Ponteiros

Prof. Thiago Felski Pereira, MSc.

Introdução

- Os conceitos de **endereço** e **ponteiro** são fundamentais em qualquer linguagem de programação. Em C/C++, esses conceitos são explícitos; em algumas outras linguagens eles são ocultos (e representados pelo conceito mais abstrato de referência)

- Embora seja muito difundida a ideia da criação de linguagens que não suportem acesso a ponteiros, pressupondo que todos os programadores são inexperientes, a falta deste recurso limita as capacidades de interação de programas com o hardware
  - Em outras palavras, a falta de um meio de manipular ponteiros faz a linguagem limitada ou dependente de fabricantes de bibliotecas que acessem o hardware

- A linguagem C++ disponibiliza o recurso para quem deseja utilizá-lo e também apresenta diversos outros recursos que são alternativas ao uso de ponteiros quando eles não são imprescindíveis

Endereços

- A memória RAM (*Random Access Memory*) de qualquer computador é uma sequência de bytes. A posição (0, 1, 2, 3, etc.) que um byte ocupa na sequência é o **endereço** (*address*) do byte
  - É como o endereço de uma casa em uma longa rua que tem casas de um lado só
  - Se `e` é o endereço de um byte então `e+1` é o endereço do byte seguinte

- Cada variável de um programa ocupa um certo número de bytes consecutivos na memória do computador
  - `char` ocupa 1 Byte
  - `int` ocupa 4 Bytes
  - `double` ocupa 8 bytes

Endereços

- O número exato de bytes de uma variável é dado pelo operador `sizeof`
  - A expressão `sizeof (char)`, por exemplo, vale 1 em todos os computadores e a expressão `sizeof (int)` vale 4 em muitos computadores.

- Cada variável (em particular, cada registro e cada vetor) na memória tem um endereço. Na maioria dos computadores, o endereço de uma variável é o endereço do seu primeiro byte. Por exemplo, depois das declarações

```
char c;
int i;
struct ponto {
    int x, y;
};
int v[4];
```

c	89421
i	89422
ponto	89426
v[0]	89434
v[1]	89438
v[2]	89442

Endereços

- O endereço de uma variável é dado pelo operador `&`
- Assim, se `i` é uma variável então `&i` é o seu endereço.
  - No exemplo, `&i` vale `89422` e `&v[3]` vale `89446`.

```
char c;
int i;
struct ponto {
    int x, y;
};
int v[4];
```

c	89421
i	89422
ponto	89426
v[0]	89434
v[1]	89438
v[2]	89442

Exercícios

- Exercício 01
  - Compile e execute o seguinte programa:

```
int main () {
    struct DATA {
        int dia, mes, ano;
    };

    cout<<sizeof(DATA);
    return 0;
}
```

Exercícios

- Exercício 02
  - Compile e execute o seguinte programa:

```
int main () {
    int i = 1234;
    cout<<i<<"\n";
    cout<<&i<<"\n";
    return 0;
}
```

