# Laboratório de Programação

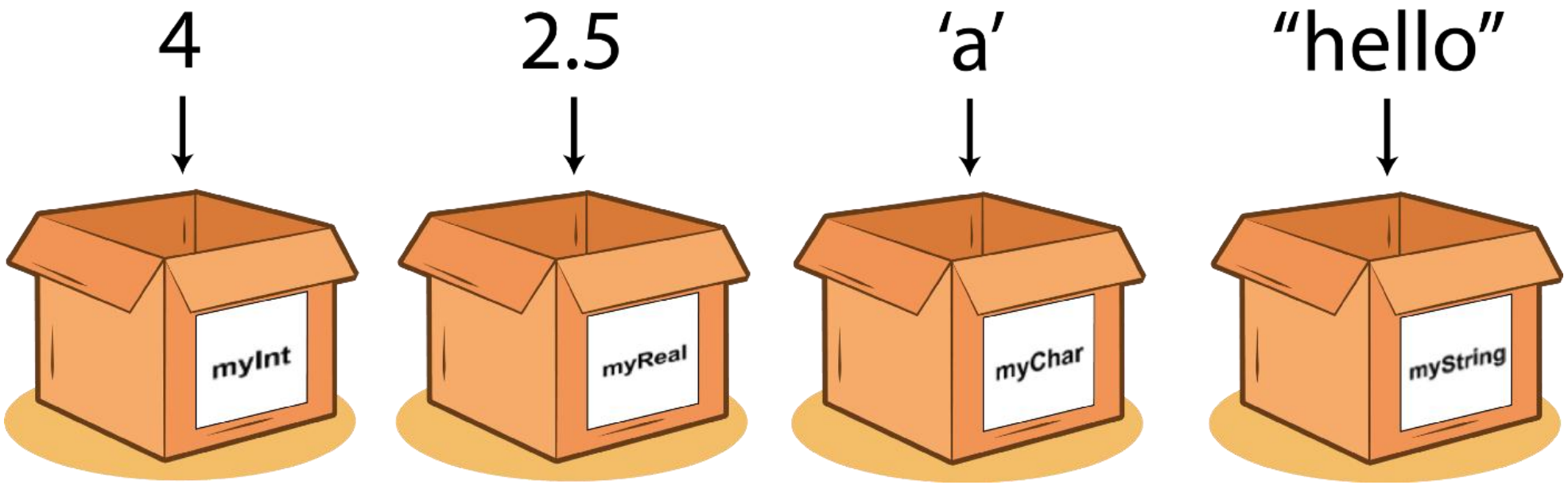
Profa. Ms. Valéria Pinheiro

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS DE RUSSAS

# Ponteiros

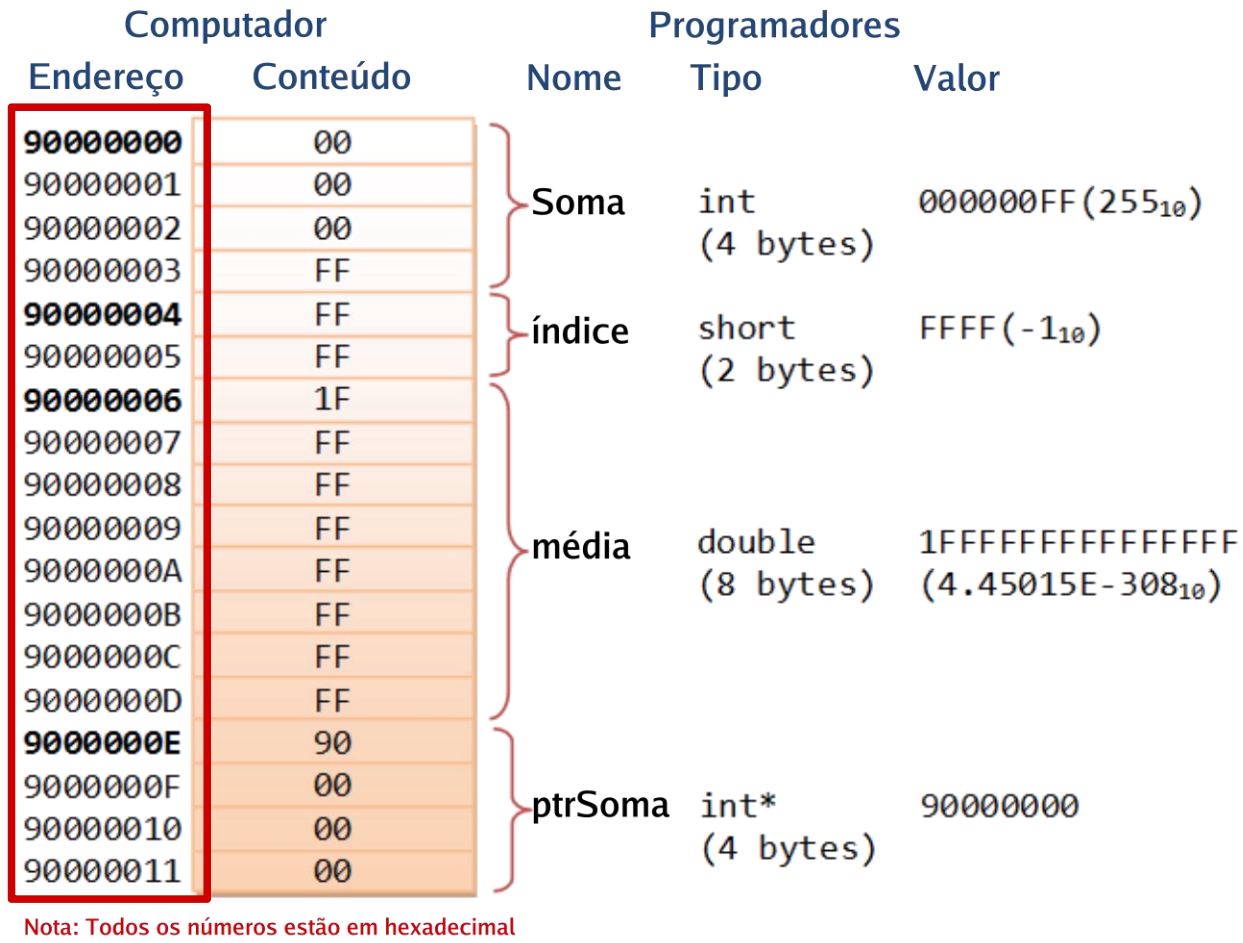
# Variáveis e Memória

- Costumamos pensar em variáveis como caixas, que guardam coisas de um determinado tipo
  - Mas onde estão essas caixas? → Na memória!
  - Onde na memória?



# Variáveis e Memória

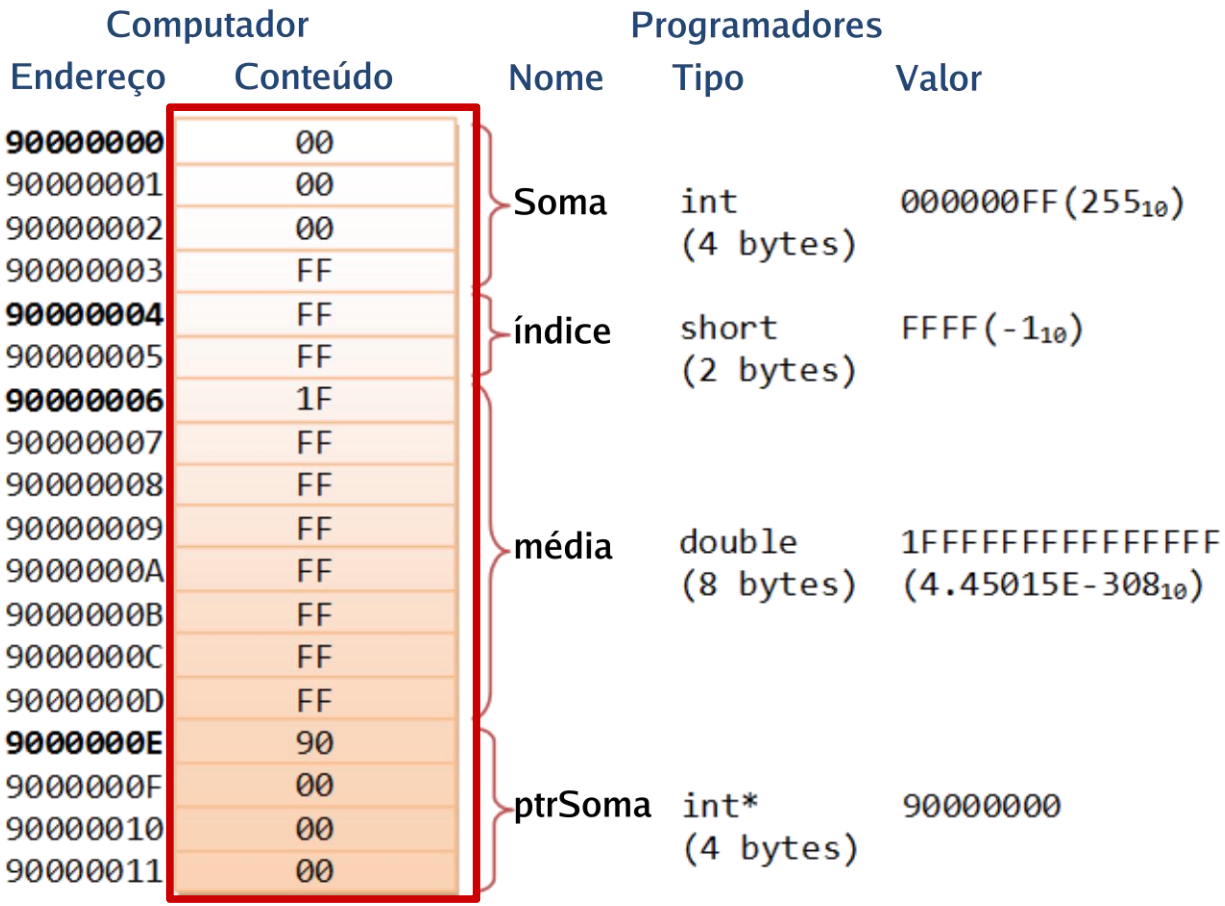
- Essa é a realidade...




Nota: Todos os números estão em hexadecimal

# Variáveis e Memória

- Essa é a realidade...




