

APONTADORES E VETORES

Nas aulas 49 e 50 aprendemos o que são os apontadores e também como são usados como parâmetros de funções e devolvidos de funções. Nesta aula veremos outra aplicação para os apontadores. A linguagem C nos permite usar expressões aritméticas de adição e subtração com apontadores que apontam para elementos de vetores. Essa é uma forma alternativa de trabalhar com vetores e seus índices. Para nos tornarmos melhores programadores da linguagem C é necessário conhecer bem essa relação íntima entre apontadores e vetores. Além disso, o uso de apontadores para trabalhar com vetores é vantajoso em termos de eficiência do programa executável resultante.

51.1 Aritmética com apontadores

Nas aulas 49 e 50 vimos que apontadores podem apontar para elementos de um vetor. Suponha, por exemplo, que temos declaradas as seguintes variáveis:

```
int v[10], *p;
```

Podemos fazer o apontador `p` apontar para o elemento `v[0]` do vetor fazendo a seguinte atribuição, como mostra a figura 51.1:

```
p = &v[0];
```

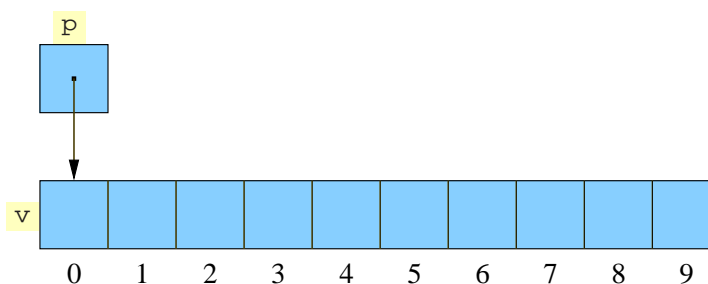


Figura 51.1: `p = &v[0];`

Podemos acessar `v[0]` através de `p`, como ilustrado na figura 51.2.

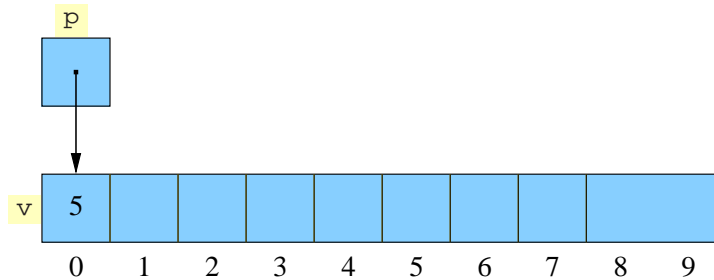


Figura 51.2: `*p = 5;`

Podemos ainda executar **aritmética com apontadores** ou **aritmética com endereços** sobre `p` e assim acessamos outros elementos do vetor `v`. A linguagem C possibilita três formas de aritmética com apontadores: (i) adicionar um número inteiro a um apontador; (ii) subtrair um número inteiro de um apontador; e (iii) subtrair um apontador de outro apontador.

Vamos olhar para cada uma dessas operações. Suponha que temos declaradas as seguintes variáveis:

```
int v[10], *p, *q, i;
```

Adicionar um inteiro `j` a um apontador `p` fornece um apontador para o elemento posicionado `j` posições após `p`. Mais precisamente, se `p` aponta para o elemento `v[i]`, então `p+j` aponta para `v[i+j]`. A figura 51.3 ilustra essa idéia.

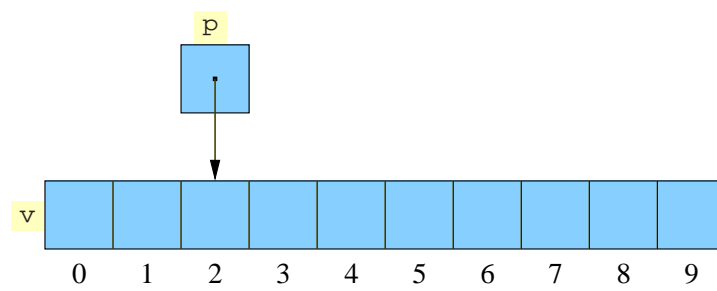
Do mesmo modo, se `p` aponta para o elemento `v[i]`, então `p-j` aponta para `v[i-j]`, como ilustrado na figura 51.4.