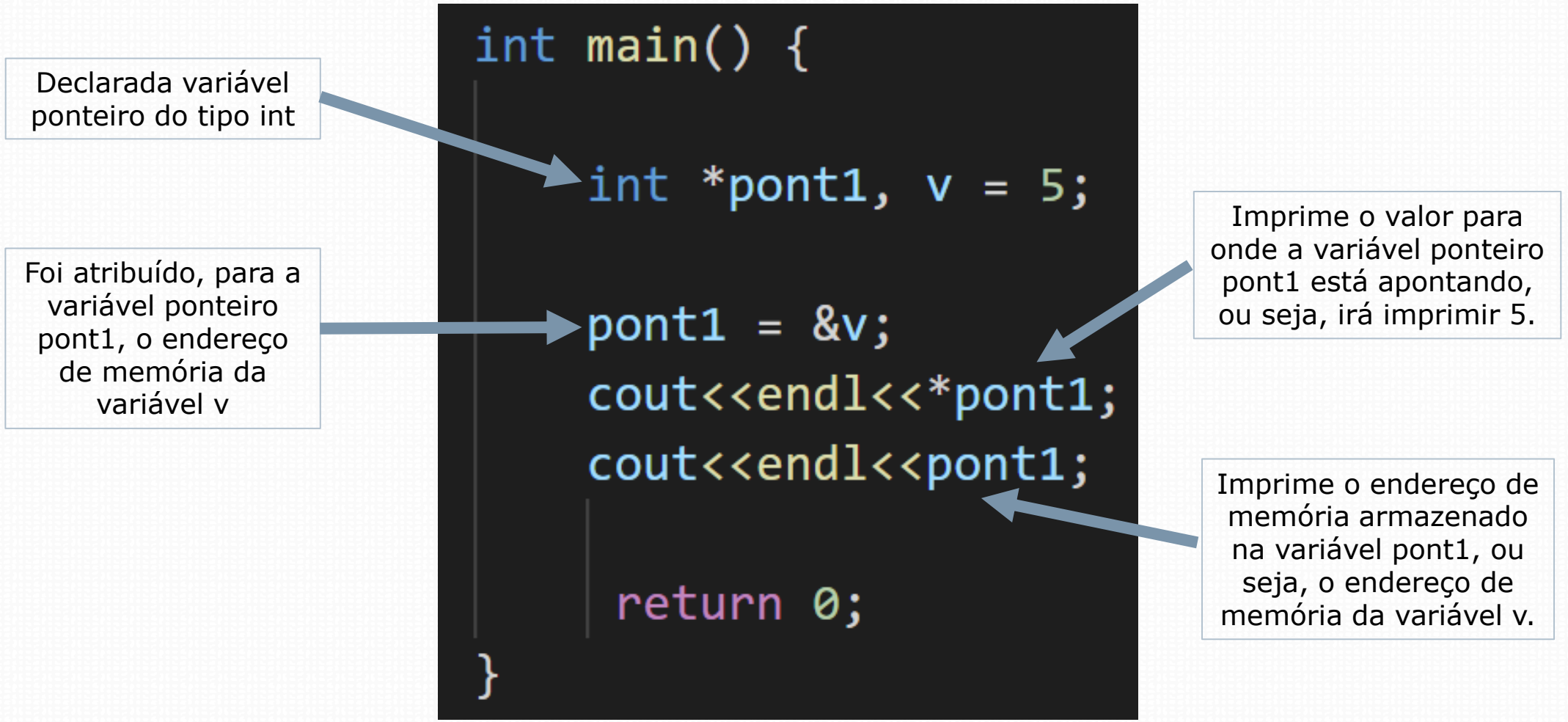
Ponteiro

- Um ponteiro (pointer) é um tipo especial de variável que armazena um endereço. Um ponteiro pode ter o valor **`NULL`**
  - que é um endereço inválido

- Se um ponteiro p armazena o endereço de uma variável `i`, podemos dizer **`p` aponta para `i`** ou **`p` é o endereço de `i`**.
  - Em termos um pouco mais abstratos, diz-se que `p` é uma *referência* à variável `i`.