Nota: Todos os números estão em hexadecimal

# Variáveis e Memória

- Essa é a realidade...

| Computador | | Programadores | | |
|---|---|---|---|---|
| Endereço | Conteúdo | Nome | Tipo | Valor |
| **90000000** | 00 | Soma | int (4 bytes) | 000000FF($255_{10}$) |
| 90000001 | 00 | | | |
| 90000002 | 00 | | | |
| 90000003 | FF | | | |
| **90000004** | FF | índice | short (2 bytes) | FFFF($-1_{10}$) |
| 90000005 | FF | | | |
| **90000006** | 1F | média | double (8 bytes) | 1FFFFFFFFFFFFFF ($4.45015E-308_{10}$) |
| 90000007 | FF | | | |
| 90000008 | FF | | | |
| 90000009 | FF | | | |
| 9000000A | FF | | | |
| 9000000B | FF | | | |
| 9000000C | FF | | | |
| 9000000D | FF | | | |
| **9000000E** | 90 | ptrSoma | int* (4 bytes) | 90000000 |
| 9000000F | 00 | | | |
| 90000010 | 00 | | | |
| 90000011 | 00 | | | |

Nota: Todos os números estão em hexadecimal

# Variáveis e Memória

- Essa é a realidade...

| Computador | | Programadores | | |
|---|---|---|---|---|
| Endereço | Conteúdo | Nome | Tipo | Valor |
| **90000000** | 00 | Soma | int (4 bytes) | 000000FF($255_{10}$) |
| 90000001 | 00 | | | |
| 90000002 | 00 | | | |
| 90000003 | FF | | | |
| **90000004** | FF | índice | short (2 bytes) | FFFF($-1_{10}$) |
| 90000005 | FF | | | |
| **90000006** | 1F | média | double (8 bytes) | 1FFFFFFFFFFFFFF ($4.45015E-308_{10}$) |
| 90000007 | FF | | | |
| 90000008 | FF | | | |
| 90000009 | FF | | | |
| 9000000A | FF | | | |
| 9000000B | FF | | | |
| 9000000C | FF | | | |
| 9000000D | FF | | | |
| **9000000E** | 90 | ptrSoma | int* (4 bytes) | 90000000 |
| 9000000F | 00 | | | |
| 90000010 | 00 | | | |
| 90000011 | 00 | | | |

Nota: Todos os números estão em hexadecimal

# Variáveis e Memória

- Essa é a realidade...

| Computador | | Programadores | | |
|---|---|---|---|---|
| Endereço | Conteúdo | Nome | Tipo | Valor |
| **90000000** | 00 | Soma | int (4 bytes) | 000000FF($255_{10}$) |
| 90000001 | 00 | | | |
| 90000002 | 00 | | | |
| 90000003 | FF | | | |
| **90000004** | FF | índice | short (2 bytes) | FFFF($-1_{10}$) |
| 90000005 | FF | | | |
| **90000006** | 1F | média | double (8 bytes) | 1FFFFFFFFFFFFFF ($4.45015E-308_{10}$) |
| 90000007 | FF | | | |
| 90000008 | FF | | | |
| 90000009 | FF | | | |
| 9000000A | FF | | | |
| 9000000B | FF | | | |
| 9000000C | FF | | | |
| 9000000D | FF | | | |
| **9000000E** | 90 | ptrSoma | int* (4 bytes) | 90000000 |
| 9000000F | 00 | | | |
| 90000010 | 00 | | | |
| 90000011 | 00 | | | |

Nota: Todos os números estão em hexadecimal

# Variáveis e Memória

- Faça esse teste:

```c
#include <stdio.h>

int main(){
    char   a = 'R';
    int    b = 2;
    float  c = 4.5f;
    double d = 3.14;

    printf("size(a) = %ld, location(a) = %p\n",sizeof(a),&a);
    printf("size(b) = %ld, location(b) = %p\n",sizeof(b),&b);
    printf("size(c) = %ld, location(c) = %p\n",sizeof(c),&c);
    printf("size(d) = %ld, location(d) = %p\n",sizeof(d),&d);

    return 0;
}
```

# Variáveis e Memória

- Faça esse teste:

```
Saída

size(a) = 1, location(a) = 0x7ffcf1436ae7
size(b) = 4, location(b) = 0x7ffcf1436ae8
size(c) = 4, location(c) = 0x7ffcf1436aec
size(d) = 8, location(d) = 0x7ffcf1436af0
```

```c
#include <stdio.h>

int main(){
    char   a = 'R';
    int    b = 2;
    float  c = 4.5f;
    double d = 3.14;

    printf("size(a) = %ld, location(a) = %p\n",sizeof(a),&a); Endereços de memória
    printf("size(b) = %ld, location(b) = %p\n",sizeof(b),&b);
    printf("size(c) = %ld, location(c) = %p\n",sizeof(c),&c);
    printf("size(d) = %ld, location(d) = %p\n",sizeof(d),&d); Inteiros em hexadecimal

    return 0;
}
```

# Variáveis e Memória

- Faça esse teste:

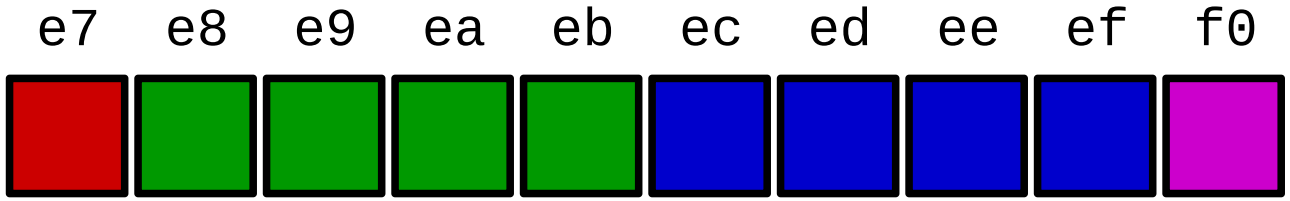
```c
#include <stdio.h>

int main(){
    char   a = 'R';
    int    b = 2;
    float  c = 4.5f;
    double d = 3.14;

    printf("size(a) = %ld, location(a) = %p\n",sizeof(a),&a);
    printf("size(b) = %ld, location(b) = %p\n",sizeof(b),&b);
    printf("size(c) = %ld, location(c) = %p\n",sizeof(c),&c);
    printf("size(d) = %ld, location(d) = %p\n",sizeof(d),&d);

    return 0;
}
```

*Saída*

```
size(a) = 1, location(a) = 0x7ffcf1436ae7
size(b) = 4, location(b) = 0x7ffcf1436ae8
size(c) = 4, location(c) = 0x7ffcf1436aec
size(d) = 8, location(d) = 0x7ffcf1436af0
```

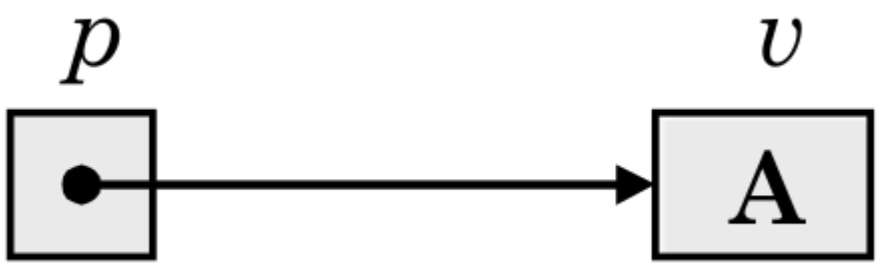
*Endereços de memória*

*Inteiros em hexadecimal*



# Definição

- Ponteiro é uma variável que contém um endereço de memória
- Normalmente, um ponteiro não guarda qualquer endereço
  - o endereço de uma outra variável
  - um pedaço alocado de memória



# Declaração

- Ponteiros são declarados indicando o tipo dos valores para os quais você pretende “apontar” com ele e um asterisco antes do nome da variável

```
char   *pc;
int    *pi;
float  *pf;
double *pd;
void   *pv;
```

*Todos esses ponteiros guardam o mesmo tipo de coisa.*

*Todos possuem o mesmo tamanho*

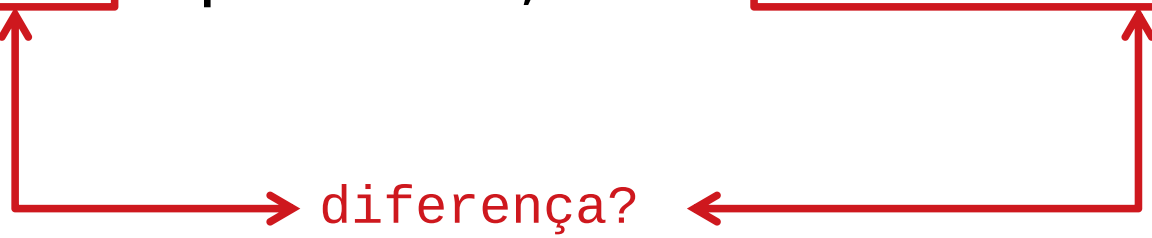
*Porém, suas diferenças ocorrem quando aritmética de ponteiros é usada ou seus dados que eles apontam são manipulados*

# Declaração

```
char  a = 'a'; //0110 0001₂ = 97₁₀
int   b = 5; //0000 0000 0000 0000 0000 0000 0000 0101

char *pa = &a;
int  *pb = &b;
```

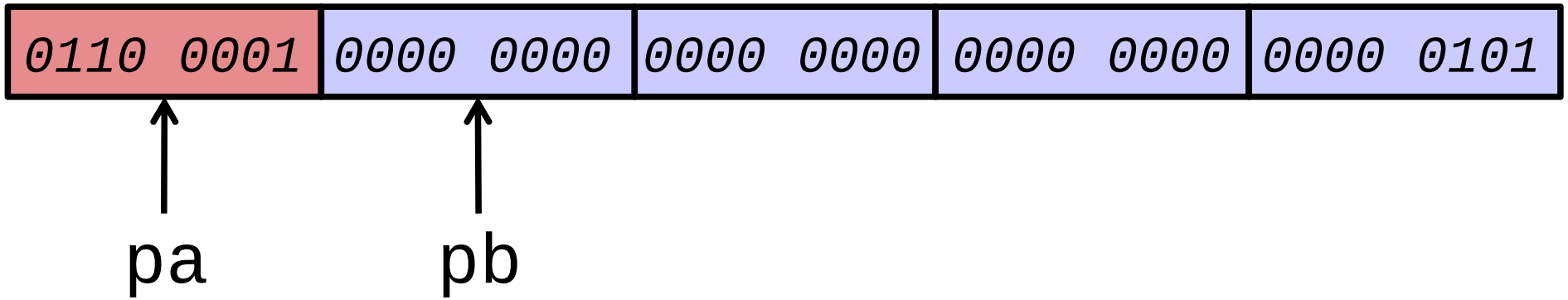
# Declaração

```
char   a = 'a'; //0110 0001₂ = 97₁₀
int    b = 5; //0000 0000 0000 0000 0000 0000 0000 0101

char  *pa = &a;      //0x7ffdf6934943
int   *pb = &b;      //0x7ffdf6934944
```



