AULA 49

INTRODUÇÃO AOS APONTADORES

Apontadores ou ponteiros são certamente uma das características mais destacáveis da linguagem de programação C. Os apontadores agregam poder e flexibilidade à linguagem de maneira a diferenciá-la de outras linguagens de programação de alto nível, permitindo a representação de estruturas de dados complexas, modificação de valores passados como argumentos a funções, alocação dinâmica de espaços na memória, entre outros destaques. Nesta aula iniciaremos o contato com esses elementos.

49.1 Variáveis apontadoras

Para bem compreender os apontadores precisamos inicialmente compreender a idéia de indireção. Conceitualmente, um apontador permite acesso indireto a um valor armazenado em algum ponto da memória. Esse acesso é realizado justamente porque um **apontador** é uma variável que armazena um valor especial, que é um endereço de memória, e por isso nos permite acessar indiretamente o valor armazenado nesse endereço.

A memória de um computador pode ser vista como constituída de muitas posições (ou compartimentos ou células), dispostas continuamente, cada qual podendo armazenar um valor, na base binária. Ou seja, a memória nada mais é do que um grande vetor que pode armazenar valores na base binária e que, por sua vez, esse valores podem ser interpretados como valores de diversos tipos. Os índices desse vetor, numerados seqüencialmente a partir de 0 (zero), são chamados de **endereços de memória**. Quando escrevemos um programa em uma linguagem de programação de alto nível, o nome ou identificador de uma variável é associado a um índice desse vetor, isto é, a um endereço da memória. A tradução do nome da variável para um endereço de memória, e vice-versa, é feita de forma automática e transparente pelo compilador da linguagem de programação.

Em geral, a memória de um computador é dividida em bytes, com cada byte sendo capaz de armazenar 8 bits de informação. Cada byte tem um único endereço que o distingue de outros bytes da memória. Se existem n bytes na memória, podemos pensar nos endereços como números de intervalo de 0 a $n - 1$, como mostra a figura 49.1.

Um programa executável é constituído por trechos de código e dados, ou seja, por instruções de máquina que correspondem às sentenças no programa original na linguagem C e de variáveis no programa original. Cada variável do programa ocupa um ou mais bytes na memória. O endereço do primeiro byte de uma variável é dito ser o endereço da variável. Na figura 49.2, a variável `i` ocupa os bytes dos endereços 2000 e 2001. Logo, o endereço da variável `i` é 2000.

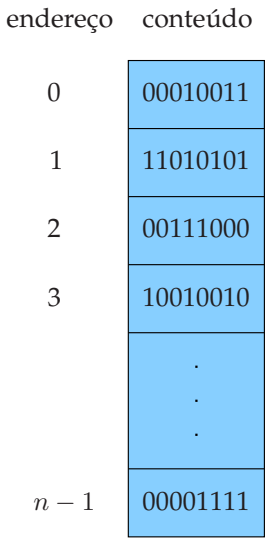

Figura 49.1: Uma ilustração da memória e de seus endereços.

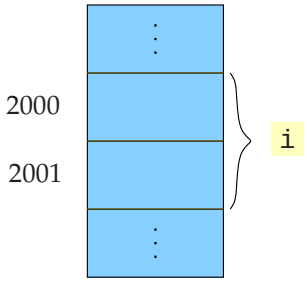

Figura 49.2: Endereço da variável `i`.

Os apontadores têm um papel importante em toda essa história. Apesar de os endereços serem representados por números, como ilustrado nas figuras 49.1 e 49.2, o intervalo de valores que esse objetos podem assumir é diferente do intervalo que os números inteiros, por exemplo, podem assumir. Isso significa, entre outras coisas, que não podemos armazenar endereços em variáveis do tipo inteiro. Endereços são armazenados em variáveis especiais, chamadas de variáveis apontadoras.

Quando armazenamos o endereço de uma variável `i` em uma variável apontadora `p`, dizemos que `p` **aponta para** `i`. Em outras palavras, um apontador nada mais é que um endereço e uma variável apontadora é uma variável que pode armazenar endereços.

Ao invés de mostrar endereços como números, usaremos uma notação simplificada de tal forma que, para indicar que uma variável apontadora `p` armazena o endereço de uma variável `i`, mostraremos o conteúdo de `p` – um endereço – como uma flecha orientada na direção de `i`, como mostra a figura 49.3.

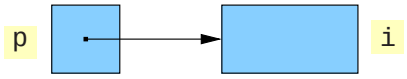

Figura 49.3: Representação de uma variável apontadora.