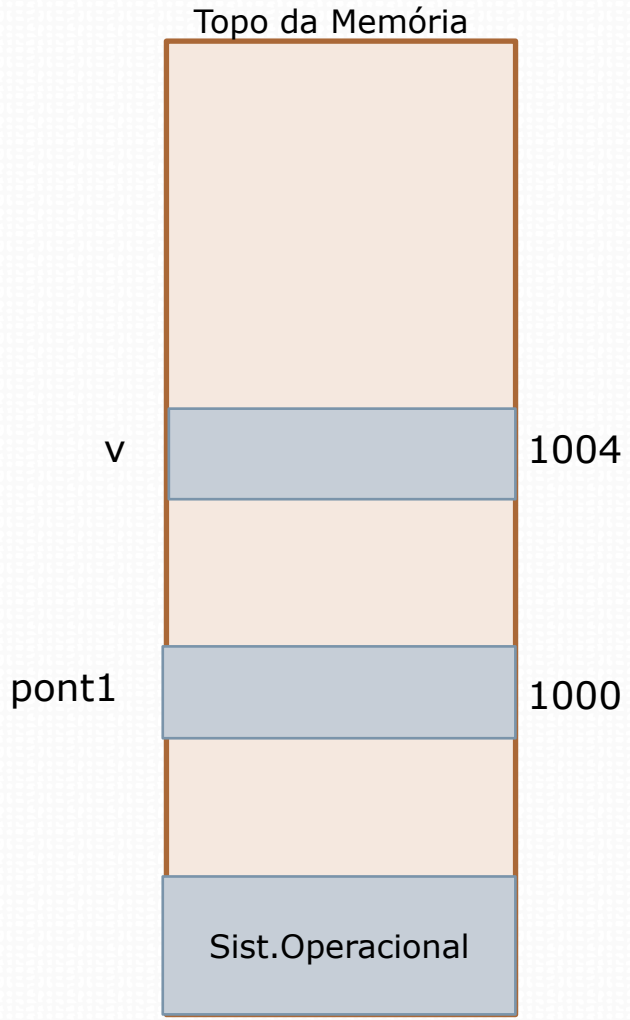
- Se um ponteiro `p` tem valor diferente de `NULL` então **`*p`**
  - é o valor da variável apontada por `p`
  - Por exemplo, se `i` é uma variável e `p` vale `&i` então dizer `*p` é o mesmo que dizer `i`.

Ponteiro: Exemplo



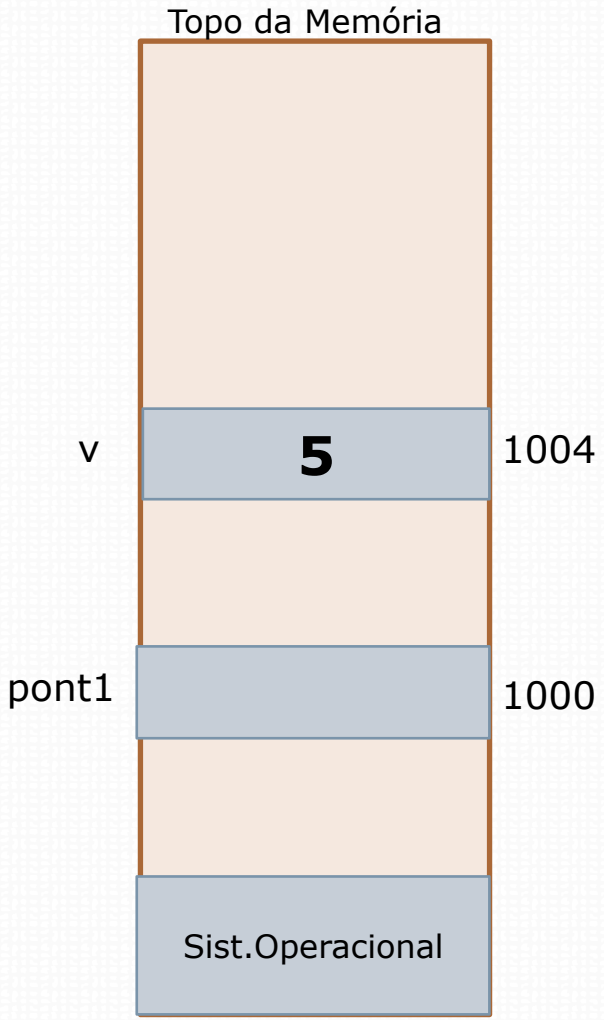
Ponteiro: Exemplo

- Tem-se a variável **v** e um ponteiro **pont1**.