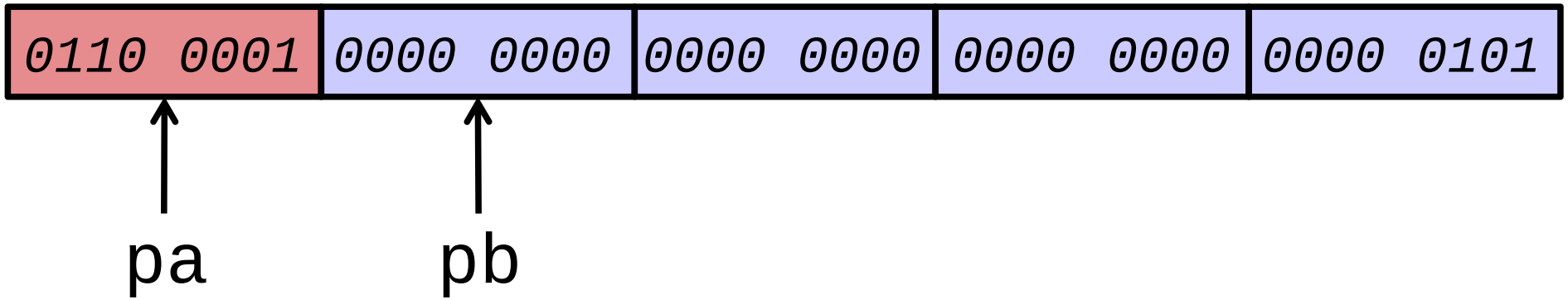
# Acesso aos dados

```
char  a = 'a'; //0110 0001₂ = 97₁₀
int   b = 5; //0000 0000 0000 0000 0000 0000 0000 0101

char *pa = &a;              //0x7ffdf6934943
int  *pb = &b;              //0x7ffdf6934944
```



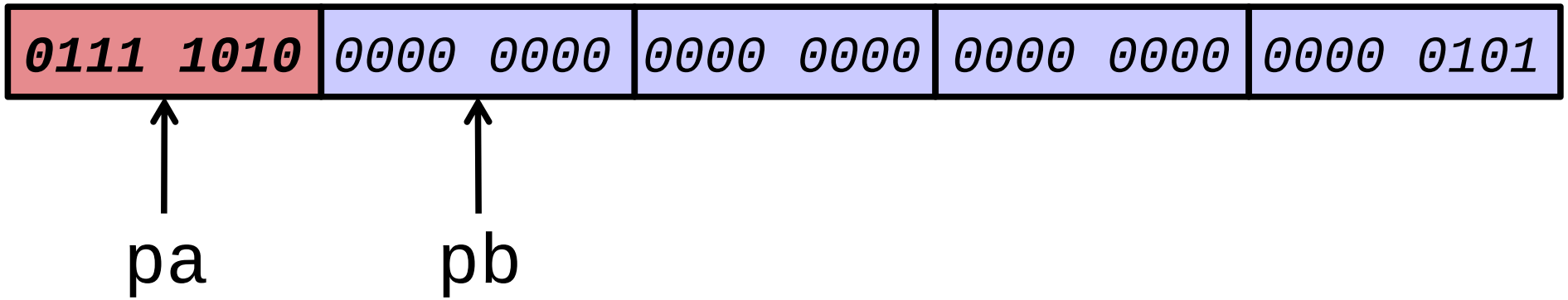
# Acesso aos dados

```
char   a = 'a'; //0110 0001₂ = 97₁₀
int    b = 5; //0000 0000 0000 0000 0000 0000 0000 0101

void *pa = &a;              //0x7ffdf6934943
void *pb = &b;              //0x7ffdf6934944

*pa = 'z';  <-------  error: invalid use of void expression
```



# Acesso aos dados

```
char  a = 'a'; //0110 0001₂ = 97₁₀
int   b = 5; //0000 0000 0000 0000 0000 0000 0000 0101

char *pa = &a;              //0x7ffdf6934943
void *pb = &b;              //0x7ffdf6934944

*pa = 'z';                  //0111 1010₂ = 122₁₀
```



# Operadores de referenciação

- O operador unário & devolve o endereço de memória do seu operando
- O operador unário * devolve o valor da variável localizada no endereço que o segue

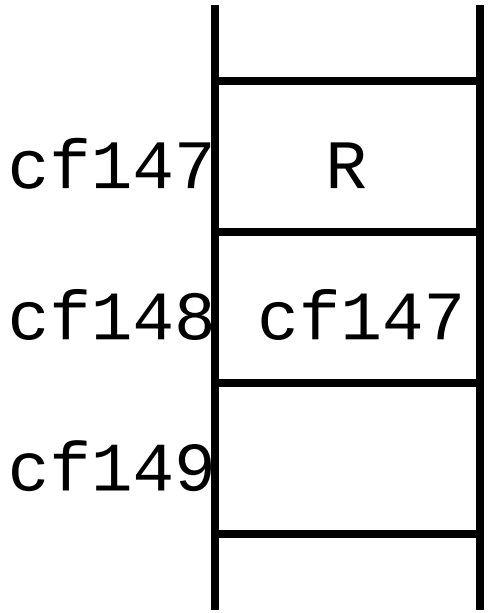
```c
#include <stdio.h>

int main(){
    char  a = 'R';
    char *p;

    p = &a;

    printf("loc(a) = %p, val(a) = %c\n" , &a , a);
    printf("loc(p) = %p, val(p) = %p, pnt(p) = %c\n", &p , p , *p);

    return 0;
}
```

*Saída*

```
loc(a) = 0x7ffdf41cf147, val(a) = R
loc(p) = 0x7ffdf41cf148, val(p) = 0x7ffdf41cf147, pnt(p) = R
```

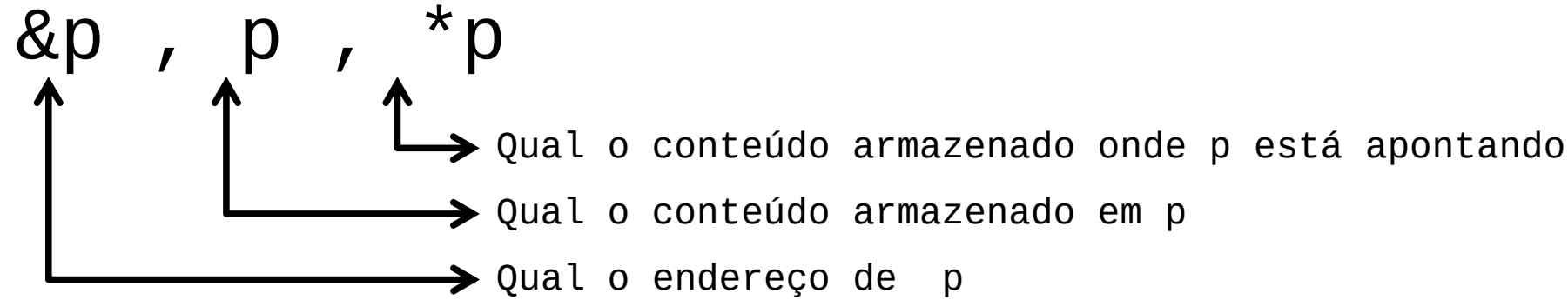
# Operadores de referenciação

- O operador unário & devolve o endereço de memória do seu operando
- O operador unário * devolve o valor da variável localizada no endereço que o segue




*Saída*

```
loc(a) = 0x7ffdf41cf147, val(a) = R
loc(p) = 0x7ffdf41cf148, val(p) = 0x7ffdf41cf147, pnt(p) = R
```



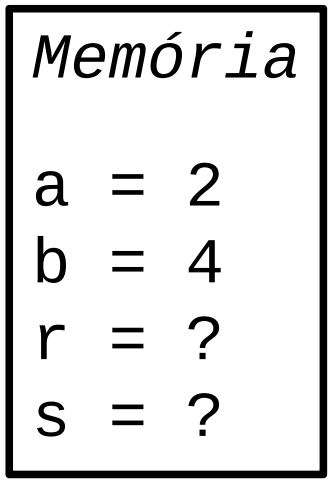
# Atribuição de Ponteiros

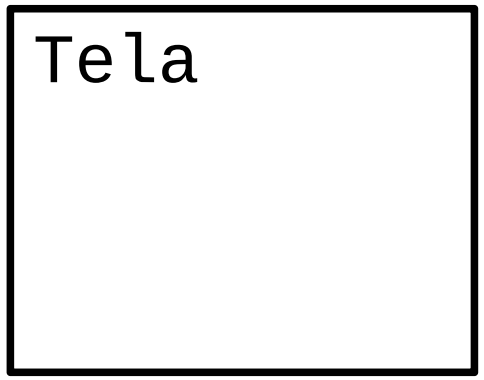
- Atribuição

```c
#include <stdio.h>

int main(){
    int a = 2, b = 4;
    int *r, *s;

    r = &a;
    *r = 5;
    printf("a = %d, b = %d\n",a,b);
    s = r;
    *s = *s + 1;
    printf("a = %d, b = %d\n",a,b);
    s = &b;
    *s = *r + 1;
    printf("a = %d, b = %d\n",a,b);

    return 0;
}
```