Uma variável apontadora pode ser declarada da mesma forma que uma variável qualquer, mas com um asterisco precedendo seu identificador. Por exemplo,

```
int *p;
```

Essa declaração indica que `p` é uma variável apontadora capaz de apontar para objetos do tipo `int`. A linguagem C obriga que toda variável apontadora aponte apenas para objetos de um tipo particular, chamado de **tipo referenciado**.

Diante do exposto até este ponto, daqui por diante não mais distinguiremos os termos apontador e variável apontadora, ficando então subentendido valor (conteúdo) e variável.

49.2 Operadores de endereçamento e de indireção

A linguagem C possui dois operadores para uso específico com apontadores. Para obter o endereço de uma variável, usamos o **operador de endereçamento (ou de endereço)** `&`. Se `v` é uma variável, então `&v` é seu endereço na memória. Para ter acesso ao objeto que um apontador aponta, temos de usar o **operador de indireção** `*`. Se `p` é um apontador, então `*p` representa o objeto para o qual `p` aponta no momento.

Declarar uma variável apontadora reserva um espaço na memória para um apontador mas não a faz apontar para um objeto. É crucial inicializar `p` antes de usá-lo. Uma forma de inicializar um apontador é atribuir-lhe o endereço de alguma variável usando o operador `&`:

```
int i, *p;
p = &i;
```

Atribuir o endereço da variável `i` para a variável `p` faz com que `p` aponte para `i`, como ilustra-a figura 49.4.

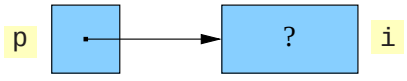

Figura 49.4: Variável apontadora `p` contendo o endereço da variável `i`.

É possível inicializar uma variável apontadora no momento de sua declaração, como abaixo:

```
int i, *p = &i;
```

Uma vez que uma variável apontadora aponta para um objeto, podemos usar o operador de indireção `*` para acessar o valor armazenado no objeto. Se `p` aponta para `i`, por exemplo, podemos imprimir o valor de `i` como segue:

```
printf("%d\n", *p);
```

Observe que a função `printf` mostrará o valor de `i` e não o seu endereço. Observe também que aplicar o operador `&` a uma variável produz um apontador para a variável e aplicar o operador `*` para um apontador retoma o valor original da variável:

```
j = *&i;
```

Na verdade, a atribuição acima é idêntica à seguinte:

```
j = i;
```

Enquanto dizemos que `p` aponta para `i`, dizemos também que `*p` é um apelido para `i`. Não apenas `*p` tem o mesmo valor que `i`, mas alterar o valor de `*p` altera o valor de `i`.

Uma observação importante que auxilia a escrever e ler programas com apontadores é sempre "traduzir" os operadores unários de endereço & e indireção * para *endereço da variável* e *conteúdo da variável apontada por*, respectivamente. Sempre que usamos um desses operadores no sentido de estabelecer indireção e apontar para valores, é importante traduzi-los desta forma.

Programa 49.1: Um exemplo do uso de apontadores.

```
1  #include <stdio.h>
2  int main(void)
3  {
4    char c, *p;
5    p = &c;
6    c = 'a';
7    printf("c=%c, *p=%c\n", c, *p);
8    c = '/';
9    printf("c=%c, *p=%c\n", c, *p);
10   *p = 'Z';
11   printf("c=%c, *p=%c\n", c, *p);
12   return 0;
13 }
```

Observe que no programa 49.1, após a declaração das variáveis `c` e `p`, temos a inicialização do apontador `p`, que recebe o endereço da variável `c`, sendo essas duas variáveis do mesmo tipo `char`. É importante sempre destacar que o valor, ou conteúdo, de um apontador na linguagem C não tem significado até que contenha, ou aponte, para algum endereço válido.

A primeira chamada da função `printf` no programa 49.1 apenas mostra o conteúdo da variável `c`, que foi inicializada com o caractere `'a'`, e também o conteúdo da variável apontada por `p`. Como a variável `p` aponta para a variável `c`, o valor apresentado na saída é também aquele armazenado na variável `c`, isto é, o caractere `'a'`. A segunda chamada da função `printf` é precedida pela alteração do conteúdo da variável `c` e, como a variável `p` mantém-se apontando para a variável `c`, a chamada da função `printf` faz com que o caractere `'/'` seja apresentado na saída. É importante notar que, a menos que o conteúdo da variável `p` seja modificado, a expressão `*p` sempre acessa o conteúdo da variável `c`. Por fim, a última chamada à função `printf` é precedida de uma atribuição que modifica o conteúdo da variável apontada por `p`, isto é, a atribuição a seguir:

```
*p = 'Z';
```

faz também com que a variável `c` receba o caractere `'Z'`.

