

# Revisão de Ponteiros

## (Linguagem C)

André Tavares da Silva

[andre.silva@udesc.br](mailto:andre.silva@udesc.br)

Baseado no Material do prof. Rui Tramontin Junior

# Introdução

- Em C, ponteiros e vetores têm uma relação muito próxima;
- O identificador de um vetor representa o endereço de memória da 1ª posição do vetor;
- Um vetor pode ser visto como um ponteiro imutável, ou seja, pode ser acessado, mas não pode ser modificado.

# Exemplo 1: vetor é ponteiro?

```
int v[5] = {2, 4, 6, 8, 10};

printf("Endereço de V: %d\n", v );
int i;
for( i = 0; i < 5 ; i++ ){
    printf("V[%d] ", i);
    printf("( %d) ", &v[i]);
    printf("= %d\n", v[i]);
}
```

# Exemplo 1: vetor é ponteiro?

```
int v[5] = {2, 4, 6, 8, 10};
```

```
printf("Endereço de V: %d\n", v );  
int i;  
for( i = 0; i < 5 ; i++ ){  
    printf("V[%d] ", i);  
    printf("( %d) ", &v[i]);  
    printf("= %d\n", v[i]);  
}
```

## Modelo da Memória

v[0]	2	1000
v[1]	4	1004
v[2]	6	1008
v[3]	8	1012
v[4]	10	1016

# Exemplo 1: vetor é ponteiro?

```
int v[5] = {2, 4, 6, 8, 10};
```

```
printf("Endereço de V: %d\n", v );
```

```
int i;
```

```
for( i = 0; i < 5 ; i++ ){
```

```
    printf("V[%d] ", i);
```

```
    printf("( %d) ", &v[i]);
```

```
    printf("= %d\n", v[i]);
```

```
}
```

## Modelo da Memória

v[0]	2	1000
v[1]	4	1004
v[2]	6	1008
v[3]	8	1012
v[4]	10	1016

# Exemplo 1: vetor é ponteiro?

```
int v[5] = {2, 4, 6, 8, 10};
```

```
printf("Endereço de V: %d\n", v );
```

```
int i;
```

```
for( i = 0; i < 5 ; i++ ){
```

```
    printf("V[%d] ", i);
```

```
    printf("( %d) ", &v[i]);
```

```
    printf("= %d\n", v[i]);
```

```
}
```

## Modelo da Memória

v[0]	2	1000
v[1]	4	1004
v[2]	6	1008
v[3]	8	1012
v[4]	10	1016

Endereço de V: 1000

# Exemplo 1: vetor é ponteiro?

```
int v[5] = {2, 4, 6, 8, 10};
```

```
printf("Endereço de V: %d\n", v );
```

```
int i;
```

```
for( i = 0; i < 5 ; i++ ){  
    printf("V[%d] ", i);  
    printf("( %d) ", &v[i]);  
    printf("= %d\n", v[i]);  
}
```

## Modelo da Memória

v[0]	2	1000
v[1]	4	1004
v[2]	6	1008
v[3]	8	1012
v[4]	10	1016

Endereço de V: 1000

# Exemplo 1: vetor é ponteiro?

```
int v[5] = {2, 4, 6, 8, 10};
```

```
printf("Endereço de V: %d\n", v );
```

```
int i;
```

```
for( i = 0; i < 5 ; i++ ){  
    printf("V[%d] ", i);  
    printf("( %d) ", &v[i]);  
    printf("= %d\n", v[i]);  
}
```

## Modelo da Memória

v[0]	2	1000
v[1]	4	1004
v[2]	6	1008
v[3]	8	1012
v[4]	10	1016

Endereço de V: 1000

V[0] (1000) = 2

V[1] (1004) = 4

V[2] (1008) = 6

V[3] (1012) = 8

V[4] (1016) = 10



# Exemplo 1: vetor é ponteiro?

```
int v[5] = {2, 4, 6, 8, 10};
```

```
printf("Endereço de V: %d\n", v );
```

```
int i;
```

```
for( i = 0; i < 5 ; i++ ){
```

```
    printf("V[%d] ", i);
```

```
    printf("( %d) ", &v[i]);
```

```
    printf("= %d\n", v[i]);
```

```
}
```