# Atribuição de Ponteiros

- Atribuição

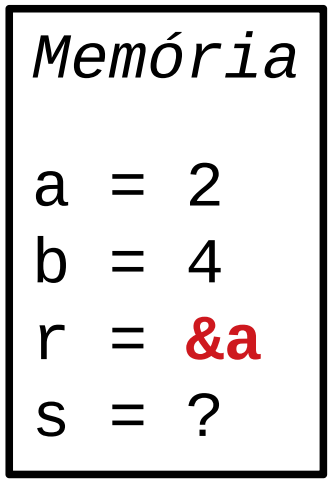


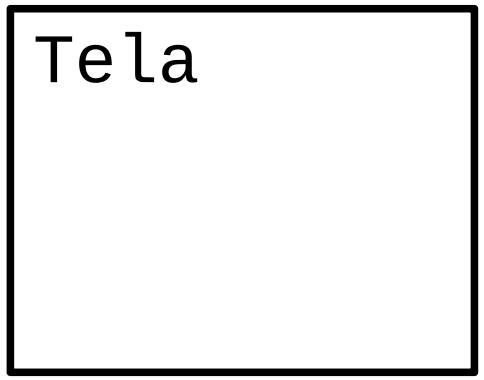




```c
#include <stdio.h>

int main(){
    int a = 2, b = 4;
    int *r, *s;

    r = &a;
    *r = 5;
    printf("a = %d, b = %d\n",a,b);
    s = r;
    *s = *s + 1;
    printf("a = %d, b = %d\n",a,b);
    s = &b;
    *s = *r + 1;
    printf("a = %d, b = %d\n",a,b);

    return 0;
}
```

# Atribuição de Ponteiros

- Atribuição

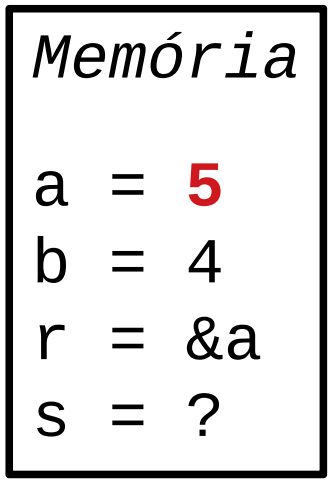


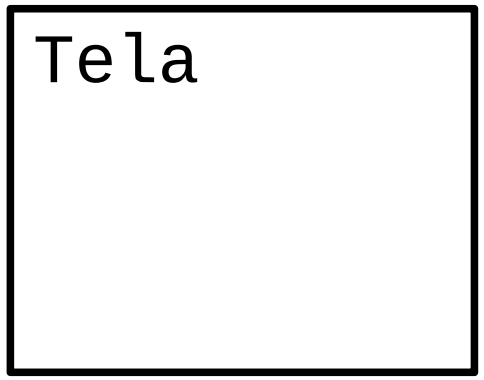




```c
#include <stdio.h>

int main(){
    int a = 2, b = 4;
    int *r, *s;

    r = &a;
    *r = 5;
    printf("a = %d, b = %d\n",a,b);
    s = r;
    *s = *s + 1;
    printf("a = %d, b = %d\n",a,b);
    s = &b;
    *s = *r + 1;
    printf("a = %d, b = %d\n",a,b);

    return 0;
}
```

# Atribuição de Ponteiros

- Atribuição




```c
#include <stdio.h>

int main(){
    int a = 2, b = 4;
    int *r, *s;

    r = &a;
    *r = 5;
    printf("a = %d, b = %d\n",a,b);
    s = r;
    *s = *s + 1;
    printf("a = %d, b = %d\n",a,b);
    s = &b;
    *s = *r + 1;
    printf("a = %d, b = %d\n",a,b);

    return 0;
}
```

# Atribuição de Ponteiros

- Atribuição

*Memória*

a = 5
b = 4
r = &a
s = ?

Tela

**a = 5, b = 4**

```c
#include <stdio.h>

int main(){
    int a = 2, b = 4;
    int *r, *s;

    r = &a;
    *r = 5;
    printf("a = %d, b = %d\n",a,b);
    s = r;
    *s = *s + 1;
    printf("a = %d, b = %d\n",a,b);
    s = &b;
    *s = *r + 1;
    printf("a = %d, b = %d\n",a,b);

    return 0;
}
```

# Atribuição de Ponteiros

- Atribuição

*Memória*

a = 5
b = 4
r = &a
s = **&a**

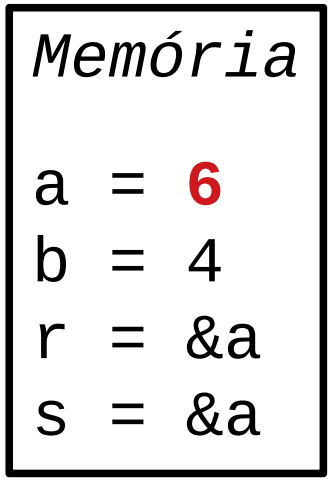
Tela

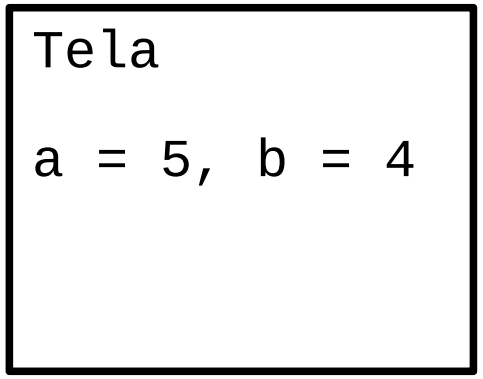
a = 5, b = 4

```c
#include <stdio.h>

int main(){
    int a = 2, b = 4;
    int *r, *s;

    r = &a;
    *r = 5;
    printf("a = %d, b = %d\n",a,b);
    s = r;
    *s = *s + 1;
    printf("a = %d, b = %d\n",a,b);
    s = &b;
    *s = *r + 1;
    printf("a = %d, b = %d\n",a,b);

    return 0;
}
```

# Atribuição de Ponteiros

- Atribuição

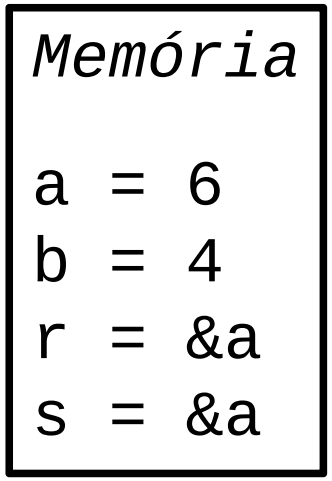


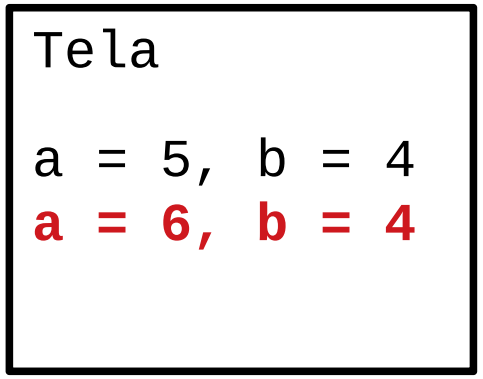




```c
#include <stdio.h>

int main(){
    int a = 2, b = 4;
    int *r, *s;

    r = &a;
    *r = 5;
    printf("a = %d, b = %d\n",a,b);
    s = r;
    *s = *s + 1;
    printf("a = %d, b = %d\n",a,b);
    s = &b;
    *s = *r + 1;
    printf("a = %d, b = %d\n",a,b);

    return 0;
}
```

# Atribuição de Ponteiros

- Atribuição




```c
#include <stdio.h>

int main(){
    int a = 2, b = 4;
    int *r, *s;

    r = &a;
    *r = 5;
    printf("a = %d, b = %d\n",a,b);
    s = r;
    *s = *s + 1;
    printf("a = %d, b = %d\n",a,b);
    s = &b;
    *s = *r + 1;
    printf("a = %d, b = %d\n",a,b);

    return 0;
}
```

# Atribuição de Ponteiros

- Atribuição

*Memória*

a = 6
b = 4
r = &a
s = **&b**

Tela

a = 5, b = 4
a = 6, b = 4

```c
#include <stdio.h>

int main(){
    int a = 2, b = 4;
    int *r, *s;

    r = &a;
    *r = 5;
    printf("a = %d, b = %d\n",a,b);
    s = r;
    *s = *s + 1;
    printf("a = %d, b = %d\n",a,b);
    s = &b;
    *s = *r + 1;
    printf("a = %d, b = %d\n",a,b);

    return 0;
}
```

# Atribuição de Ponteiros

- Atribuição

*Memória*

a = 6
b = **7**
r = &a
s = &b

Tela

a = 5, b = 4
a = 6, b = 4

```c
#include <stdio.h>

int main(){
    int a = 2, b = 4;
    int *r, *s;

    r = &a;
    *r = 5;
    printf("a = %d, b = %d\n",a,b);
    s = r;
    *s = *s + 1;
    printf("a = %d, b = %d\n",a,b);
    s = &b;
    *s = *r + 1;
    printf("a = %d, b = %d\n",a,b);

    return 0;
}
```

# Atribuição de Ponteiros

- Atribuição

*Memória*

a = 6
b = 7
r = &a
s = &b

Tela

a = 5, b = 4
a = 6, b = 4
**a = 6, b = 7**

Uma variável comum só consegue ver e modificar seu próprio espaço reservado

Um ponteiro pode viajar entre várias e várias variáveis vendo e alterando seu conteúdo

```c
#include <stdio.h>

int main(){
    int a = 2, b = 4;
    int *r, *s;

    r = &a;
    *r = 5;
    printf("a = %d, b = %d\n",a,b);
    s = r;
    *s = *s + 1;
    printf("a = %d, b = %d\n",a,b);
    s = &b;
    *s = *r + 1;
    printf("a = %d, b = %d\n",a,b);

    return 0;
}
```

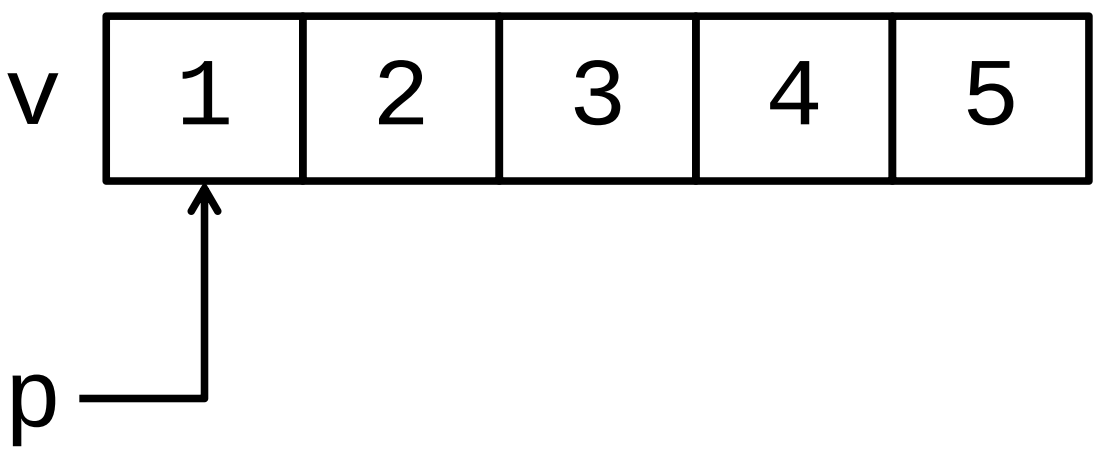
# Ponteiros e Matrizes

- Matrizes (vetores) são ponteiros!

```
int v[] = {1,2,3,4,5};
int *p;

p = v;
```

