

Ponteiros e vetores

Aula 11

Diego Padilha Rubert

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação II

Conteúdo da aula

- 1 Introdução
- 2 Aritmética com ponteiros
- 3 Uso de ponteiros para processamento de vetores
- 4 Uso do identificador de um vetor como ponteiro
- 5 Exercícios

- ▶ a linguagem C nos permite usar expressões aritméticas de adição e subtração com ponteiros que apontam para elementos de vetores
- ▶ forma alternativa de trabalhar com vetores e seus índices
- ▶ relação íntima entre ponteiros e vetores na linguagem C
- ▶ programa executável mais eficiente quando usamos ponteiros para vetores

- ▶ a linguagem C nos permite usar expressões aritméticas de adição e subtração com ponteiros que apontam para elementos de vetores
- ▶ forma alternativa de trabalhar com vetores e seus índices
- ▶ relação íntima entre ponteiros e vetores na linguagem C
- ▶ programa executável mais eficiente quando usamos ponteiros para vetores

- ▶ a linguagem C nos permite usar expressões aritméticas de adição e subtração com ponteiros que apontam para elementos de vetores
- ▶ forma alternativa de trabalhar com vetores e seus índices
- ▶ relação íntima entre ponteiros e vetores na linguagem C
- ▶ programa executável mais eficiente quando usamos ponteiros para vetores

- ▶ a linguagem C nos permite usar expressões aritméticas de adição e subtração com ponteiros que apontam para elementos de vetores
- ▶ forma alternativa de trabalhar com vetores e seus índices
- ▶ relação íntima entre ponteiros e vetores na linguagem C
- ▶ programa executável mais eficiente quando usamos ponteiros para vetores

- ▶ suponha que temos declaradas as seguintes variáveis:

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

- ▶ podemos fazer o ponteiro p apontar para o primeiro elemento do vetor v fazendo a seguinte atribuição:

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

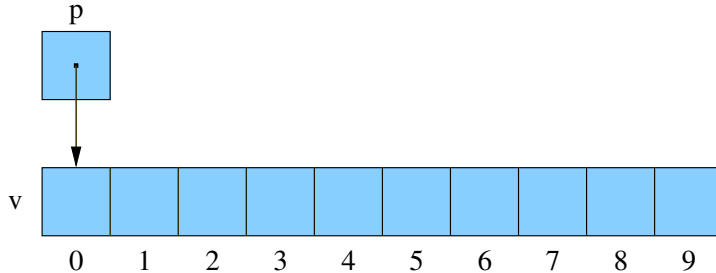
- ▶ suponha que temos declaradas as seguintes variáveis:

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

- ▶ podemos fazer o ponteiro p apontar para o primeiro elemento do vetor v fazendo a seguinte atribuição:

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

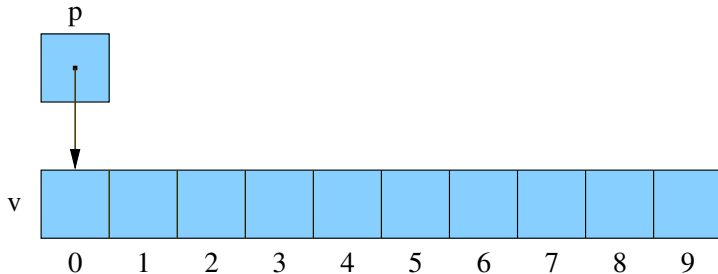

Aritmética com ponteiros



Aritmética com ponteiros

- ▶ podemos acessar o primeiro compartimento de v através de p :

```
*p = 5;
```



- ▶ podemos executar **aritmética com ponteiros** ou **aritmética com endereços** sobre p e assim acessamos outros elementos do vetor v
- ▶ a linguagem C possibilita três formas de aritmética com ponteiros:
 - ▶ adicionar um número inteiro a um ponteiro
 - ▶ subtrair um número inteiro de um ponteiro
 - ▶ subtrair um ponteiro de outro ponteiro

- ▶ podemos executar **aritmética com ponteiros** ou **aritmética com endereços** sobre p e assim acessamos outros elementos do vetor v
- ▶ a linguagem C possibilita três formas de aritmética com ponteiros:
 - ▶ adicionar um número inteiro a um ponteiro
 - ▶ subtrair um número inteiro de um ponteiro
 - ▶ subtrair um ponteiro de outro ponteiro