## Modelo da Memória

v[0]	2	1000
v[1]	4	1004
v[2]	6	1008
v[3]	8	1012
v[4]	10	1016

Endereço de V: 1000

V[0] (1000) = 2  
V[1] (1004) = 4  
V[2] (1008) = 6  
V[3] (1012) = 8  
V[4] (1016) = 10

# Considerações

- Com base no exemplo, vemos as seguintes equivalências:

**v** <-> **&v** <-> **&v[0]**

## **EXEMPLO 2: ACESSANDO VETOR COM PONTEIRO**

## Exemplo 2: acessando vetor com ponteiro

```
int v[5];
```

## Exemplo 2: acessando vetor com ponteiro

```
int v[5];
```

### **Modelo da Memória**

v[0]		1000
v[1]		1004
v[2]		1008
v[3]		1012
v[4]		1016

## Exemplo 2: acessando vetor com ponteiro

```
int v[5];  
int *p = v;
```

### **Modelo da Memória**

v[0]		1000
v[1]		1004
v[2]		1008
v[3]		1012
v[4]		1016

## Exemplo 2: acessando vetor com ponteiro

```
int v[5];
```

```
int *p = v;
```

### **Modelo da Memória**

v[0]		1000
v[1]		1004
v[2]		1008
v[3]		1012
v[4]		1016

## Exemplo 2: acessando vetor com ponteiro

```
int v[5];
```

```
int *p = v;
```

### **Modelo da Memória**

v[0]		1000
v[1]		1004
v[2]		1008
v[3]		1012
v[4]		1016
p	1000	1020



## Exemplo 2: acessando vetor com ponteiro

```
int v[5];  
int *p = v; // &v ou &v[0]
```

### **Modelo da Memória**

v[0]		1000
v[1]		1004
v[2]		1008
v[3]		1012
v[4]		1016
p	1000	1020

## Exemplo 2: acessando vetor com ponteiro

```
int v[5];  
int *p = v; // &v ou &v[0]  
  
int i;  
for( i = 0; i < 5 ; i++ ){  
    scanf("%d", &p[i] );  
}
```

### **Modelo da Memória**

v[0]		1000
v[1]		1004
v[2]		1008
v[3]		1012
v[4]		1016
p	1000	1020

## Exemplo 2: acessando vetor com ponteiro

```
int v[5];  
int *p = v; // &v ou &v[0]  
  
int i;  
for( i = 0; i < 5 ; i++ ){  
    scanf("%d", &p[i] );  
} // Acessa v usando p.
```

### Modelo da Memória

v[0]		1000
v[1]		1004
v[2]		1008
v[3]		1012
v[4]		1016
p	1000	1020

## Exemplo 2: acessando vetor com ponteiro

```
int v[5];  
int *p = v; // &v ou &v[0]  
  
int i;  
for( i = 0; i < 5 ; i++ ){  
    scanf("%d", &p[i] );  
}  
  
for( i = 0; i < 5 ; i++ ){  
    printf("%d\n", v[i] );  
}
```

### **Modelo da Memória**

v[0]		1000
v[1]		1004
v[2]		1008
v[3]		1012
v[4]		1016
p	1000	1020

## Exemplo 2: acessando vetor com ponteiro

```
int v[5];  
int *p = v; // &v ou &v[0]
```

```
int i;  
for( i = 0; i < 5 ; i++ ){  
    scanf("%d", &p[i] );  
}
```

```
for( i = 0; i < 5 ; i++ ){  
    printf("%d\n", v[i] ); // Acessa v diretamente.  
}
```

### Modelo da Memória

v[0]		1000
v[1]		1004
v[2]		1008
v[3]		1012
v[4]		1016
p	1000	1020

# Considerações

- Ponteiros podem ser usados para acessar elementos de um vetor;
- Portanto, em C é possível utilizar a notação de colchetes ( `[]` ) com ponteiros;
- Na prática, o uso dos colchetes significa um acesso à memória a partir do endereço apontado pelo ponteiro;

# **EXEMPLO 3: PASSAGEM POR REFERÊNCIA**

## Exemplo 3: passagem por referência

```
int main() {  
    int v[] = {3, 6, 9, 12, 15};  
  
    return 0;  
}
```



## Exemplo 3: passagem por referência

```
int main() {  
    int v[] = {3, 6, 9, 12, 15};  
  
    return 0;  
}
```

### **Modelo da Memória**

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008
v[3]	12	1012
v[4]	15	1016