# Ponteiros e Matrizes

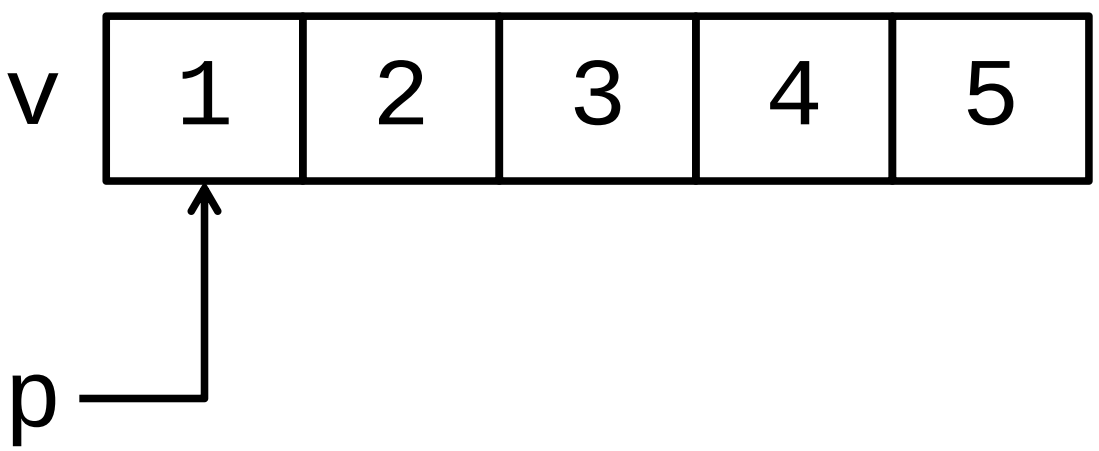
- Matrizes (vetores) são ponteiros!




```
int v[] = {1,2,3,4,5};
int *p;

p = v;
```

# Ponteiros e Matrizes

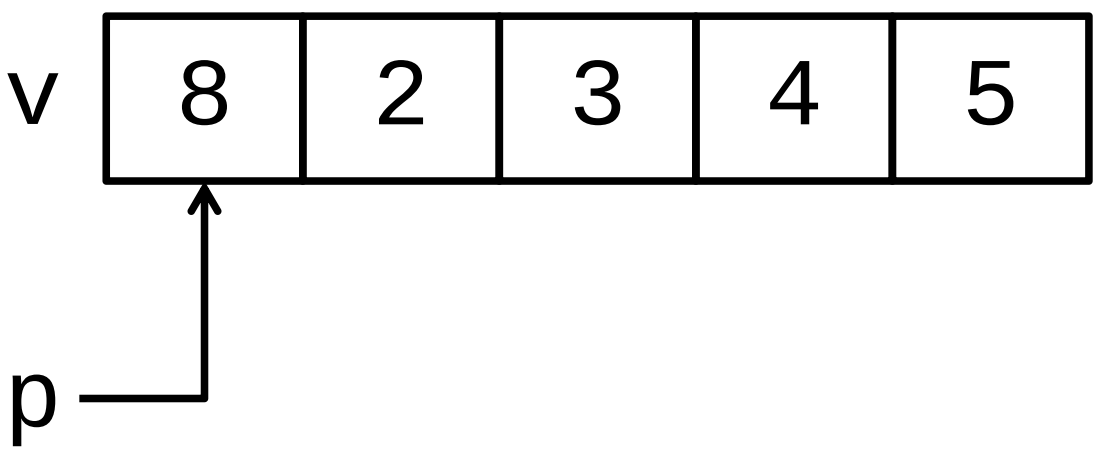
- Matrizes (vetores) são ponteiros!




```
int v[] = {1,2,3,4,5};
int *p;

p = v;

*p = 8;
```

# Ponteiros e Matrizes

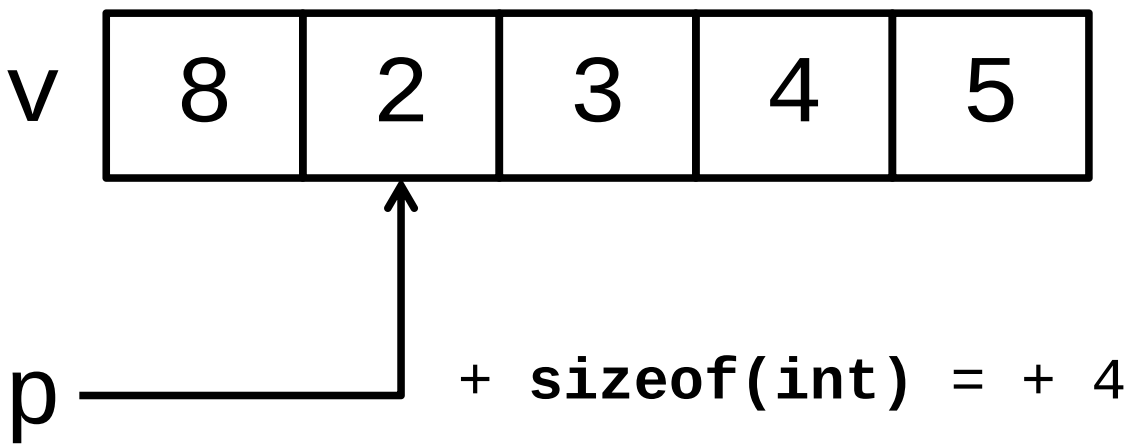
- Matrizes (vetores) são ponteiros!




```
int v[] = {1,2,3,4,5};
int *p;

p = v;

*p = 8;
```

# Ponteiros e Matrizes

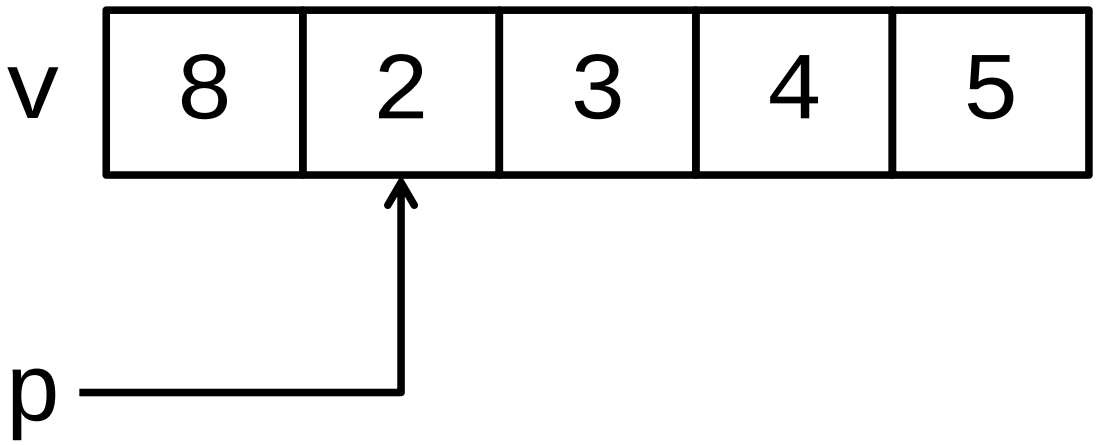
- Matrizes (vetores) são ponteiros!




```
int v[] = {1,2,3,4,5};
int *p;

p = v;

*p = 8;
```

# Ponteiros e Matrizes

- Matrizes (vetores) são ponteiros!



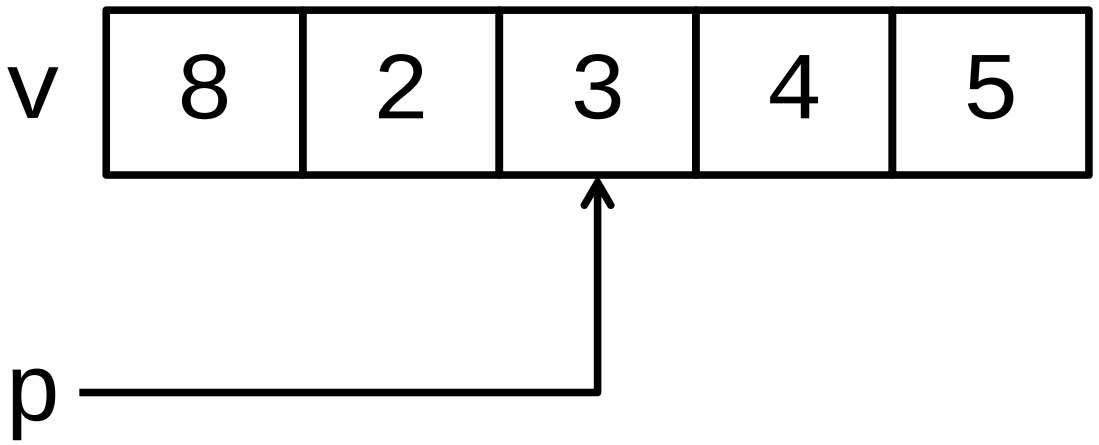

```
int v[] = {1,2,3,4,5};
int *p;

p = v;

*p = 8;
```

```
p = p + 1;
```

Some ao endereço p, o tamanho de 1 inteiro.

# Ponteiros e Matrizes

- Matrizes (vetores) são ponteiros!