- ▶ suponha que temos declaradas as seguintes variáveis:

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

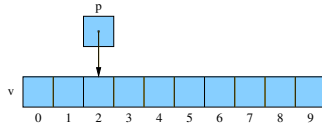
- ▶ se p aponta para o elemento $v[i]$, então $p + j$ aponta para $v[i+j]$

- ▶ suponha que temos declaradas as seguintes variáveis:

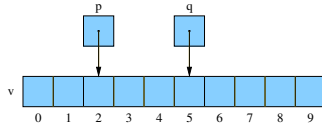
```
int v[10], *p, *q, i;
```

- ▶ se p aponta para o elemento $v[i]$, então $p + j$ aponta para $v[i+j]$

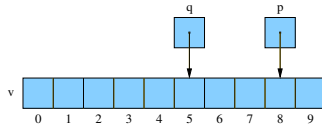
Aritmética com ponteiros



a



b

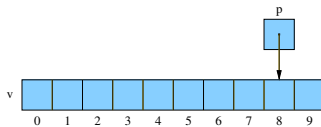


c

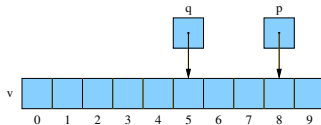
(a) $p = \&v[2];$ (b) $q = p + 3;$ (c) $p = p + 6;$

- ▶ se p aponta para o elemento $v[i]$, então $p - j$ aponta para $v[i-j]$

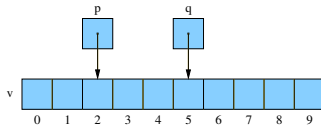
Aritmética com ponteiros



a



b



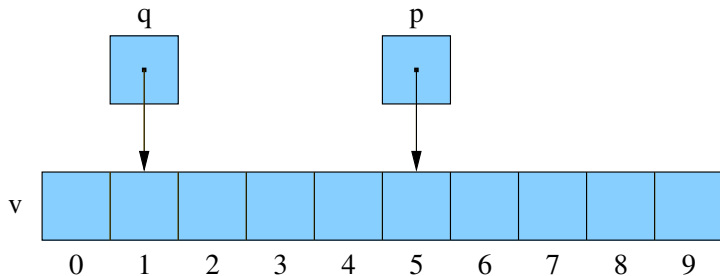
c

(a) $p = \&v[8];$ (b) $q = p - 3;$ (c) $p = p - 6;$

- ▶ quando um ponteiro é subtraído de outro, o resultado é a distância, medida em elementos do vetor, entre os ponteiros
- ▶ se p aponta para `v[i]` e q aponta para `v[j]`, então $p - q$ é igual a $i - j$

- ▶ quando um ponteiro é subtraído de outro, o resultado é a distância, medida em elementos do vetor, entre os ponteiros
- ▶ se p aponta para $v[i]$ e q aponta para $v[j]$, então $p - q$ é igual a $i - j$

Aritmética com ponteiros



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

A expressão $p - q$ tem valor 4 e a expressão $q - p$ tem valor -4

Aritmética com ponteiros

- ▶ podemos comparar variáveis ponteiros entre si usando os operadores relacionais (`<`, `<=`, `>`, `>=`, `==` e `!=`)
- ▶ 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.

Aritmética com ponteiros

- ▶ podemos comparar variáveis ponteiros entre si usando os operadores relacionais (`<`, `<=`, `>`, `>=`, `==` e `!=`)
- ▶ 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.

- ▶ podemos comparar variáveis ponteiros entre si usando os operadores relacionais (`<`, `<=`, `>`, `>=`, `==` e `!=`)
- ▶ 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.