## Exemplo 3: passagem por referência

```
int main() {  
    int v[] = {3, 6, 9, 12, 15};  
    mostra_vetor( v , 5 );  
    return 0;  
}
```

### **Modelo da Memória**

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008
v[3]	12	1012
v[4]	15	1016



# Exemplo 3: passagem por referência

```
int main() {  
    int v[] = {3, 6, 9, 12, 15};  
    mostra_vetor( v , 5 );  
    return 0;  
}  
  
void mostra_vetor( int *p , int k ) {  
    int i;  
    for( i = 0; i < k ; i++ )  
        printf("%d\n", p[i]);  
}
```

## **Modelo da Memória**

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008
v[3]	12	1012
v[4]	15	1016

## Exemplo 3: passagem por referência

```
int main() {
    int v[] = {3, 6, 9, 12, 15};
    mostra_vetor( v , 5 );
    return 0;
}

void mostra_vetor( int *p , int k ) {
    int i;
    for( i = 0; i < k ; i++ )
        printf("%d\n", p[i]);
}
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008
v[3]	12	1012
v[4]	15	1016
p	1000	1020
k	5	1028
i		1032

## Exemplo 3: passagem por referência

```
int main() {  
    int v[] = {3, 6, 9, 12, 15};  
    mostra_vetor( v , 5 );  
    return 0;  
}  
  
void mostra_vetor( int *p , int k ) {  
    int i;  
    for( i = 0; i < k ; i++ )  
        printf("%d\n", p[i]);  
}
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008
v[3]	12	1012
v[4]	15	1016
p	1000	1020
k	5	1028
i		1032

## Exemplo 3: passagem por referência

```
int main() {  
    int v[] = {3, 6, 9, 12, 15};  
    mostra_vetor( v, 5 );  
    return 0;  
}  
  
void mostra_vetor( int *p, int k ) {  
    int i;  
    for( i = 0; i < k ; i++ )  
        printf("%d\n", p[i]);  
}  
  
// Vetores sempre são passados por referência!
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008
v[3]	12	1012
v[4]	15	1016
p	1000	1020
k	5	1028
i		1032

# Considerações

- Vetores sempre são passados por referência;
- Portanto, não importa a notação utilizada na declaração do parâmetro;
- Nesse contexto, elas são equivalentes:

```
int *p <-> int p[]
```

# Mais considerações...

- O uso de colchetes pode ser usado com um ponteiro mesmo que ele não aponte para um vetor;
- Neste caso, o acesso feito à posição 0 ( **[0]** ) é equivalente a usar o operador de indireção ( **\*** );



# Mais considerações...

- O uso de colchetes pode ser usado com um ponteiro mesmo que ele não aponte para um vetor;
- Neste caso, o acesso feito à posição 0 ( **[0]** ) é equivalente a usar o operador de indireção ( **\*** );
- Exemplo (incremento de variável):

```
void inc( int *x ){  
    (*x)++;  
}
```

# Mais considerações...

- O uso de colchetes pode ser usado com um ponteiro mesmo que ele não aponte para um vetor;
- Neste caso, o acesso feito à posição 0 ( **[0]** ) é equivalente a usar o operador de indireção ( **\*** );
- Exemplo (incremento de variável):