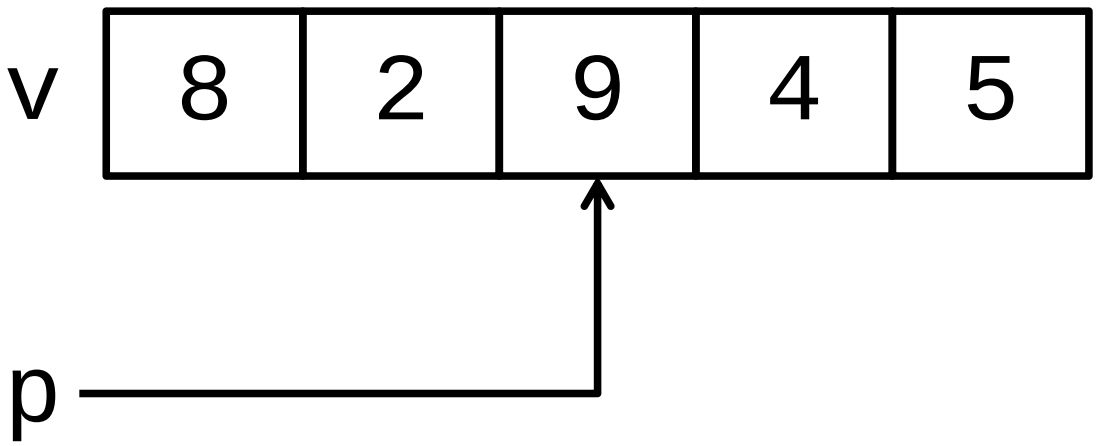

```
int v[] = {1,2,3,4,5};
int *p;

p = v;

*p = 8;
```

```
p = p + 2;
```

Some ao endereço p, o tamanho de 2 inteiros.

# Ponteiros e Matrizes

- Matrizes (vetores) são ponteiros!




```
int v[] = {1,2,3,4,5};
int *p;

p = v;

*p = 8;

p = p + 2;

*p = 9;
```

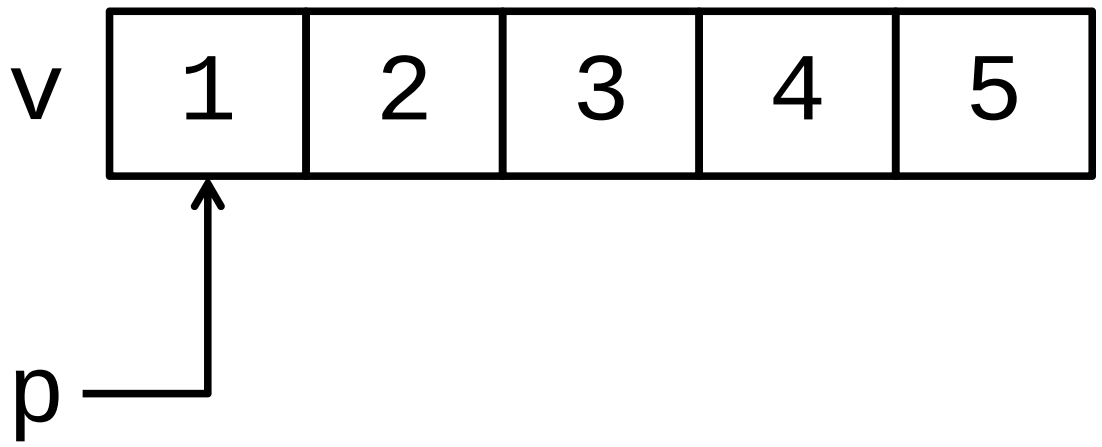
```
v[2] = 9;
```

↔

```
*(v+2) = 9;
```

# Operadores Aritméticos

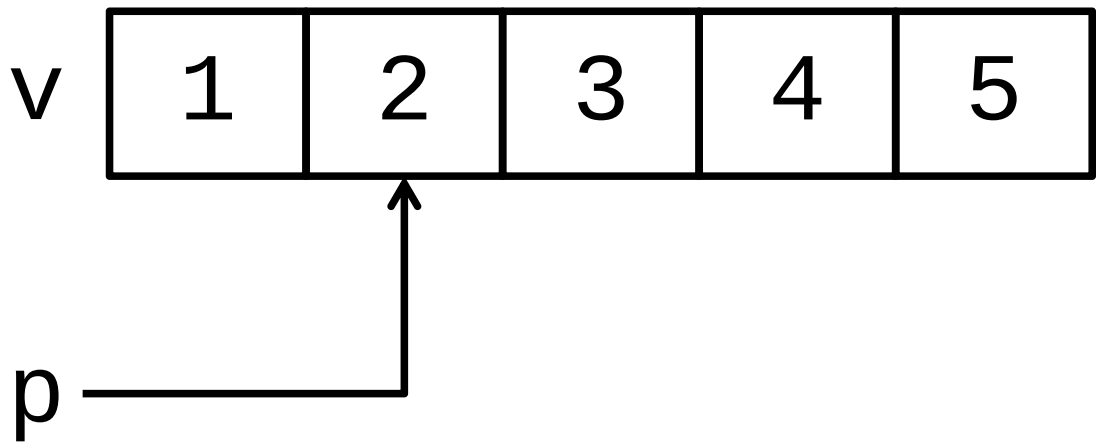
- Apenas operadores de soma e subtração entre ponteiros e inteiros pode ser usada




```c
int v[] = {1,2,3,4,5};
int *p;

p = v;
```

# Operadores Aritméticos

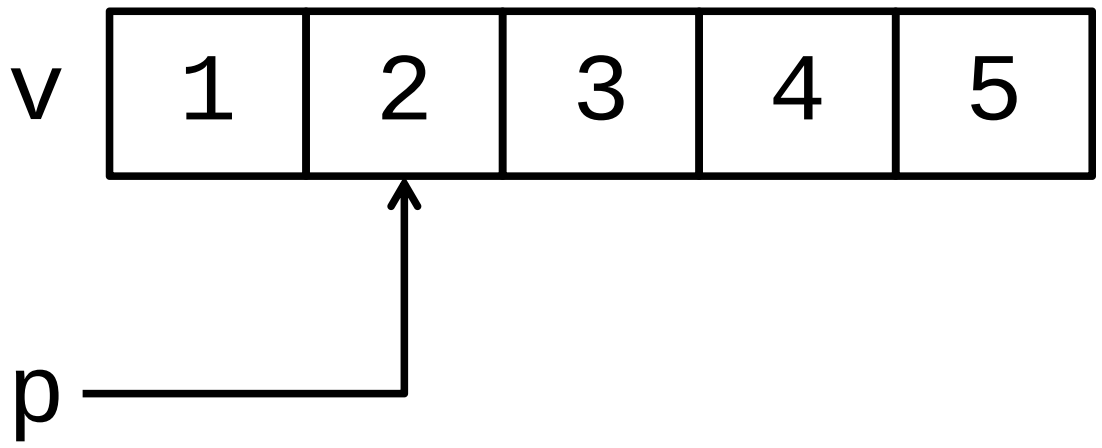
- Apenas operadores de soma e subtração entre ponteiros e inteiros pode ser usada




```
int v[] = {1,2,3,4,5};
int *p;

p = v;

p = p + 1;
```

# Operadores Aritméticos

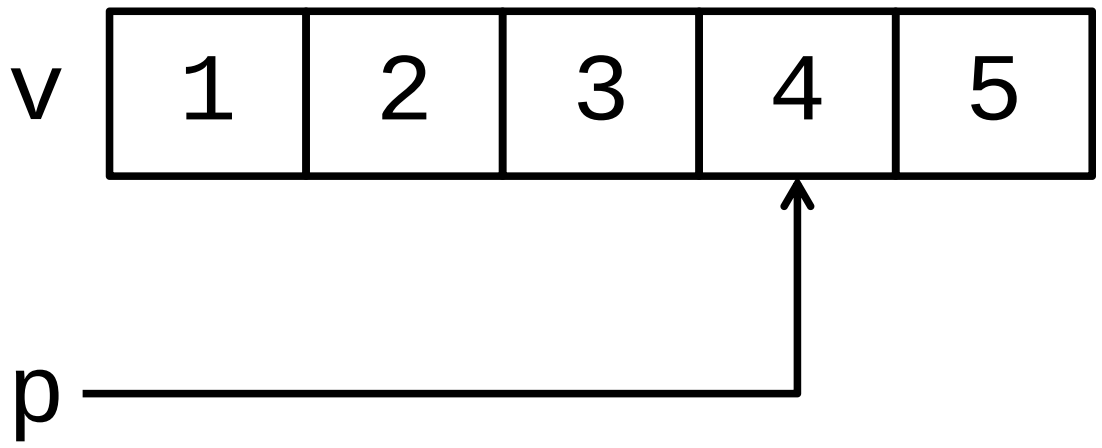
- Apenas operadores de soma e subtração entre ponteiros e inteiros pode ser usada




```
int v[] = {1,2,3,4,5};
int *p;

p = v;

p++;
```

# Operadores Aritméticos

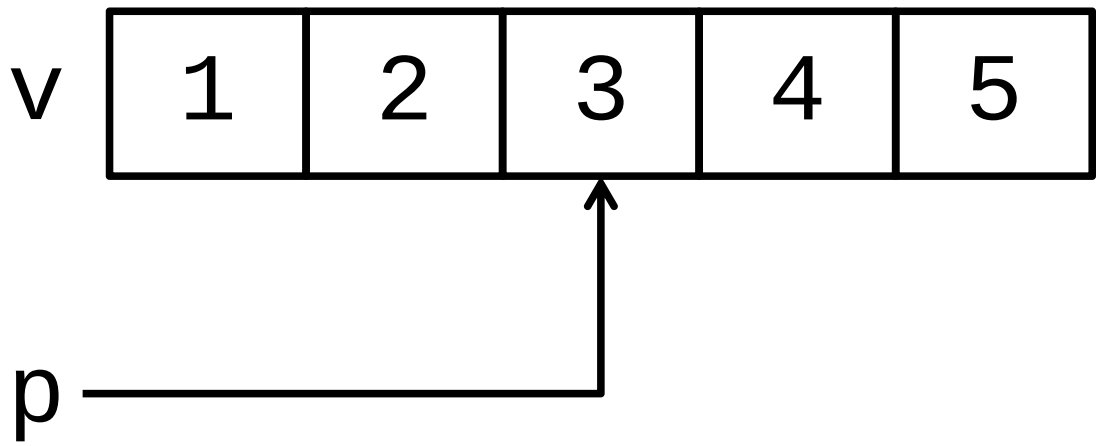
- Apenas operadores de soma e subtração entre ponteiros e inteiros pode ser usada




```
int v[] = {1,2,3,4,5};
int *p;

p = v;

p += 3;
```