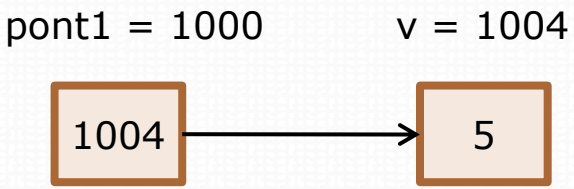
Ponteiro: Exemplo

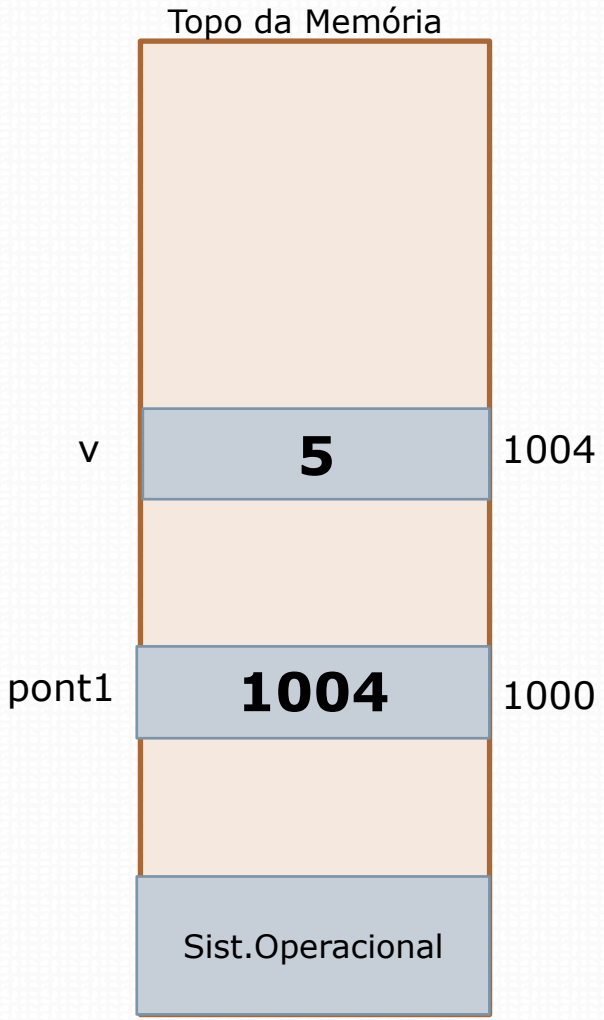
- Tem-se a variável **`v`** e um ponteiro **`pont1`**.
- O conteúdo (valor) da variável **`v`** é **`5`**.



Ponteiro: Exemplo

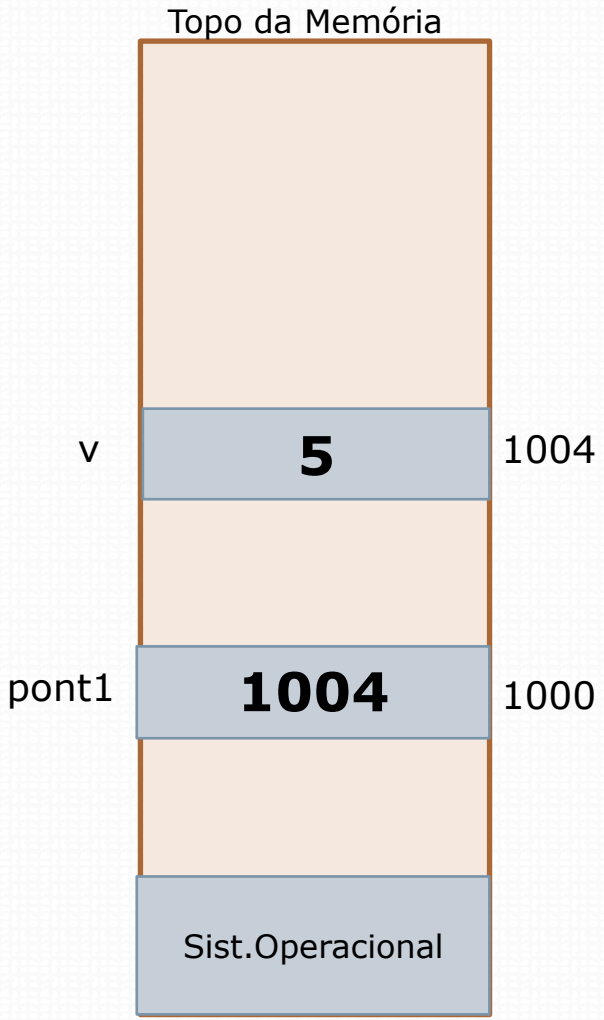
- Tem-se a variável **`v`** e um ponteiro **`pont1`**.
- O conteúdo (valor) da variável **`v`** é **`5`**
- O ponteiro **`pont1`** aponta para o endereço da variável **`v`**