A linguagem C permite ainda que o operador de atribuição copie apontadores, supondo que possuam o mesmo tipo. Suponha que a seguinte declaração tenha sido feita:

```
int i, j, *p, *q;
```

A sentença

```
p = &i;
```

é um exemplo de atribuição de um apontador, onde o endereço de `i` é copiado em `p`. Um outro exemplo de atribuição de apontador é dado a seguir:

```
q = p;
```

Essa sentença copia o conteúdo de `p`, o endereço de `i`, para `q`, fazendo com que `q` aponte para o mesmo lugar que `p` aponta, como podemos visualizar na figura 49.5.

Ambos os apontadores `p` e `q` apontam para `i` e, assim, podemos modificar o conteúdo de `i` indiretamente através da atribuição de valores para `*p` e `*q`.

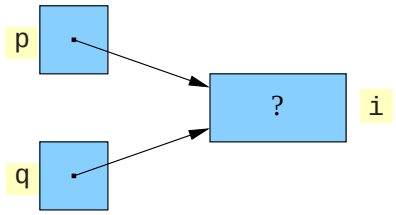

Figura 49.5: Dois apontadores para um mesmo endereço.

49.3 Apontadores em expressões

Apontadores podem ser usados em expressões aritméticas de mesmo tipo que seus tipos referenciados. Por exemplo, um apontador para uma variável do tipo inteiro pode ser usado em uma expressão aritmética envolvendo números do tipo inteiro, supondo o uso correto do operador de indireção `*`.

É importante observar também que os operadores `&` e `*`, por serem operadores unários, têm precedência sobre os operadores binários das expressões aritméticas em que se envolvem. Por exemplo, em uma expressão aritmética envolvendo números do tipo inteiro, os operadores binários `+`, `-`, `*` e `/` têm menor prioridade que o operador unário de indireção `*`.

Vejamos a seguir, como um exemplo simples, o programa 49.2 que usa um apontador como operando em uma expressão aritmética.

Programa 49.2: Outro exemplo do uso de apontadores.

```
1  #include <stdio.h>
2  int main(void)
3  {
4    int i, j, *apt1, *apt2;
5    apt1 = &i;
6    i = 5;
7    j = 2 * *apt1 + 3;
8    apt2 = apt1;
9    printf("i=%d, j=%d, *apt1=%d, *apt2=%d\n", i, j, *apt1, *apt2);
10   return 0;
11 }
```

Exercícios

49.1 Se `i` é uma variável e `p` é uma variável apontadora que aponta para `i`, quais das seguintes expressões são apelidos para `i`?

(a) `*p`

(b) `&p`

(c) `*&p`
(d) `&*p`
(e) `*i`
(f) `&i`
(g) `*&i`
(h) `&*i`

49.2 Se `i` é uma variável do tipo `int` e `p` e `q` são apontadores para `int`, quais das seguintes atribuições são corretas?

(a) `p = i;`
(b) `*p = &i;`
(c) `&p = q;`
(d) `p = &q;`
(e) `p = *&q;`
(f) `p = q;`
(g) `p = *q;`
(h) `*p = q;`
(i) `*p = *q;`

49.3 Entenda o que o programa 49.3 faz, simulando sua execução passo a passo. Depois disso, implemente-o.

Programa 49.3: Programa do exercício 49.3.

```
1  #include <stdio.h>
2  int main(void)
3  {
4    int a, b, *apt1, *apt2;
5    apt1 = &a;
6    apt2 = &b;
7    a = 1;
8    (*apt1)++;
9    b = a + *apt1;
10   *apt2 = *apt1 * *apt2;
11   printf("a=%d, b=%d, *apt1=%d, *apt2=%d\n", a, b, *apt1, *apt2);
12   return 0;
13 }
```

49.4 Entenda o que o programa 49.4 faz, simulando sua execução passo a passo. Depois disso, implemente-o.

49.5 Entenda o que o programa 49.5 faz, simulando sua execução passo a passo. Depois disso, implemente-o.

Programa 49.4: Programa do exercício 49.4.

```
#include <stdio.h>
int main(void)
{
  int a, b, c, *apt;
  a = 3;
  b = 7;
  printf("a=%d, b=%d\n", a, b);
  apt = &a;
  c = *apt;
  apt = &b;
  a = *apt;
  apt = &c;
  b = *apt;
  printf("a=%d, b=%d\n", a, b);
  return 0;
}
```

Programa 49.5: Programa do exercício 49.5.

```
#include <stdio.h>
int main(void)
{
  int i, j, *p, *q;
  p = &i;
  q = p;
  *p = 1;
  printf("i=%d, *p=%d, *q=%d\n", i, *p, *q);
  q = &j;
  i = 6;
  *q = *p;
  printf("i=%d, j=%d, *p=%d, *q=%d\n", i, j, *p, *q);
  return 0;
}
```