Argumentos de Funções

Quando vamos passar um vetor como argumento de uma função, podemos declarar a função de três maneiras equivalentes. Seja o vetor:

```
int matrx [50];
```

e que queiramos passá-la como argumento de uma função **func()**. Podemos declarar **func()** das três maneiras seguintes:

```
void func (int matrx[50]);

void func (int matrx[]);

void func (int *matrx);
```

Nos três casos, teremos dentro de **func()** um **int*** chamado **matrx**. Ao passarmos um vetor para uma função, na realidade estamos passando um ponteiro. Neste ponteiro é armazenado o endereço do primeiro elemento do vetor. Isto significa que não é feita uma cópia, elemento a elemento do vetor. Isto faz com que possamos alterar o valor dos elementos do vetor dentro da função.

Passagem de parâmetros por valor e passagem por referência

Já vimos que, na linguagem C, quando chamamos uma função os parâmetros formais da função copiam os valores dos parâmetros que são passados para a função. Isto quer dizer que não são alterados os valores que os parâmetros têm fora da função. Este tipo de chamada de função é denominado chamada por valor. Isto ocorre porque são passados para a função apenas os valores dos parâmetros e não os próprios parâmetros. Veja o exemplo abaixo:

```
#include <stdio.h>

float sqr (float num);

void main ()

{

        float num,sq;

        printf ("Entre com um numero: ");

        scanf ("%f",&num);

        sq=sqr(num);

        printf ("\n\nO numero original e: %f\n",num);

        printf ("O seu quadrado vale: %f\n",sq);

}



float sqr (float num)

{

        num=num*num;

        return num;

}
```

No exemplo acima o parâmetro formal **num** da função **sqr()** sofre alterações dentro da função, mas a variável **num** da função **main()** permanece inalterada: é uma chamada por valor.

Outro tipo de passagem de parâmetros para uma função ocorre quando alterações nos parâmetros formais, dentro da função, alteram os valores dos parâmetros que foram passados para a função. Este tipo de chamada de função tem o nome de "chamada por referência". Este nome vem do fato de que, neste tipo de chamada, não se passa para a função os valores das variáveis, mas sim suas referências (a função usa as referências para alterar os valores das variáveis fora da função).

O C só faz chamadas por valor. Isto é bom quando queremos usar os parâmetros formais à vontade dentro da função, sem termos que nos preocupar em estar alterando os valores dos parâmetros que foram passados para a função. Mas isto também pode ser ruim às vezes, porque podemos querer mudar os valores dos parâmetros fora da função também. O C++ tem um recurso que permite ao programador fazer chamadas por referência. Há entretanto, no C, um recurso de programação que podemos usar para simular uma chamada por referência.

Quando queremos alterar as variáveis que são passadas para uma função, nós podemos declarar seus parâmetros formais como sendo *ponteiros*. Os ponteiros são a "referência" que precisamos para poder alterar a variável fora da função. O único inconveniente é que, quando usarmos a função, teremos de lembrar de colocar um **&** na frente das variáveis que estivermos passando para a função.

printf

A função **printf()** tem a seguinte forma geral:

printf (string_de_controle,lista_de_argumentos);

Teremos, na string de controle, uma descrição de tudo que a função vai colocar na tela. A string de controle mostra não apenas os caracteres que devem ser colocados na tela, mas também quais as variáveis e suas respectivas posições. Isto é feito usando-se os códigos de controle, que usam a notação **%**. Na string de controle indicamos quais, de qual tipo e em que posição estão as variáveis a serem apresentadas. É muito importante que, para cada código de controle, tenhamos um argumento na lista de argumentos. Apresentamos agora alguns dos códigos **%**:

Código	Significado
%d	Inteiro
%f	Float
%c	Caractere
%s	String
%%	Coloca na tela um %

Vamos ver alguns exemplos de **printf()** e o que eles exibem:

```
printf ("Teste %% %%") -> "Teste % %"
```

```
printf ("%f",40.345) -> "40.345"
```

```
printf ("Um caractere %c e um inteiro %d",'D',120) -> "Um caractere D e
um inteiro 120"
```

```
printf ("%s e um exemplo","Este") -> "Este e um exemplo"
```

```
printf ("%s%d%%","Juros de ",10) -> "Juros de 10%"
```

Maiores detalhes sobre a função **printf()** (incluindo outros códigos de controle) serão vistos posteriormente, mas podem ser consultados de antemão pelos interessados.

scanf

O formato geral da função **scanf()** é:

scanf (string-de-controle,lista-de-argumentos);

Usando a função **scanf()** podemos pedir dados ao usuário. Um exemplo de uso, pode ser visto acima. Mais uma vez, devemos ficar atentos a fim de colocar o mesmo número de argumentos que o de códigos de controle na string de controle. Outra coisa importante é lembrarmos de colocar o **&** antes das variáveis da lista de argumentos. É impossível justificar isto agora, mas veremos depois a razão para este procedimento. Maiores detalhes sobre a função **scanf()** serão vistos posteriormente, mas podem ser consultados de antemão pelos interessados.