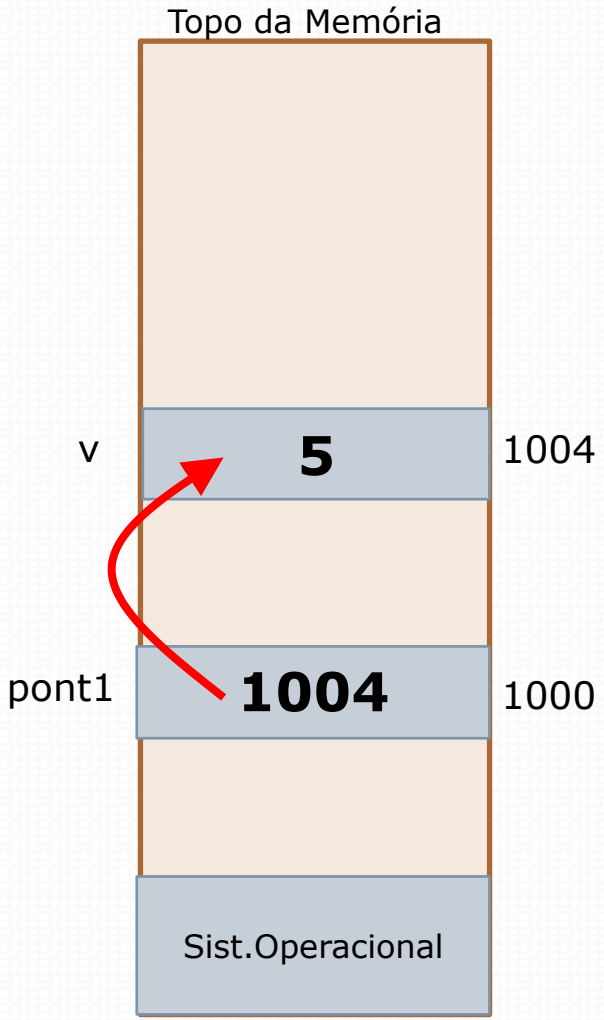
Ponteiro: Exemplo

- Tem-se a variável **`v`** e um ponteiro **`pont1`**.
- O conteúdo (valor) da variável **`v`** é **`5`**.
- O ponteiro **`pont1`** aponta para o endereço da variável **`v`**.
- Qual é o conteúdo da posição de memória apontada por **`pont1`**?



Ponteiro: Exemplo

- Tem-se a variável **`v`** e um ponteiro **`pont1`**.
- O conteúdo (valor) da variável **`v`** é **`5`**.
- O ponteiro **`pont1`** aponta para o endereço da variável **`v`**.
- Qual é o conteúdo da posição de memória apontada por **`pont1`**?
  - **`[1004] = 5`**



Exercícios

- Exercício 03
  - Pode-se atribuir o valor de uma variável ponteiro para outra variável ponteiro:

```
int main() {
    int *p1, *p2, v = 5, x = 3;

    p1 = &v;
    p2 = &x;
    *p1 = *p2;
    cout<<*p1<<"\n"<<*p2<<"\n";

    *p1 = 9;
    cout<<"\n\n"<<*p1<<"\n"<<*p2;
    p2 = p1;
    cout<<endl<<endl<<*p1;
    cout<<endl<<*p2;

    *p2 = 10;
    cout<<"\n\n"<<*p1<<"\n"<<*p2;

    return 0;
}
```

Ponteiro: Aplicação

- Suponha que precisamos de uma função que troque os valores de duas variáveis inteiras, digamos i e j

```
void troca (int i, int j) {
    int temp;
    temp = i; i = j; j = temp;
}
```