# Operadores Aritméticos

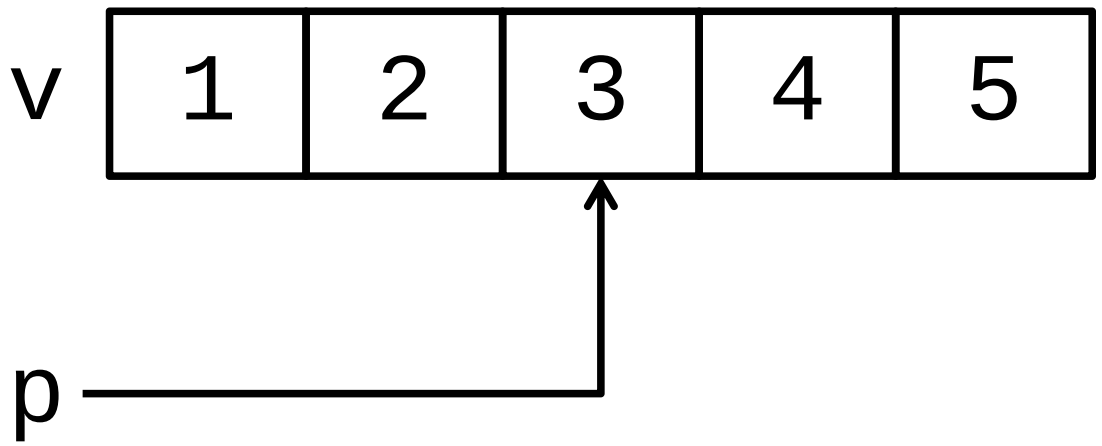
- Apenas operadores de soma e subtração entre ponteiros e inteiros pode ser usada




```
int v[] = {1,2,3,4,5};
int *p;

p = v;

p += 3;

p = p - 1;
```

# Operadores Aritméticos

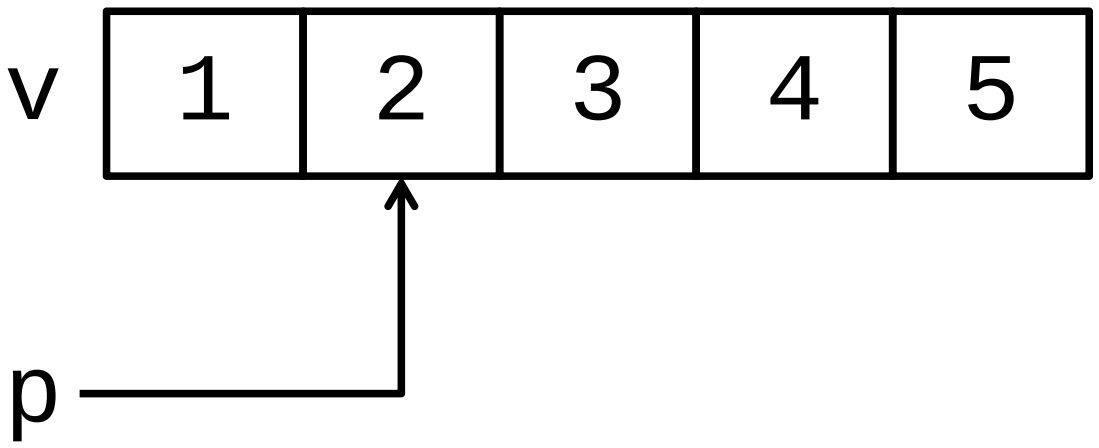
- Apenas operadores de soma e subtração entre ponteiros e inteiros pode ser usada




```
int v[] = {1,2,3,4,5};
int *p;

p = v;

p += 3;

p--;
```

# Operadores Aritméticos

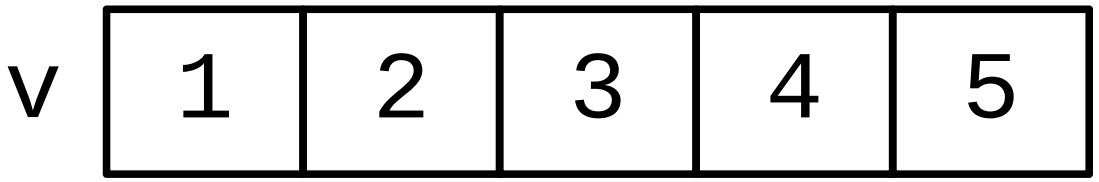
- Apenas operadores de soma e subtração entre ponteiros e inteiros pode ser usada




```
int v[] = {1,2,3,4,5};
int *p;

p = v;

p += 3;

p -= 2;
```

Os operadores de multiplicação, divisão e resto da divisão não são definidos para ponteiros.

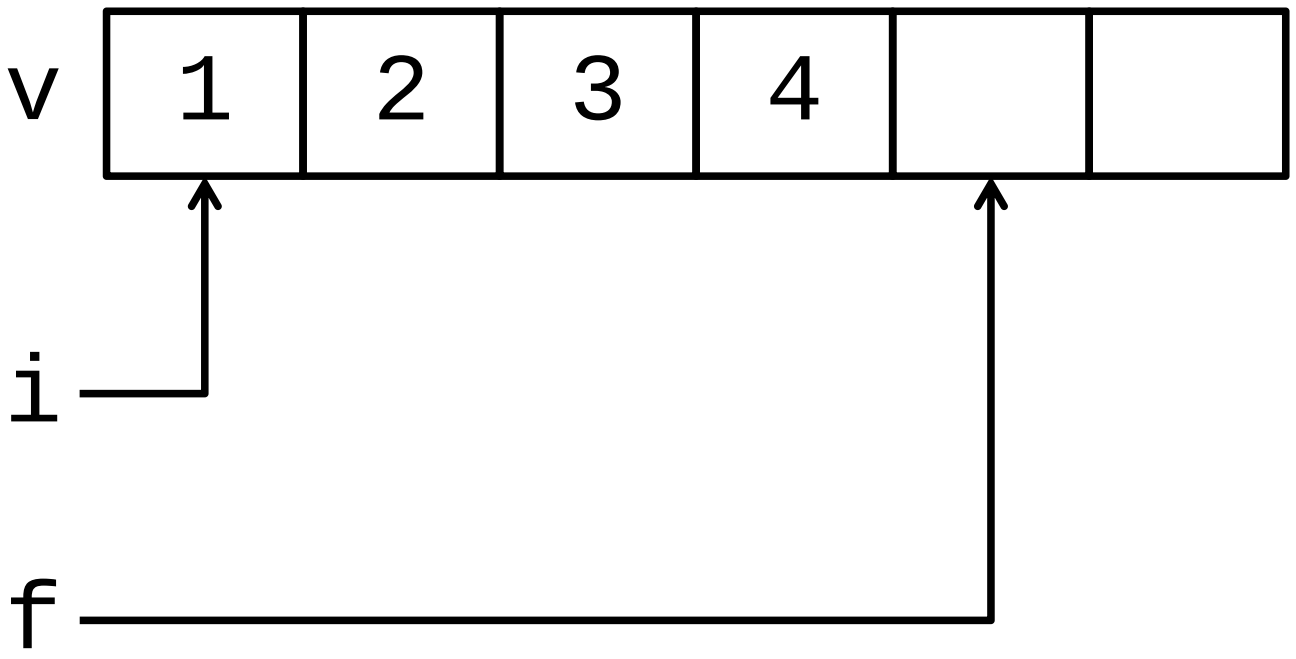
# Operadores de Comparação




```c
int v[6] = {1,2,3,4};
int n = 4;
int i;

for(i = 0 ; i < n ; i++)
    printf("%d, ",v[i]);
```

# Operadores de Comparação

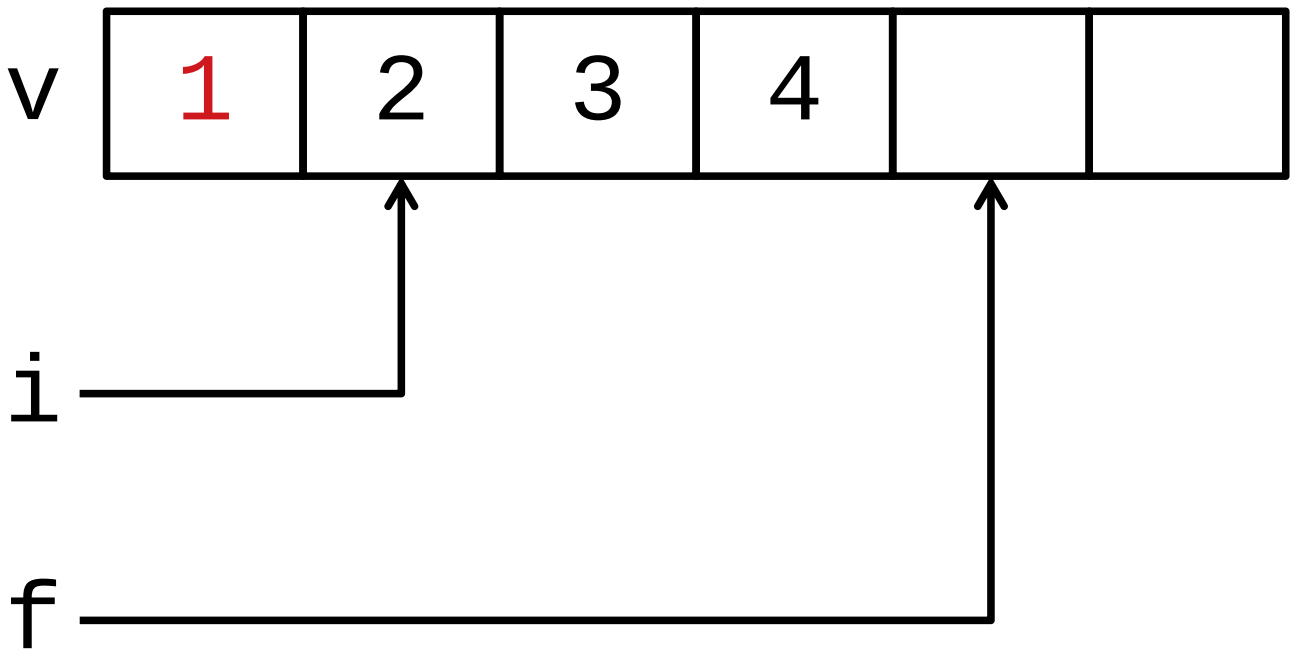
- Como um vetor é alocado sequencialmente, podemos comparar vetores e saber quem está antes ou depois




```
int v[6] = {1,2,3,4};
int *f = &v[4];
int *i;

for(i = v ; i < f ; i++)
    printf("%d, ",*i);
```