```
void inc( int *x ){  
    (*x)++;  
}
```

ou

```
void inc( int *x ){  
    x[0]++;  
}
```

# Ainda mais considerações...

- Portanto, o uso de índices com um ponteiro permite percorrer a memória;

# Ainda mais considerações...

- Portanto, o uso de índices com um ponteiro permite percorrer a memória;
  - Deve ser usado com cuidado!

# Ainda mais considerações...

- Portanto, o uso de índices com um ponteiro permite percorrer a memória;
  - Deve ser usado com cuidado!
- Por outro lado, é possível utilizar também o operador de indireção (  $*$  ) em vetores para acessar seus valores;

# Ainda mais considerações...

- Portanto, o uso de índices com um ponteiro permite percorrer a memória;
  - Deve ser usado com cuidado!
- Por outro lado, é possível utilizar também o operador de indireção ( \* ) em vetores para acessar seus valores;
- Mas como é possível acessar os índices além do 0?

# Ainda mais considerações...

- Portanto, o uso de índices com um ponteiro permite percorrer a memória;
  - Deve ser usado com cuidado!
- Por outro lado, é possível utilizar também o operador de indireção (  $*$  ) em vetores para acessar seus valores;
- Mas como é possível acessar os índices além do 0?

**Aritmética de Ponteiros!**

# Aritmética de Ponteiros

- Permite o cálculo de endereços de memória de duas formas:



# Aritmética de Ponteiros

- Permite o cálculo de endereços de memória de duas formas:
- Soma:

$\text{endereço} \pm \textit{valor escalar} = \text{novo endereço}$

# Aritmética de Ponteiros

- Permite o cálculo de endereços de memória de duas formas:

- Soma:

$\text{endereço} \pm \text{valor escalar} = \text{novo endereço}$

- Subtração:

$\text{endereço} - \text{endereço} = \text{valor escalar}$

# Aritmética de Ponteiros

- Soma: um endereço é somado a um valor inteiro (índice), gerando um novo endereço;

# Aritmética de Ponteiros

- Soma: um endereço é somado a um valor inteiro (índice), gerando um novo endereço;
- O resultado da soma depende do tamanho do tipo de dados (função `sizeof()` );

# Aritmética de Ponteiros

- Soma: um endereço é somado a um valor inteiro (índice), gerando um novo endereço;
- O resultado da soma depende do tamanho do tipo de dados (função `sizeof()` );

$\text{endereço} \pm \text{valor escalar} = \text{novo endereço}$

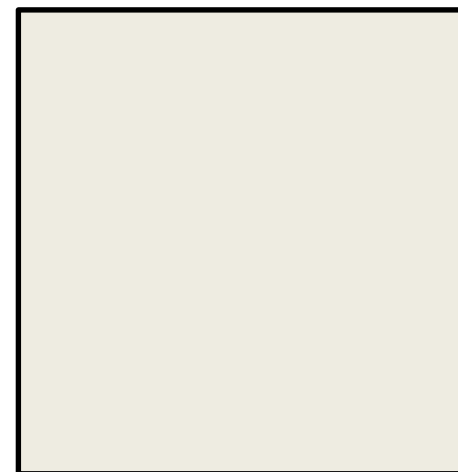
# **EXEMPLO 4: ARITMÉTICA DE PONTEIROS**

## Exemplo 4: aritmética de ponteiros

```
int v[3] = { 3, 6, 9 };  
printf("%d\n", v ); // v+0 ou &v[0]  
printf("%d\n", v + 1 ); // &v[1]  
printf("%d\n", v + 2 ); // &v[2]
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008



## Exemplo 4: aritmética de ponteiros

```
int v[3] = { 3, 6, 9 };  
printf("%d\n", v); // v+0 ou &v[0]  
printf("%d\n", v + 1); // &v[1]  
printf("%d\n", v + 2); // &v[2]
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008



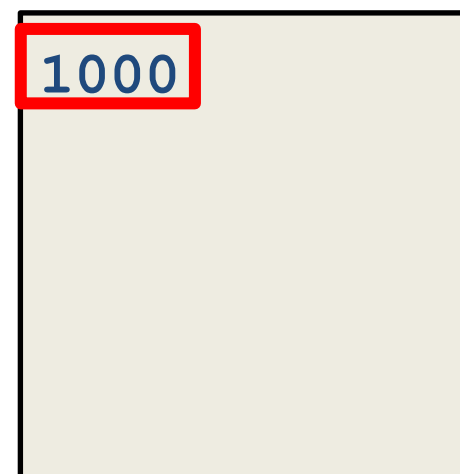


## Exemplo 4: aritmética de ponteiros

```
int v[3] = { 3, 6, 9 };  
printf("%d\n", v); // v+0 ou &v[0]  
printf("%d\n", v + 1); // &v[1]  
printf("%d\n", v + 2); // &v[2]
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008