- A função, acima, não produz o efeito desejado, pois recebe apenas **os valores das variáveis** e não as variáveis propriamente ditas

Ponteiro: Aplicação

- Ponteiros resolveriam o problema da seguinte forma

```
void troca (int *i, int *j) {
    int temp;
    temp = *i; *i = *j; *j = temp;
}
```

- Para aplicar essa função às variáveis i e j basta dizer

```
troca(&i,&j);
```

- Ou então

```
int *p, *q;
p = &i; q = &j;
troca (p, q);
```

Exercícios

- **Exercício 04**: Porque o código abaixo está errado?

```
void troca (int *i, int *j) {
    int *temp;
    *temp = *i; *i = *j; *j = *temp;
}
```

- **Exercício 05**: Um ponteiro pode ser usado para dizer a uma função onde ela deve depositar o resultado de seus cálculos. Escreva uma função `hm` que converta minutos em horas-e-minutos. A função recebe um inteiro `mnts` e os endereços de duas variáveis inteiras, digamos `h` e `m`, e atribui valores a essas variáveis de modo que `m` seja menor que `60` e que `60*h + m` seja igual a `mnts`. Escreva também uma função `main` que use a função `hm`

Exercícios

- **Exercício 06:** Escreva uma função `mm` que receba um vetor inteiro `v[0..n-1]` e os endereços de duas variáveis inteiras, digamos `min` e `max`, e deposite nessas variáveis o valor de um elemento `mínimo` e o valor de um elemento `máximo` do vetor. Escreva também uma função `main` que use a função `mm`

Aritmética de Endereços

- Os elementos de qualquer vetor são armazenados em bytes consecutivos na memória do computador. Se cada elemento do vetor ocupa k bytes, a diferença entre os endereços de dois elementos consecutivos é k. Mas o **compilador** ajusta os detalhes internos de modo a criar a ilusão de que a diferença entre os endereços de dois elementos consecutivos é 1, qualquer que seja o valor de k. Por exemplo, depois da declaração

```
int *v = (int*)malloc(sizeof(int*)*100);
```

- O endereço do primeiro elemento do vetor é v, o endereço do segundo elemento é v+1, o endereço do terceiro elemento é v+2, etc.

Aritmética de Endereços

- Se `i` é uma variável do tipo `int` então `v+i` é o endereço do `(i+1)-ésimo` elemento do vetor. As expressões `v + i` e `&v[i]` têm exatamente o mesmo valor e portanto as atribuições

```
*(v+i) = 789;
v[i] = 789;
```

- Analogamente, qualquer dos dois fragmentos de código abaixo pode ser usado para preencher o vetor v

```
for (i = 0; i < 100; ++i)  cin>>v[i];
for (i = 0; i < 100; ++i)  cin>>*(v+i);
```

Aritmética de Endereços

- ...

```
for (i = 0; i < 100; ++i)  cin>>v[i];
for (i = 0; i < 100; ++i)  cin>>*(v+i);
```

- Todas essas considerações também valem se o vetor for alocado estaticamente (ou seja, antes que o programa comece a ser executado) por uma declaração como

```
int v[100];
```

- mas nesse caso `v` é uma espécie de “ponteiro constante”, cujo valor não pode ser alterado.

Exercícios

- **Exercício 07:** Suponha que os elementos de um vetor `v` são do tipo `int` e cada `int` ocupa 4 bytes no seu computador. Se o endereço de `v[0]` é `55000`, qual o valor da expressão `v + 3`?

- **Exercício 08:** Suponha que `i` é uma variável inteira e `v` um vetor de inteiros. Descreva, em português, a sequência de operações que deve ser executada para calcular o valor da expressão `&v[i + 9]`

- **Exercício 09:** Suponha que `v` é um vetor. Descreva a diferença conceitual entre as expressões `v[3]` e `v + 3`

Exercícios

- **Exercício 10:** O que faz a seguinte função?

```
void imprime (char *v, int n) {
    char *c;
    for (c = v; c < v + n; c++)
        cout<< *c;
}
```