12.6 Apontadores para funções

O código correspondente a uma função está armazenado em uma posição de memória.

Portanto é possível ter um um apontador para uma função, ou seja para a região de memória correspondente a uma dada função.

Exemplo 1:

```
// func é uma função que tem como parâmetro um inteiro

// declarando func
void func (int );

//definindo func
void func (int x)
{
...

}


// definindo um ponteiro func_ptr para uma função como func
// func_ptr é um ponteiro para função void que recebe um int

void (*func_ptr)(int);

/* fazendo func_ptr  apontar para func
   O uso do & antes do nome da função é opcional, geralmente só se usa o nome dela mesmo.  */

func_ptr = &func;

// chamando func, através do ponteiro com o parâmetro 42
func_ptr(42);
```

Sintaxe para declarar um apontador para uma função

<tipo_retorno_função> (*<nome_do_apontador>) (<parâmetros_da função>)

Os parênteses são importantes para indicar que o * está ligado à variável nome_do_apontador. Sem eles seria declarada uma função que retorna um apontador para o tipo que a função retorna. Veja abaixo

// apontador fun_ptr para função que recebe char* e retorna int
 	
int (*fun_ptr)(char *);
	// declaração de função que recebe char* e retorna int*
	
	int *fun_ptr(char *);

Exemplo 2 com typedef:

```
// FuncPtr é um apontador para funções que recebem e retornam char*
typedef char *(*FuncPtr)(char *);

// declara um ponteiro e faz ele apontar para a função gets
FunPtr ptr = gets;
```

Exemplo 3: o uso de apontadores permite fazer com que uma função possa variar. Geralmente são usados para fazer uma parte de função de diferente formas. A maneira como vai ser feita esta parte pode ser decidida em tempo de execução, mediante uma opção do usuário.

```
#include <stdio.h>

typedef int (*Operacao) (int, int);   // define o tipo de apontadores
int soma(int, int ) ;
int multiplica(int a, int b) ;
int opera(int, int, Operacao)

int soma(int a, int b)
{
   return a + b;
}

int multiplica(int a, int b)
{
   return a * b;

}
t

/* define uma operação binária (genérica) sobre dois operandos.
 A operação a ser executada depende do parâmetro op que é um
apontador  para este tipo de operação */

int opera(int a, int b, Operacao op)
{
  return op(a,b);
}

// uso 1
int main()
{
  int o, x, y;
  printf(" Entre com dois numeros\n");
  scanf("%d %d", &x, &y);

  printf(" Digite 1 para somar e 2 para multiplicar\n");
  scanf("%d",  &o);
```

```
  if (o==1)
    printf("%d\n", opera(x, y, soma));
  eles
   printf("%d\n", opera(x, y, multiplica));

}

// uso 2
int main()
{
  int o,x, y;
  Operacao p_op;

  printf(" Entre com dois numeros\n");
  scanf("%d %d", &x, &y);

  printf(" Digite 1 para somar e 2 para multiplicar\n");
  scanf("%d",  &o);
  p_op = (o==1)? soma : multiplica;
  printf("%d\n", opera(x, y, p_op));

}
```

Exemplo 4: Utilização comum dos apontadores para funções é a implementação de menus. O usuário escolhe uma opção de um menu indicando, por exemplo, um número inteiro. Para cada opção haverá uma função diferente a ser chamada.

Apontadores para as diferentes funções são guardados num vetor de apontadores para as funções. O número indicado será o índice para escolher uma das posições do vetor de apontadores que corresponde à função escolhida.

```
#include <stdio.h>
void f0(int);
void f1(int);
void f2(int);
typedef void (*PtrFunVoidInt)(int);
```

```
int main(void)
{
  PtrFunVoidInt f[3] = {f0, f1, f2};
  int opcao;
  printf("Indicar numero entre 0 e 2; outro valor termina: ");
  scanf("%d", &opcao);
  while (opcao >= 0 && opcao <3) {
    (*f[opcao])(opcao);
    printf("Indicar numero entre 0 e 2; outro valor termina: ");
    scanf("%d", &opcao);
    }
  printf("O programa vai terminar\n");
  return 0;
}

void f0(int a)
{
  printf("Valor indicado foi %d e por\n", a);
  printf("isso foi chamada f0\n");
}

void f1(int a)
{
  printf("Valor indicado foi %d e por\n", a);
  printf("isso foi chamada f1\n");
}

void f2(int a)
{
  printf("Valor indicado foi %d e por\n", a);
  printf("isso foi chamada f3\n");
}
```

13.5 Práticas com Apontadores

Não desreferenciar um apontador que não tenha sido inicializado pode causar erros fatais.

Não utilizar aritmética de apontadores sobre um apontador que não aponte para um array. Não subtrair ou comparar apontadores que não apontem para o mesmo array

Não ultrapassar os limites de um array quando se manipula um apontador

Embora os nomes dos arrays sejam apontadores para o início desses arrays, não poderão ser manipulados pela aritmética dos apontadores, por serem constantes

Incluir Ptr no nome das variÃ¡veis apontador
 Ex: xPtr, intPtr, funPtr

Utilizar a passagem de um argumento por valor quando não for desejado alterar esse argumento. Trata-se de um exemplo de aplicação do conceito do privilégio mínimo.

Utilizar a notação de array em vez da notação de apontador para manipular arrays. Apenas torna a compilação ligeiramente mais demorada, mas a leitura do programa torna-se muito mais clara

Passar grandes estruturas de dados usando apontadores para dados qualificados com const, para obter o desempenho da passagem por referência e a segurança da passagem por valor (O ANSI C prevê const, outros não).

Se um argumento não irá ser alterado no corpo de uma função, então deverá ser declarado com const (mesmo que passado por valor!!!)... isto permite identificar eventuais tentativas incorretas de alteração.