Ainda, quando um apontador é subtraído de outro, o resultado é a distância, medida em elementos do vetor, entre os apontadores. Dessa forma, se `p` aponta para `v[i]` e `q` aponta para `v[j]`, então `p - q` é igual a `i - j`. A figura 51.5 ilustra essa situação.

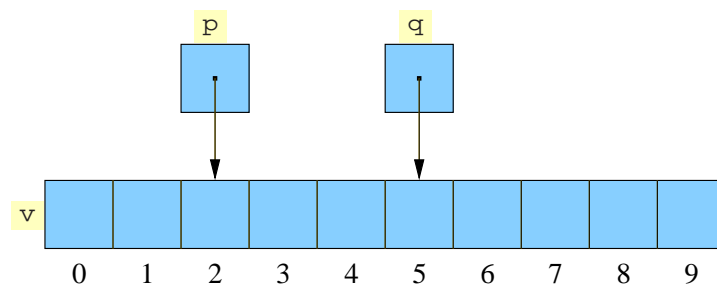
Podemos comparar variáveis apontadoras entre si usando os operadores relacionais usuais (`<`, `<=`, `>`, `>=`, `==` e `!=`). Usar os operadores relacionais para comparar dois apontadores que apontam para um mesmo vetor é uma ótima idéia. O resultado da comparação depende das posições relativas dos dois elementos do vetor. Por exemplo, depois das atribuições dadas a seguir:

```
p = &v[5];
q = &v[1];
```

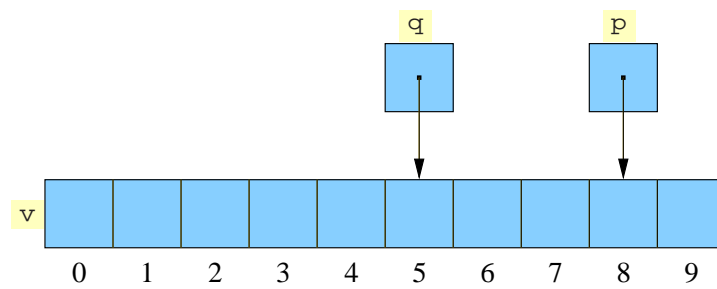
o resultado da comparação `p <= q` é falso e o resultado de `p >= q` é verdadeiro.