Uso de ponteiros para processamento de vetores

- ▶ podemos usar aritmética de ponteiros para visitar os elementos de um vetor
- ▶ fazemos isso através da atribuição de um ponteiro para seu início e do seu incremento em cada passo

```
...
...
#define DIM 100

int main(void)
{
    int v[DIM], soma, *p;
    :
    :
    soma = 0;
    for (p = &v[0]; p < &v[DIM]; p++)
        soma = soma + *p;
    :
    :
```


Uso de ponteiros para processamento de vetores

- ▶ podemos usar aritmética de ponteiros para visitar os elementos de um vetor
- ▶ fazemos isso através da atribuição de um ponteiro para seu início e do seu incremento em cada passo

```
...
...
#define DIM 100

int main(void)
{
    int v[DIM], soma, *p;
    :
    :
    soma = 0;
    for (p = &v[0]; p < &v[DIM]; p++)
        soma = soma + *p;
    :
    :
```

Uso de ponteiros para processamento de vetores

- ▶ podemos usar aritmética de ponteiros para visitar os elementos de um vetor
- ▶ fazemos isso através da atribuição de um ponteiro para seu início e do seu incremento em cada passo

```
⋮
⋮
#define DIM 100

int main(void)
{
    int v[DIM], soma, *p;
    ⋮
    soma = 0;
    for (p = &v[0]; p < &v[DIM]; p++)
        soma = soma + *p;
    ⋮
}
```

Uso de ponteiros para processamento de vetores

- ▶ podemos combinar o operador de indireção `*` com operadores de incremento `++` ou decremento `--` em sentenças que processam elementos de um vetor
- ▶ 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 o $(i + 1)$ -ésimo elemento do vetor, a sentença correspondente usando esse ponteiro é:

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

Uso de ponteiros para processamento de vetores

- ▶ podemos combinar o operador de indireção `*` com operadores de incremento `++` ou decremento `--` em sentenças que processam elementos de um vetor
- ▶ 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 o $(i + 1)$ -ésimo elemento do vetor, a sentença correspondente usando esse ponteiro é:

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

Uso de ponteiros para processamento de vetores

- ▶ podemos combinar o operador de indireção `*` com operadores de incremento `++` ou decremento `--` em sentenças que processam elementos de um vetor
- ▶ 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 o $(i + 1)$ -ésimo elemento do vetor, a sentença correspondente usando esse ponteiro é:

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

Uso de ponteiros para processamento de vetores

- ▶ podemos combinar o operador de indireção `*` com operadores de incremento `++` ou decremento `--` em sentenças que processam elementos de um vetor
- ▶ 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 o $(i + 1)$ -ésimo elemento do vetor, a sentença correspondente usando esse ponteiro é:

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

Uso de ponteiros para processamento de vetores

- ▶ devido à precedência do operador `++` sobre o operador `*`, o compilador enxerga a sentença `*p++ = j;` 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`

Uso de ponteiros para processamento de vetores

- ▶ devido à precedência do operador `++` sobre o operador `*`, o compilador enxerga a sentença `*p++ = j;` 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`

Uso de ponteiros para processamento de vetores

- ▶ 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`
- ▶ podemos escrever `*++p` para incrementar `p` e o valor da expressão é `*p`, depois do incremento
- ▶ podemos escrever `++*p` para incrementar `*p` e o valor da expressão é `*p`, depois do incremento

Uso de ponteiros para processamento de vetores

- ▶ 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`
- ▶ podemos escrever `*++p` para incrementar `p` e o valor da expressão é `*p`, depois do incremento
- ▶ podemos escrever `++*p` para incrementar `*p` e o valor da expressão é `*p`, depois do incremento

Uso de ponteiros para processamento de vetores

- ▶ 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`
- ▶ podemos escrever `*++p` para incrementar `p` e o valor da expressão é `*p`, depois do incremento
- ▶ podemos escrever `++*p` para incrementar `*p` e o valor da expressão é `*p`, depois do incremento

Uso de ponteiros para processamento de vetores

- ▶ O trecho de código que realiza a soma dos elementos do vetor v usando aritmética com ponteiros, 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++;  
:  
:  
:
```

Uso de ponteiros para processamento de vetores

- ▶ O trecho de código que realiza a soma dos elementos do vetor v usando aritmética com ponteiros, 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++;  
⋮  
⋮
```

Uso do identificador de um vetor como ponteiro

- ▶ o identificador de um vetor pode ser usado como um ponteiro para o primeiro elemento do vetor
- ▶ isso simplifica a aritmética com ponteiros e estabelece ganho de versatilidade em ambos, ponteiros e vetores
- ▶ suponha que temos o vetor v declarado como abaixo:

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

- ▶ usando v como um ponteiro 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 ponteiro $v + 1$:

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

Uso do identificador de um vetor como ponteiro

- ▶ o identificador de um vetor pode ser usado como um ponteiro para o primeiro elemento do vetor
- ▶ isso simplifica a aritmética com ponteiros e estabelece ganho de versatilidade em ambos, ponteiros e vetores
- ▶ suponha que temos o vetor v declarado como abaixo:

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

- ▶ usando v como um ponteiro 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 ponteiro $v + 1$:

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

Uso do identificador de um vetor como ponteiro

- ▶ o identificador de um vetor pode ser usado como um ponteiro para o primeiro elemento do vetor
- ▶ isso simplifica a aritmética com ponteiros e estabelece ganho de versatilidade em ambos, ponteiros e vetores
- ▶ suponha que temos o vetor v declarado como abaixo:

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

- ▶ usando v como um ponteiro 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 ponteiro $v + 1$:

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


Uso do identificador de um vetor como ponteiro

- ▶ o identificador de um vetor pode ser usado como um ponteiro para o primeiro elemento do vetor
- ▶ isso simplifica a aritmética com ponteiros e estabelece ganho de versatilidade em ambos, ponteiros e vetores
- ▶ suponha que temos o vetor v declarado como abaixo:

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

- ▶ usando v como um ponteiro 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 ponteiro $v + 1$:

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

Uso do identificador de um vetor como ponteiro

- ▶ o identificador de um vetor pode ser usado como um ponteiro para o primeiro elemento do vetor
- ▶ isso simplifica a aritmética com ponteiros e estabelece ganho de versatilidade em ambos, ponteiros e vetores
- ▶ suponha que temos o vetor v declarado como abaixo:

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

- ▶ usando v como um ponteiro 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 ponteiro $v + 1$:

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

Uso do identificador de um vetor como ponteiro

- ▶ 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 ponteiros
- ▶ o fato que o identificador de um vetor pode servir como um ponteiro facilita nossa programação de estruturas de repetição que percorrem vetores

Uso do identificador de um vetor como ponteiro

- ▶ 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 ponteiros
- ▶ o fato que o identificador de um vetor pode servir como um ponteiro facilita nossa programação de estruturas de repetição que percorrem vetores

Uso do identificador de um vetor como ponteiro

- ▶ 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 ponteiros
- ▶ o fato que o identificador de um vetor pode servir como um ponteiro facilita nossa programação de estruturas de repetição que percorrem vetores

Uso do identificador de um vetor como ponteiro

- ▶ 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`

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

Uso do identificador de um vetor como ponteiro

- ▶ 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`

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

Uso do identificador de um vetor como ponteiro

- ▶ apesar de podermos usar o identificador de um vetor como um ponteiro, **não** é possível atribuir-lhe um novo valor
- ▶ a tentativa de fazê-lo apontar para qualquer outro lugar é um **erro**

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


Uso do identificador de um vetor como ponteiro

- ▶ apesar de podermos usar o identificador de um vetor como um ponteiro, **não** é possível atribuir-lhe um novo valor
- ▶ a tentativa de fazê-lo apontar para qualquer outro lugar é um **erro**

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

Uso do identificador de um vetor como ponteiro

- ▶ apesar de podermos usar o identificador de um vetor como um ponteiro, **não** é possível atribuir-lhe um novo valor
- ▶ a tentativa de fazê-lo apontar para qualquer outro lugar é um **erro**

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

Uso do identificador de um vetor como ponteiro

```
#include <stdio.h>

#define N 10

int main(void)
{
    int v[N], *p;

    printf("Informe %d números: ", N);
    for (p = v; p < v + N; p++)
        scanf("%d", p);

    printf("Em ordem inversa: ");
    for (p = v + N - 1; p >= v; p--)
        printf("%d ", *p);
    printf("\n");

    return 0;
}
```

Uso do identificador de um vetor como ponteiro

- ▶ outro uso do identificador de um vetor como um ponteiro é quando um vetor é um argumento em uma chamada de função
- ▶ o vetor é sempre tratado como um ponteiro

Uso do identificador de um vetor como ponteiro

- ▶ outro uso do identificador de um vetor como um ponteiro é quando um vetor é um argumento em uma chamada de função
- ▶ o vetor é sempre tratado como um ponteiro

Uso do identificador de um vetor como ponteiro

- ▶ considere a seguinte função que recebe um vetor de n números inteiros e devolve um maior elemento nesse vetor

```
int max(int n, int v[MAX])
{
    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(N, u);
```