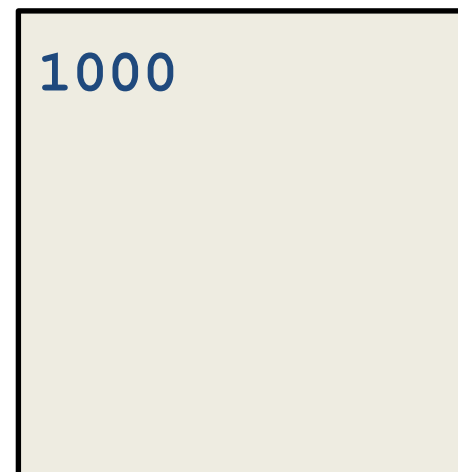


## Exemplo 4: aritmética de ponteiros

```
int v[3] = { 3, 6, 9 };  
printf("%d\n", v ); // v+0 ou &v[0]  
printf("%d\n", v + 1 ); // &v[1]  
printf("%d\n", v + 2 ); // &v[2]
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008



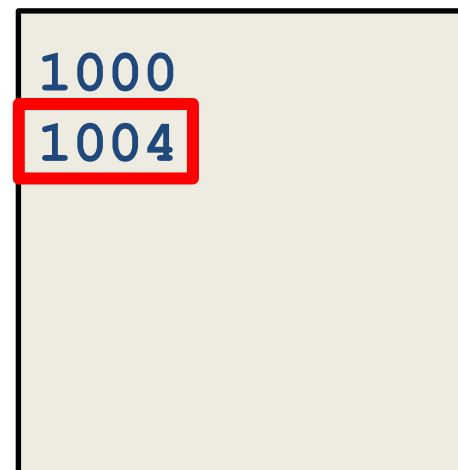
## Exemplo 4: aritmética de ponteiros

```
int v[3] = { 3, 6, 9 };  
printf("%d\n", v ); // v+0 ou &v[0]  
printf("%d\n", v + 1 ); // &v[1]  
printf("%d\n", v + 2 ); // &v[2]
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008

```
// Endereços variam a cada 4 bytes  
// sizeof(int) -> 4
```



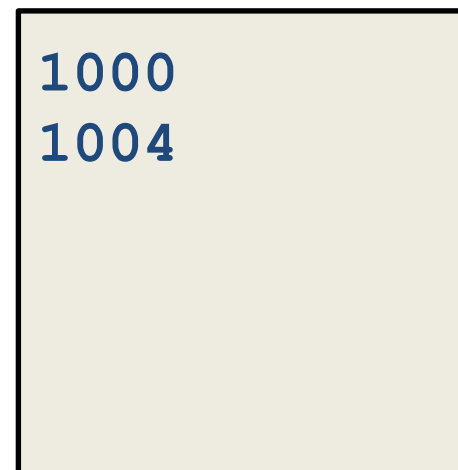
## Exemplo 4: aritmética de ponteiros

```
int v[3] = { 3, 6, 9 };  
printf("%d\n", v ); // v+0 ou &v[0]  
printf("%d\n", v + 1 ); // &v[1]  
printf("%d\n", v + 2 ); // &v[2]
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008

```
// Endereços variam a cada 4 bytes  
// sizeof(int) -> 4
```



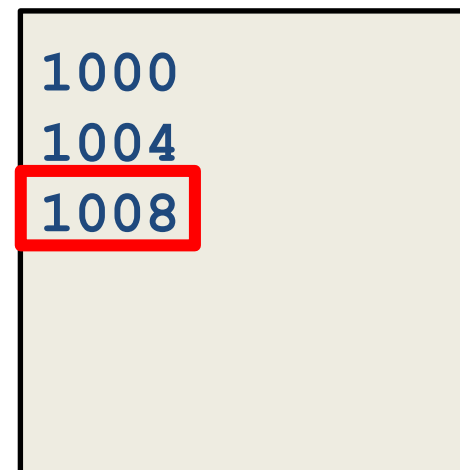
## Exemplo 4: aritmética de ponteiros

```
int v[3] = { 3, 6, 9 };  
printf("%d\n", v ); // v+0 ou &v[0]  
printf("%d\n", v + 1 ); // &v[1]  
printf("%d\n", v + 2 ); // &v[2]
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008

```
// Endereços variam a cada 4 bytes  
// sizeof(int) -> 4
```