a

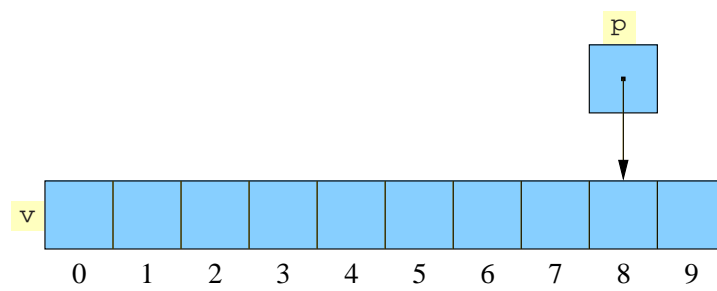


b

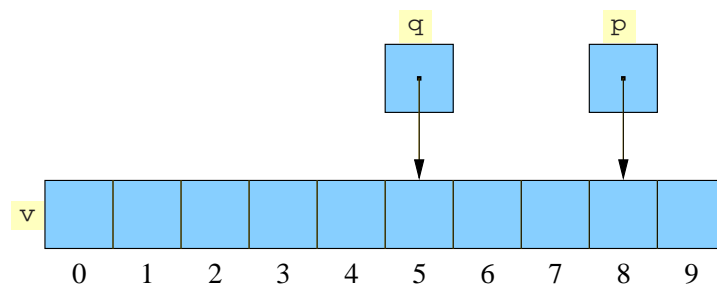


c

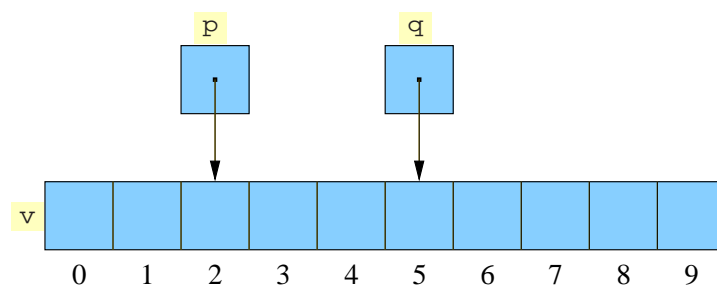
Figura 51.3: (a) `p = &v[2];` (b) `q = p + 3;` (c) `p = p + 6;`



a



b



c

Figura 51.4: (a) `p = &v[8];` (b) `q = p - 3;` (c) `p = p - 6;`

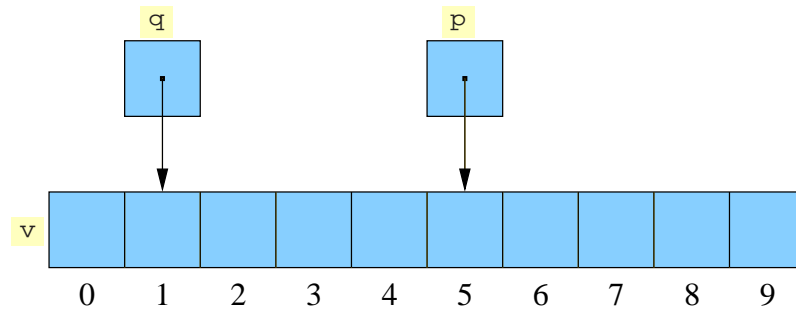


Figura 51.5: `p = &v[5];` e `q = &v[1];` A expressão `p - q` tem valor 4 e a expressão `q - p` tem valor -4.

51.2 Uso de apontadores para processamento de vetores

Usando aritmética de apontadores podemos visitar os elementos de um vetor através da atribuição de um apontador para seu início e do seu incremento em cada passo, como mostrado no trecho de código abaixo:

```
#define DIM 100
...
int v[DIM], soma, *p;
...
soma = 0;
for (p = &v[0]; p < &v[DIM]; p++)
    soma = soma + *p;
```

A condição `p < &v[DIM]` na estrutura de repetição `for` necessita de atenção especial. Apesar de estranho, é possível aplicar o operador de endereço para `v[DIM]`, mesmo sabendo que este elemento não existe no vetor `v`. Usar `v[DIM]` dessa maneira é perfeitamente seguro, já que a sentença `for` não tenta examinar o seu valor. O corpo da estrutura de repetição `for` será executado com `p` igual a `&v[0]`, `&v[1]`, ..., `&v[DIM-1]`, mas quando `p` é igual a `&v[DIM]` a estrutura de repetição termina.

Como já vimos, podemos também combinar o operador de indireção `*` com operadores de incremento `++` ou decremento `--` em sentenças que processam elementos de um vetor. Considere inicialmente o caso em que queremos armazenar um valor em um vetor e então avançar para o próximo elemento. Usando um índice, podemos fazer diretamente:

```
v[i++] = j;
```

Se `p` está apontando para um elemento de um vetor, a sentença correspondente usando esse apontador seria:

```
*p++ = j;
```

Devido à precedência do operador `++` sobre o operador `*`, o compilador enxerga essa sentença como

```
*(p++) = j;
```

O valor da expressão `*p++` é o valor de `*p`, antes do incremento. Depois que esse valor é devolvido, a sentença incrementa `p`.