- ▶ 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

Uso do identificador de um vetor como ponteiro

- ▶ considere a seguinte função que recebe um vetor de n números inteiros e devolve um maior elemento nesse vetor

```
int max(int n, int v[MAX])
{
    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(N, u);
```

- ▶ 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

Uso do identificador de um vetor como ponteiro

- ▶ considere a seguinte função que recebe um vetor de n números inteiros e devolve um maior elemento nesse vetor

```
int max(int n, int v[MAX])
{
    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(N, u);
```

- ▶ 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

Uso do identificador de um vetor como ponteiro

- ▶ considere a seguinte função que recebe um vetor de n números inteiros e devolve um maior elemento nesse vetor

```
int max(int n, int v[MAX])
{
    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(N, u);
```

- ▶ 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

Uso do identificador de um vetor como ponteiro

- ▶ para indicar que 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
- ▶ qualquer alteração no parâmetro correspondente não afeta a variável
- ▶ um vetor usado como um argumento não está protegido contra alterações, já que não ocorre uma cópia do vetor todo
- ▶ assim, o tempo necessário para passar um vetor a uma função independe de seu tamanho
- ▶ não há perda (de tempo) por passar vetores grandes, já que nenhuma cópia do vetor é realizada
- ▶ um *parâmetro* que é um vetor pode ser declarado como um ponteiro

Uso do identificador de um vetor como ponteiro

- ▶ para indicar que 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
- ▶ qualquer alteração no parâmetro correspondente não afeta a variável
- ▶ um vetor usado como um argumento não está protegido contra alterações, já que não ocorre uma cópia do vetor todo
- ▶ assim, o tempo necessário para passar um vetor a uma função independe de seu tamanho
- ▶ não há perda (de tempo) por passar vetores grandes, já que nenhuma cópia do vetor é realizada
- ▶ um *parâmetro* que é um vetor pode ser declarado como um ponteiro

Uso do identificador de um vetor como ponteiro

- ▶ para indicar que 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
- ▶ qualquer alteração no parâmetro correspondente não afeta a variável
- ▶ um vetor usado como um argumento não está protegido contra alterações, já que não ocorre uma cópia do vetor todo
- ▶ assim, o tempo necessário para passar um vetor a uma função independe de seu tamanho
- ▶ não há perda (de tempo) por passar vetores grandes, já que nenhuma cópia do vetor é realizada
- ▶ um *parâmetro* que é um vetor pode ser declarado como um ponteiro

Uso do identificador de um vetor como ponteiro

- ▶ para indicar que 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
- ▶ qualquer alteração no parâmetro correspondente não afeta a variável
- ▶ um vetor usado como um argumento não está protegido contra alterações, já que não ocorre uma cópia do vetor todo
- ▶ assim, o tempo necessário para passar um vetor a uma função independe de seu tamanho
- ▶ não há perda (de tempo) por passar vetores grandes, já que nenhuma cópia do vetor é realizada
- ▶ um *parâmetro* que é um vetor pode ser declarado como um ponteiro

Uso do identificador de um vetor como ponteiro

- ▶ para indicar que 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
- ▶ qualquer alteração no parâmetro correspondente não afeta a variável
- ▶ um vetor usado como um argumento não está protegido contra alterações, já que não ocorre uma cópia do vetor todo
- ▶ assim, o tempo necessário para passar um vetor a uma função independe de seu tamanho
- ▶ não há perda (de tempo) por passar vetores grandes, já que nenhuma cópia do vetor é realizada
- ▶ um *parâmetro* que é um vetor pode ser declarado como um ponteiro

Uso do identificador de um vetor como ponteiro

- ▶ para indicar que 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
- ▶ qualquer alteração no parâmetro correspondente não afeta a variável
- ▶ um vetor usado como um argumento não está protegido contra alterações, já que não ocorre uma cópia do vetor todo
- ▶ assim, o tempo necessário para passar um vetor a uma função independe de seu tamanho
- ▶ não há perda (de tempo) por passar vetores grandes, já que nenhuma cópia do vetor é realizada
- ▶ um *parâmetro* que é um vetor pode ser declarado como um ponteiro