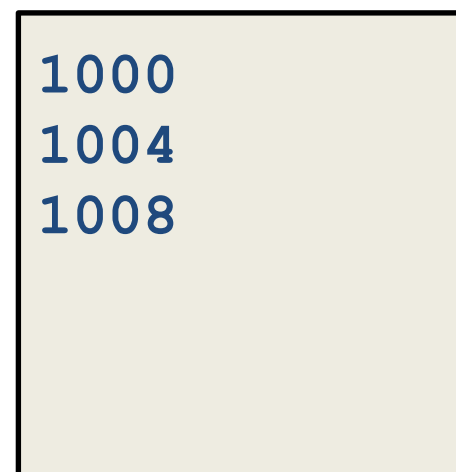
## Exemplo 4: aritmética de ponteiros

```
int v[3] = { 3, 6, 9 };  
printf("%d\n", v ); // v+0 ou &v[0]  
printf("%d\n", v + 1 ); // &v[1]  
printf("%d\n", v + 2 ); // &v[2]
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008

```
// Endereços variam a cada 4 bytes  
// sizeof(int) -> 4
```



## Exemplo 4: aritmética de ponteiros

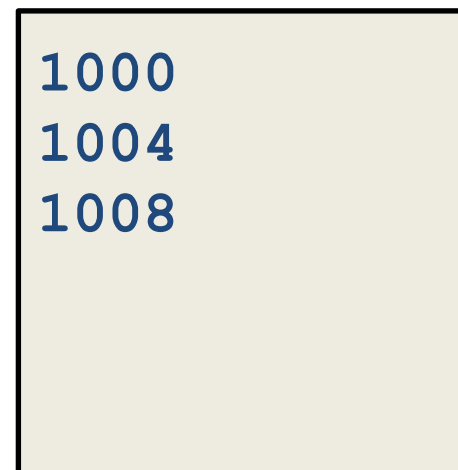
```
int v[3] = { 3, 6, 9 };  
printf("%d\n", v ); // v+0 ou &v[0]  
printf("%d\n", v + 1 ); // &v[1]  
printf("%d\n", v + 2 ); // &v[2]
```

```
printf("%d\n", *v ); // v[0]  
printf("%d\n", *(v + 1) ); // v[1]  
printf("%d\n", *(v + 2) ); // v[2]
```

```
// Endereços variam a cada 4 bytes  
// sizeof(int) -> 4
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008



## Exemplo 4: aritmética de ponteiros

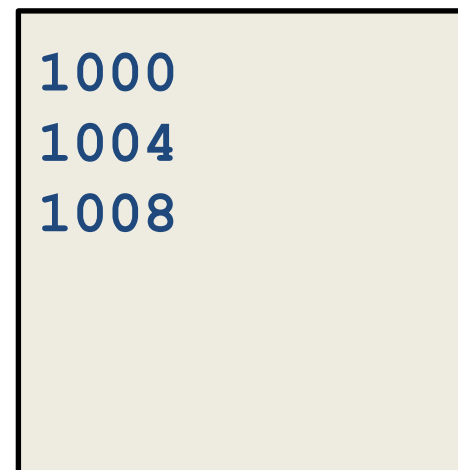
```
int v[3] = { 3, 6, 9 };  
printf("%d\n", v ); // v+0 ou &v[0]  
printf("%d\n", v + 1 ); // &v[1]  
printf("%d\n", v + 2 ); // &v[2]
```

```
printf("%d\n", *v ); // v[0]  
printf("%d\n", *(v + 1) ); // v[1]  
printf("%d\n", *(v + 2) ); // v[2]
```

```
// Endereços variam a cada 4 bytes  
// sizeof(int) -> 4
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008





## Exemplo 4: aritmética de ponteiros

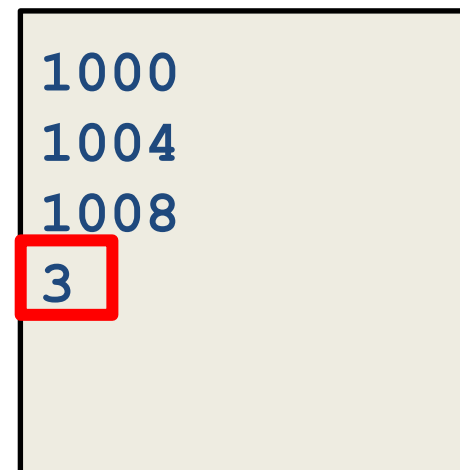
```
int v[3] = { 3, 6, 9 };  
printf("%d\n", v ); // v+0 ou &v[0]  
printf("%d\n", v + 1 ); // &v[1]  
printf("%d\n", v + 2 ); // &v[2]
```

