Ponteiros para funções

Variáveis são espaços reservados da memória utilizados para armazenar dados de nossos programas. Já um programa é, na verdade, um conjunto de instruções armazenadas na memória, juntamente com seus dados.

Uma função nada mais é do que um bloco de código (ou seja, declarações e outros comandos) que pode ser nomeado e chamado de dentro de um programa.

Uma função, também é um conjunto de instruções armazenada na memória. Portanto, podemos acessar uma função por meio de um ponteiro que aponte para o local (endereço) onde a função está na memória.

A principal vantagem de se declarar um ponteiro para uma função, é a construção de códigos genéricos. Pense na ordenação de números. Podemos definir um algoritmo que ordene números inteiros e querer utilizar essa implementação para ordenar outros tipos de dados, como por exemplo *strings*. Em vez de reescrever toda a função de ordenação, podemos passar para essa função o ponteiro da função de comparação que desejamos utilizar para cada tipo de dado. Ponteiros permitem realizar uma chamada indireta à função e passá-la como parâmetro para outras funções. Isso é muito útil na implementação de algoritmos genéricos em C.

Declarando um ponteiro para uma função

Em linguagem C, a declaração de um ponteiro para uma função segue a seguinte forma geral:

```
tipo_retornado (*nome_do_ponteiro)(lista_de_tipos);
```

O nome_do_ponteiro deve ser sempre colocado entre parênteses juntamente com o asterisco: (*nome_do_ponteiro). Isso é necessário para evitar confusões com a declaração de funções que retornem ponteiros, por exemplo:

```
tipo_retornado *nome_do_ponteiro(lista_de_tipos);
```

que é uma função que retorna um ponteiro do **tipo_retornado**, enquanto

```
tipo_retornado (*nome_do_ponteiro)(lista_de_tipos);
```

é um ponteiro para funções que retornam **tipo_retornado**.

Nota: um ponteiro para funções só pode apontar para funções que possuam o mesmo protótipo.

Temos agora que **nome_do_ponteiro** é um ponteiro para funções. Mas não para qualquer função. Ele é um ponteiro apenas para funções que possuam o mesmo protótipo que foi definido para o ponteiro. Dessa forma declaramos um ponteiro para funções como sendo:

```
int (*p)(int, int);
```

Onde **p** é um ponteiro para uma função qualquer (definida posteriormente), que receba dois inteiros como parâmetro.

O ponteiro **p** poderá ser apontado para qualquer função que receba dois parâmetros inteiros (independentemente de seu nome e retorne um valor inteiro, como por exemplo:

```
int soma(int x, int y);
```

Apontando um ponteiro para uma função

Como qualquer outro ponteiro, quando um ponteiro de função é declarado, ele não possui um endereço associado. Qualquer tentativa de uso desse ponteiro causa comportamento indefinido no programa. O ponteiro para função também pode ser inicializado com a constante NULL, o que indica que aquele ponteiro aponta para uma região de memória inexistente, ou seja, nenhuma função.

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int max(int a, int b);
5
6   int main(){
7       int x, y, z;
8       int (*p)(int, int) = NULL;
9       printf("Digite dois numeros: ");
10      scanf(" %d %d", &x, &y);
11      //ponteiro recebe o endereço da função
12      p = max;
13      z = p(x, y);
14      printf("O maior e: %d\n", z);
15      getchar();
16      return 0;
17  }
18
19  int max(int a, int b){
20      return (a > b) ? a : b;
21  }
```

Lembre-se: Quando criamos uma função, o computador a guarda em um espaço reservado de memória. Ao nome que damos a essa função o computador associa o endereço do espaço que reservou na memória para guardá-la. Assim basta atribuir o nome da função ao ponteiro para que ele passe a apontar para a função na memória, como feito na linha 12.

Assim para utilizar a função apontada por um ponteiro, basta utilizar o nome do ponteiro como se fosse o nome da função, como foi feito na linha 13 do programa anterior. Nela utilizamos o ponteiro p como se fosse um outro nome, ou um sinônimo para a função ***max()***.

Um outro uso possível para ponteiros para funções:

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

int soma(int a, int b);
int subtracao(int a, int b);
int multiplicacao(int a, int b);
int divisao(int a, int b);

int main(){
    int x, y;
    char ch;
    int (*p)(int, int) = NULL;
    printf("Digite uma operacao matematica (+, -, *, /): ");
    ch = getchar();
    printf("Digite dois números: ");
    scanf(" %d %d", &x, &y);
    switch(ch){
    case '+':
        p = soma; //recebe end de soma
        break;
    case '-':
        p = subtracao; //recebe end de subtracao
        break;
    case '*':
        p = multiplicacao; //recebe end de multiplicacao
        break;
    case '/':
        p = divisao; //recebe end de divisao
        break;
    default:
        p = NULL;
    }
    if(p != NULL){
        printf("Resultado: %d\n", p(x, y));
    }else{
        printf("Operacao invalida\n");
    }
    system("pause");
    return 0;
}
int soma(int a, int b){
    return a + b;
}
int subtracao(int a, int b){
    return a - b;
}
int multiplicacao(int a, int b){
    return a * b;
}
int divisao(int a, int b){
    return a / b;
}
```