A expressão `*p++` não é a única combinação possível dos operadores `*` e `++`. Podemos escrever `(*p)++` para incrementar o valor de `*p`. Nesse caso, o valor devolvido pela expressão é também `*p`, antes do incremento. Em seguida, a sentença incrementa `*p`. Ainda, podemos escrever `++*p` ou ainda `+++p`. No primeiro caso, incrementa `p` e o valor da expressão é `*p`, depois do incremento. No segundo, incrementa `*p` e o valor da expressão é `*p`, depois do incremento.

O trecho de código acima, que realiza a soma dos elementos do vetor `v` usando aritmética com apontadores, pode então ser reescrito como a seguir, usando uma combinação dos operadores `*` e `++`.

```
soma = 0;
p = &v[0];
while (p < &v[DIM])
    soma = soma + *p++;
```

51.3 Uso do identificador de um vetor como apontador

Apontadores e vetores estão intimamente relacionados. Como vimos nas seções anteriores, usamos aritmética de apontadores para trabalhar com vetores. Mas essa não é a única relação entre eles. Outra relação importante entre apontadores e vetores fornecida pela linguagem C é que o identificador de um vetor pode ser usado como um apontador para o primeiro elemento do vetor. Essa relação simplifica a aritmética com apontadores e estabelece ganho de versatilidade em ambos, apontadores e vetores.

Por exemplo, suponha que temos o vetor `v` declarado como abaixo:

```
int v[10];
```

Usando `v` como um apontador para o primeiro elemento do vetor, podemos modificar o conteúdo de `v[0]` da seguinte forma:

```
*v = 7;
```

Podemos também modificar o conteúdo de `v[1]` através do apontador `v+1`:

```
*(v+1) = 12;
```

Em geral, `v+i` é o mesmo que `&v[i]` e `*(v+i)` é equivalente a `v[i]`. Em outras palavras, índices de vetores podem ser vistos como uma forma de aritmética de apontadores.

O fato de que o identificador de um vetor pode servir como um apontador facilita nossa programação de estruturas de repetição que percorrem vetores. Considere a estrutura de repetição do exemplo dado na seção anterior:

```
soma = 0;
for (p = &v[0]; p < &v[DIM]; p++)
    soma = soma + *p;
```

Para simplificar essa estrutura de repetição, podemos substituir `&v[0]` por `v` e `&v[DIM]` por `v+DIM`, como mostra o trecho de código abaixo:

```
soma = 0;
for (p = v; p < v+DIM; p++)
    soma = soma + *p;
```

Apesar de podermos usar o identificador de um vetor como um apontador, não é possível atribuir-lhe um novo valor. A tentativa de fazê-lo apontar para qualquer outro lugar é um erro, como mostra o trecho de código abaixo:

```
while (*v != 0)
    v++;
```

O programa 51.1 mostra um exemplo do uso desses conceitos, realizando a impressão dos elementos de um vetor na ordem inversa da qual foram lidos.

Programa 51.1: Imprime os elementos na ordem inversa da de leitura.

```
1  #include <stdio.h>
2  #define N 10
3  int main(void)
4  {
5      int v[N], *p;
6      printf("Informe %d números: ", N);
7      for (p = v; p < v+N; p++)
8          scanf("%d", p);
9      printf("Em ordem reversa: ");
10     for (p = v+N-1; p >= v; p--)
11         printf(" %d", *p);
12     printf("\n");
13     return 0;
14 }
```

Outro uso do identificador de um vetor como um apontador é quando um vetor é um argumento em uma chamada de função. Nesse caso, o vetor é sempre tratado como um apontador. Considere a seguinte função que recebe um vetor de n números inteiros e devolve um maior elemento nesse vetor.

```
int max(int v[], int n)
{
    int i, maior;
    maior = v[0];
    for (i = 1; i < n; i++)
        if (v[i] > maior)
            maior = v[i];
    return maior;
}
```

Suponha que chamamos a função `max` da seguinte forma:

```
M = max(U, N);
```

Essa chamada faz com que o endereço do primeiro compartimento do vetor `U` seja atribuído à `v`. O vetor `U` não é de fato copiado.