Uso do identificador de um vetor como ponteiro

- ▶ para indicar que 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
- ▶ qualquer alteração no parâmetro correspondente não afeta a variável
- ▶ um vetor usado como um argumento não está protegido contra alterações, já que não ocorre uma cópia do vetor todo
- ▶ assim, o tempo necessário para passar um vetor a uma função independe de seu tamanho
- ▶ não há perda (de tempo) por passar vetores grandes, já que nenhuma cópia do vetor é realizada
- ▶ um *parâmetro* que é um vetor pode ser declarado como um ponteiro

Uso do identificador de um vetor como ponteiro

```
int max(int n, int *v)
{
    int i, maior;

    maior = v[0];
    for (i = 1; i < n; i++)
        if (v[i] > maior)
            maior = v[i];

    return maior;
}
```

- ▶ neste caso, declarar o parâmetro *v* como sendo um ponteiro é equivalente a declarar *v* como sendo um vetor
- ▶ o compilador trata ambas as declarações como idênticas

Uso do identificador de um vetor como ponteiro

```
int max(int n, int *v)
{
    int i, maior;

    maior = v[0];
    for (i = 1; i < n; i++)
        if (v[i] > maior)
            maior = v[i];

    return maior;
}
```

- ▶ neste caso, declarar o parâmetro *v* como sendo um ponteiro é equivalente a declarar *v* como sendo um vetor
- ▶ o compilador trata ambas as declarações como idênticas

Uso do identificador de um vetor como ponteiro

- ▶ **cuidado!** apesar de a declaração de um *parâmetro* como um vetor ser equivalente à declaração do mesmo *parâmetro* como um ponteiro, o mesmo não vale para uma *variável*
- ▶ Por exemplo, as declarações das *variáveis*

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

e

```
int *v;
```

determinam *variáveis* muito diferentes

- ▶ Assim, se logo em seguida fazemos 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

Uso do identificador de um vetor como ponteiro

- ▶ **cuidado!** apesar de a declaração de um *parâmetro* como um vetor ser equivalente à declaração do mesmo *parâmetro* como um ponteiro, o mesmo não vale para uma *variável*
- ▶ Por exemplo, as declarações das *variáveis*

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

e

```
int *v;
```

determinam *variáveis* muito diferentes

- ▶ Assim, se logo em seguida fazemos 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

Uso do identificador de um vetor como ponteiro

- ▶ **cuidado!** apesar de a declaração de um *parâmetro* como um vetor ser equivalente à declaração do mesmo *parâmetro* como um ponteiro, o mesmo não vale para uma *variável*
- ▶ Por exemplo, as declarações das *variáveis*

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

e

```
int *v;
```

determinam *variáveis* muito diferentes

- ▶ Assim, se logo em seguida fazemos 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

Uso do identificador de um vetor como ponteiro

- ▶ podemos usar uma variável ponteiro, que aponta para uma posição de um vetor, como um vetor

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

- ▶ o compilador trata a referência `p[i]` como `*(p + i)`

Uso do identificador de um vetor como ponteiro

- ▶ podemos usar uma variável ponteiro, que aponta para uma posição de um vetor, como um vetor

```
⋮  
⋮  
#define DIM 100  
  
int main(void)  
{  
    int v[DIM], soma, *p;  
    ⋮  
    soma = 0;  
    p = v;  
    for (i = 0; i < DIM; i++)  
        soma = soma + p[i];  
}
```

- ▶ o compilador trata a referência `p[i]` como `*(p + i)`

Uso do identificador de um vetor como ponteiro

- ▶ podemos usar uma variável ponteiro, que aponta para uma posição de um vetor, como um vetor

```
:  
:  
#define DIM 100  
  
int main(void)  
{  
    int v[DIM], soma, *p;  
    :  
    soma = 0;  
    p = v;  
    for (i = 0; i < DIM; i++)  
        soma = soma + p[i];  
}
```

- ▶ o compilador trata a referência `p[i]` como `*(p + i)`

1. Suponha que as declarações e atribuições simultâneas tenham sido realizadas nas variáveis listadas abaixo:

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

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

2. Qual o conteúdo do vetor v após a execução do seguinte trecho de código?

```
⋮
⋮
#define N 10

int main(void)
{
    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;
    }
    ⋮
}
```

3. Suponha que v é um vetor e p é um ponteiro. 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]`

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 ponteiro, ao invés de um índice como um número inteiro, para controlar a posição corrente no vetor.