Passando um ponteiro para uma função como parâmetro de outra função

Como visto anteriormente, a principal vantagem de se declarar um ponteiro para uma função, é permitir a construção de códigos genéricos. Isso ocorre porque esses ponteiros permitem fazer uma chamada indireta à função, de modo que possam ser passados com parâmetro para outras funções. Relembrando como é a declaração de um ponteiro para uma função:

```
tipo_retornado (*nome_do_ponteiro)(lista_de_tipos);
```

Se quisermos declarar uma função que possa receber um ponteiro para função como parâmetro, tudo o que precisamos fazer é incorporar a declaração de um ponteiro para uma função dentro da declaração dos parâmetros da função. Considere o seguinte ponteiro para função:

```
int (*p)(int, int);
```

Se quisermos passar esse ponteiro para uma outra função, devemos declarar esse ponteiro na sua lista de parâmetros:

```
int executa(int (*p)(int, int), int x, int y);
```

Temos agora que a função ***executa()*** recebe três parâmetros:

- **p**: um ponteiro para uma função que recebe dois parâmetros inteiros e retorne um valor inteiro;
- **x**: um valor inteiro;
- **y**: outro valor inteiro.

Abaixo um exemplo de utilização de um ponteiro para função sendo passado como parâmetro para outra função:

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


float soma(float a, float b);
float subtracao(float a, float b);
float multiplicacao(float a, float b);
float divisao(float a, float b);
float executa(float (*p)(float, float), float x, float y);

int main(){
    float x, y;
    char ch;
    float (*p)(float, float) = NULL;
    printf("Digite uma operacao matematica (+, -, *, /): ");
    ch = getchar();
    printf("Digite dois números: ");
    scanf(" %f %f", &x, &y);
```

```c
    switch(ch){
    case '+':
        p = soma; //recebe end de soma
        break;
    case '-':
        p = subtracao; //recebe end de subtracao
        break;
    case '*':
        p = multiplicacao; //recebe end de multiplicacao
        break;
    case '/':
        p = divisao; //recebe end de divisao
        break;
    default:
        p = NULL;
    }
    if(p != NULL){
        printf("Resultado: %6.2f\n", executa(p, x, y));
    }else{
        printf("Operacao invalida\n");
    }
    system("pause");
    return 0;
}

float soma(float a, float b){
    return a + b;
}
float subtracao(float a, float b){
    return a - b;
}
float multiplicacao(float a, float b){
    return a * b;
}
float divisao(float a, float b){
    return a / b;
}
float executa(float (*p)(float, float), float x, float y){
    return p(x, y);
}
```

Criando um *array* de ponteiros para função

Relembrando a forma de declaração de *arrays*. Para declarar uma variável, a forma geral é:

```
tipo nome;
```

Para declarar um *array*, basta indicar, entre colchetes, o tamanho do *array* que queremos criar:

```
tipo nome[tamanho];
```

Para declarar um *array* de ponteiros para funções, o principio é o mesmo utilizado na declaração de *arrays* dos tipos básicos: basta indicar na declaração o seu tamanho entre colchetes para transformar essa declaração na declaração de um *array*.

A declaração de *arrays* de ponteiros para funções funciona exatamente da mesma maneira que a declaração para outros tipos, ou seja, basta indicar na declaração do ponteiro para função o seu tamanho entre colchetes:

```
//ponteiro para função
tipo_retornado (*nome_do_ponteiro)(lista_de_tipos);
//array de ponteiros para função com "tamanho" elementos
tipo_retornado (*nome_do_ponteiro[tamanho])(lista_de_tipos);
```

Feito isso, cada posição do *array* pode agora apontar para uma função diferente, como no programa exemplo a seguir:

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

int soma(int a, int b);
int subtracao(int a, int b);
int multiplicacao(int a, int b);
int divisao(int a, int b);

int main(){
    int x, y;
    int indice = -1;
    int (*p[4])(int, int);
    p[0] = soma;
    p[1] = subtracao;
    p[2] = multiplicacao;
    p[3] = divisao;
    char ch;
    printf("Digite uma operacao matematica (+, -, *, /): ");
    ch = getchar();
    printf("Digite dois números: ");
    scanf(" %f %f", &x, &y);
    switch(ch){
    case '+':
        indice = 0;
        break;
    case '-':
        indice = 1;
        break;
    case '*':
        indice = 2;
        break;
    case '/':
        indice = 3;
        break;
    default:
        indice = 1;
    }
```

```
38      if(indice >= 0){
39          printf("Resultado: %6.2f\n", p[indice](x, y));
40      }else{
41          printf("Operacao invalida\n");
42      }
43      system("pause");
44      return 0;
45  }
46
47  int soma(int a, int b){
48      return a + b;
49  }
50  int subtracao(int a, int b){
51      return a - b;
52  }
53  int multiplicacao(int a, int b){
54      return a * b;
55  }
56  int divisao(int a, int b){
57      return a / b;
58  }
```