Para indicar que não queremos que um parâmetro que é um vetor não seja modificado, podemos incluir a palavra reservada `const` precedendo a sua declaração.

Quando uma variável simples é passada para uma função, isto é, quando é um argumento de uma função, seu valor é copiado no parâmetro correspondente. Então, qualquer alteração no parâmetro correspondente não afeta a variável. Em contraste, um vetor usado como um argumento não está protegido contra alterações, já que não ocorre uma cópia do vetor todo. Desse modo, o tempo necessário para passar um vetor a uma função independe de seu tamanho. Não há perda por passar vetores grandes, já que nenhuma cópia do vetor é realizada. Além disso, um *parâmetro* que é um vetor pode ser declarado como um apontador. Por exemplo, a função `max` descrita acima pode ser declarada como a seguir:

```
int max(int *v, int n)
{
    ...
}
```

Neste caso, declarar `v` como sendo um apontador é equivalente a declarar `v` como sendo um vetor. O compilador trata ambas as declarações como idênticas.

Apesar de a declaração de um *parâmetro* como um vetor ser equivalente à declaração do mesmo *parâmetro* como um apontador, o mesmo não vale para uma variável. A declaração a seguir:

```
int v[10];
```

faz com que o compilador reserve espaço para 10 números inteiros. Por outro lado, a declaração abaixo:

```
int *v;
```

faz o compilador reservar espaço para uma variável apontadora. Nesse último caso, `v` não é um vetor e tentar usá-lo como tal pode causar resultados desastrosos. Por exemplo, a atribuição:

```
*v = 7;
```

armazena o valor 7 onde `v` está apontando. Como não sabemos para onde `v` está apontando, o resultado da execução dessa linha de código é imprevisível.

Do mesmo modo, podemos usar uma variável apontadora, que aponta para uma posição de um vetor, como um vetor. O trecho de código a seguir ilustra essa afirmação.

```

#define DIM 100
...
int v[DIM], soma, *p;
...
soma = 0;
p = v;
for (i = 0; i < DIM; i++)
    soma = soma + p[i];

```

O compilador trata `p[i]` como `*(p+i)`, que é uma forma possível de usar aritmética com apontadores. Essa possibilidade de uso, que parece um tanto estranha à primeira vista, é muito útil em alocação dinâmica de memória, como veremos em breve.

Exercícios

51.1 Suponha que as seguintes declarações foram realizadas:

```

int v[] = {5, 15, 34, 54, 14, 2, 52, 72};
int *p = &v[1], *q = &v[5];

```

- (a) Qual o valor de `*(p + 3)`?
- (b) Qual o valor de `*(q - 3)`?
- (c) Qual o valor de `q - p`?
- (d) A expressão `p < q` tem valor verdadeiro ou falso?
- (e) A expressão `*p < *q` tem valor verdadeiro ou falso?

51.2 Qual o conteúdo do vetor `v` após a execução do seguinte trecho de código?

```

#define N 10
int v[N] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int *p = &v[0], *q = &v[N-1], temp;
while (p < q) {
    temp = *p;
    *p++ = *q;
    *q-- = temp;
}

```

51.3 Suponha que `v` é um vetor e `p` é um apontador. Considere que a atribuição `p = v;` foi realizada previamente. Quais das expressões abaixo não são permitidas? Das restantes, quais têm valor verdadeiro?

- (a) `p == v[0]`
- (b) `p == &v[0]`
- (c) `*p == v[0]`
- (d) `p[0] == v[0]`

51.4 Escreva um programa que leia uma mensagem e a imprima em ordem reversa. Use a função `getchar` para ler caractere por caractere, armazenando-os em um vetor. Pare quando encontrar um caractere de mudança de linha `'\n'`. Faça o programa de forma a usar um apontador, ao invés de um índice como um número inteiro, para controlar a posição corrente no vetor.