# Operadores de Comparação

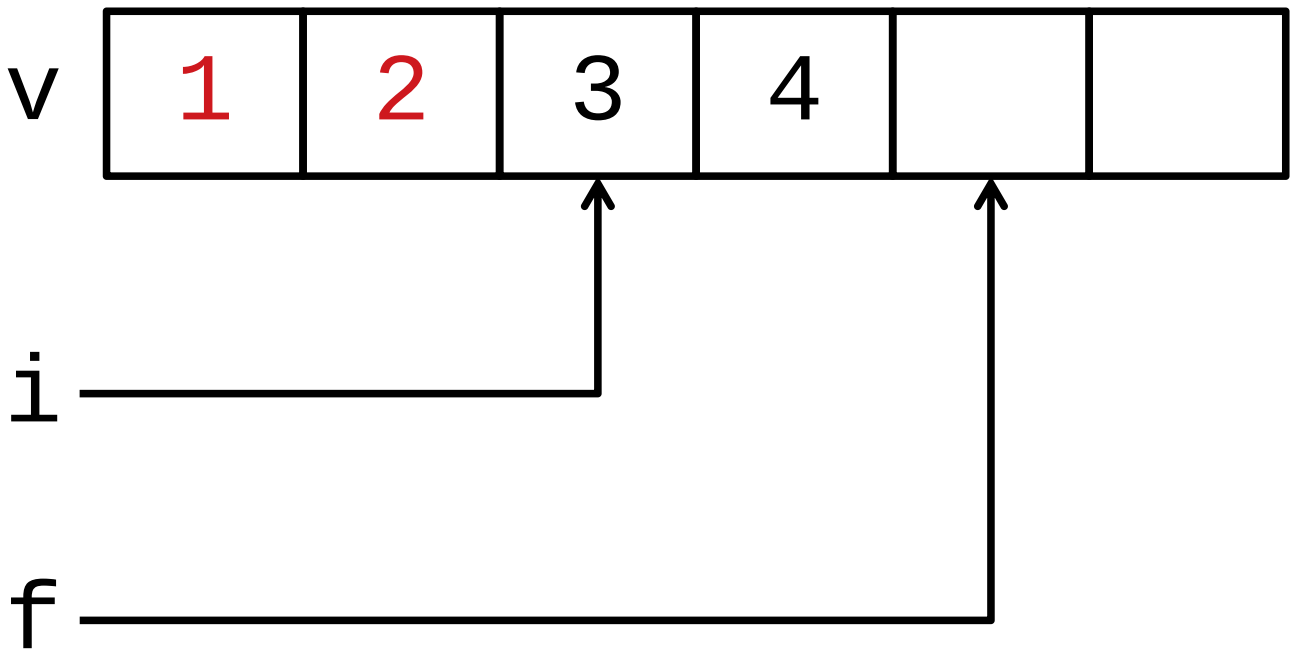
- Como um vetor é alocado sequencialmente, podemos comparar vetores e saber quem está antes ou depois




```
int v[6] = {1,2,3,4};
int *f = &v[4];
int *i;

for(i = v ; i < f ; i++)
    printf("%d, ",*i);
```

# Operadores de Comparação

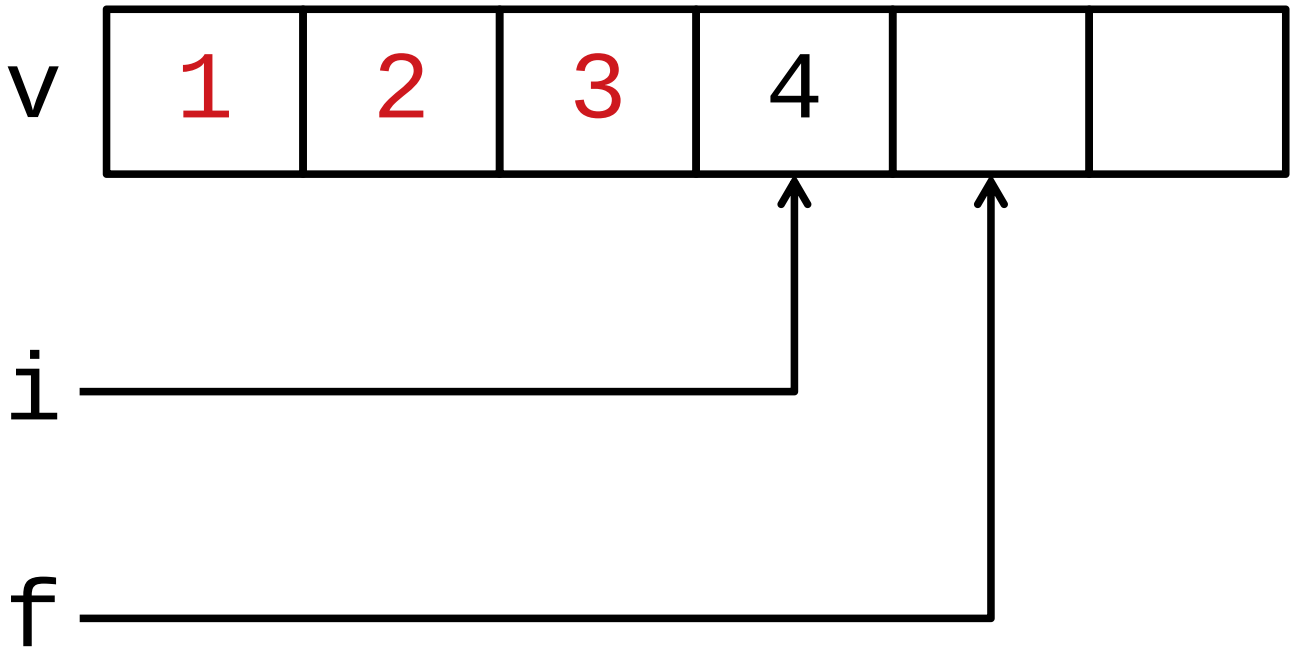
- Como um vetor é alocado sequencialmente, podemos comparar vetores e saber quem está antes ou depois




```
int v[6] = {1,2,3,4};
int *f = &v[4];
int *i;

for(i = v ; i < f ; i++)
    printf("%d, ",*i);
```

# Operadores de Comparação

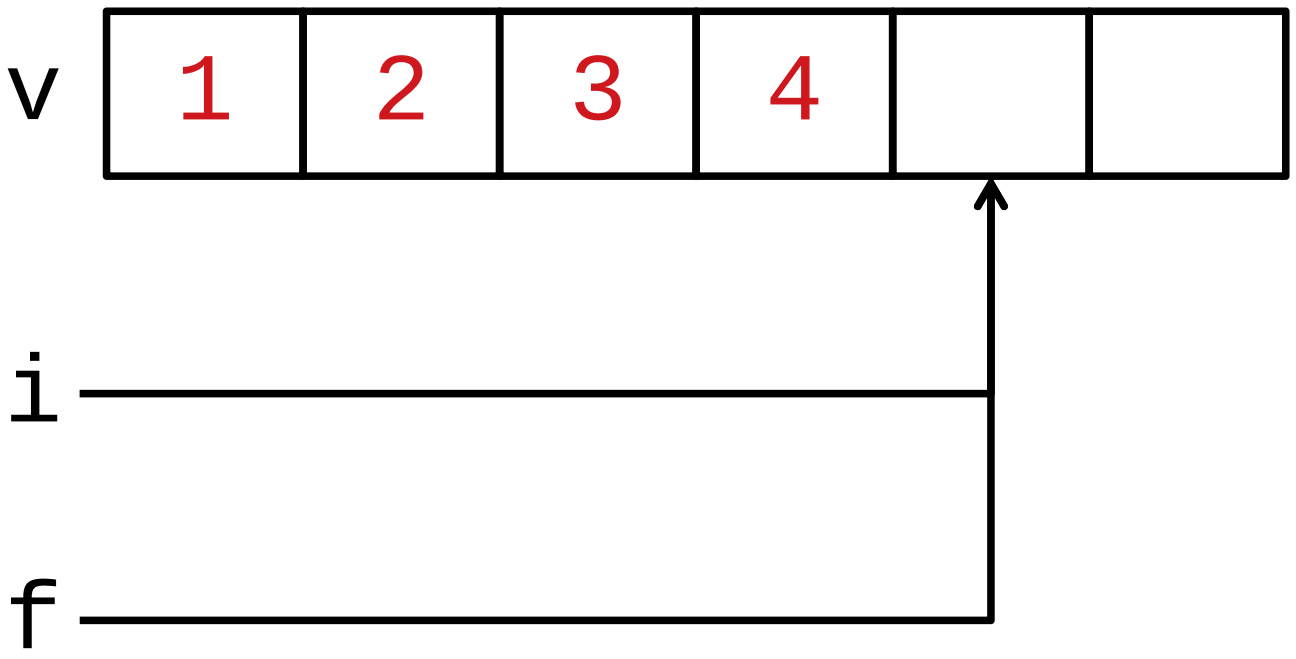
- Como um vetor é alocado sequencialmente, podemos comparar vetores e saber quem está antes ou depois




```
int v[6] = {1,2,3,4};
int *f = &v[4];
int *i;

for(i = v ; i < f ; i++)
    printf("%d, ",*i);
```

# Operadores de Comparação

- Como um vetor é alocado sequencialmente, podemos comparar vetores e saber quem está antes ou depois




```
int v[6] = {1,2,3,4};
int *f = &v[4];
int *i;

for(i = v ; i < f ; i++)
    printf("%d, ",*i);
```

# Exercícios

- Escreva uma função que declara variáveis do tipo inteiro, real e caractere e ponteiros para esses tipos. Associe os ponteiros às variáveis e altere seus valores através dos ponteiros. Por fim, imprima os valores das variáveis originais.
- Crie uma função que preenche um vetor de 5 posições com valores digitados pelo usuário. Restrição: não pode usar o operador colchete para acessar os elementos do vetor.
- Calcule o comprimento de uma string usando ponteiros e sua aritmética. (Dica: subtrair dois ponteiros retorna a quantidade de elementos do tipo do ponteiro que podem caber entre seus endereços)
- Escreva um programa que escreve uma string de trás pra frente usando ponteiros.