```
printf("%d\n", *v ); // v[0]  
printf("%d\n", *(v + 1) ); // v[1]  
printf("%d\n", *(v + 2) ); // v[2]
```

```
// Endereços variam a cada 4 bytes  
// sizeof(int) -> 4
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008



## Exemplo 4: aritmética de ponteiros

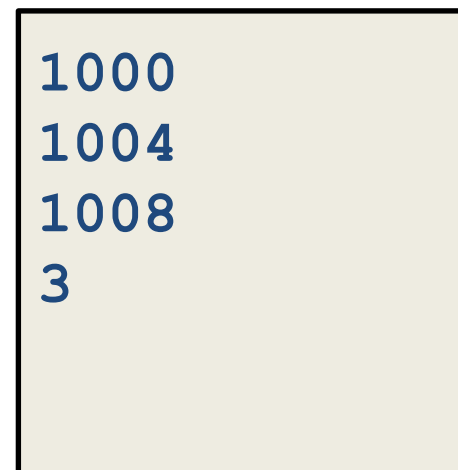
```
int v[3] = { 3, 6, 9 };  
printf("%d\n", v ); // v+0 ou &v[0]  
printf("%d\n", v + 1 ); // &v[1]  
printf("%d\n", v + 2 ); // &v[2]
```

```
printf("%d\n", *v ); // v[0]  
printf("%d\n", *(v + 1) ); // v[1]  
printf("%d\n", *(v + 2) ); // v[2]
```

```
// Endereços variam a cada 4 bytes  
// sizeof(int) -> 4
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008



## Exemplo 4: aritmética de ponteiros

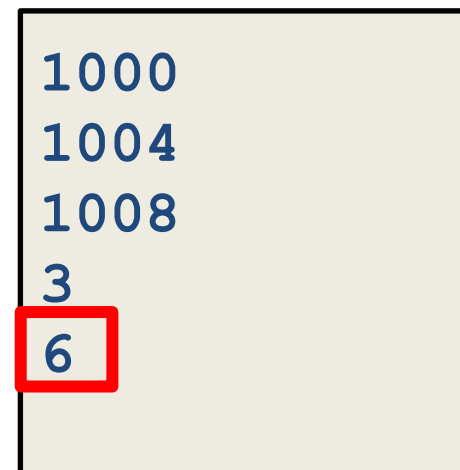
```
int v[3] = { 3, 6, 9 };  
printf("%d\n", v ); // v+0 ou &v[0]  
printf("%d\n", v + 1 ); // &v[1]  
printf("%d\n", v + 2 ); // &v[2]
```

```
printf("%d\n", *v ); // v[0]  
printf("%d\n", *(v + 1) ); // v[1]  
printf("%d\n", *(v + 2) ); // v[2]
```

```
// Endereços variam a cada 4 bytes  
// sizeof(int) -> 4
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008



## Exemplo 4: aritmética de ponteiros

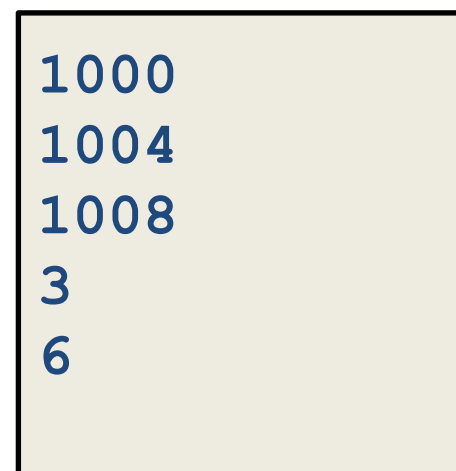
```
int v[3] = { 3, 6, 9 };  
printf("%d\n", v ); // v+0 ou &v[0]  
printf("%d\n", v + 1 ); // &v[1]  
printf("%d\n", v + 2 ); // &v[2]
```

```
printf("%d\n", *v ); // v[0]  
printf("%d\n", *(v + 1) ); // v[1]  
printf("%d\n", *(v + 2) ); // v[2]
```

```
// Endereços variam a cada 4 bytes  
// sizeof(int) -> 4
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008



## Exemplo 4: aritmética de ponteiros

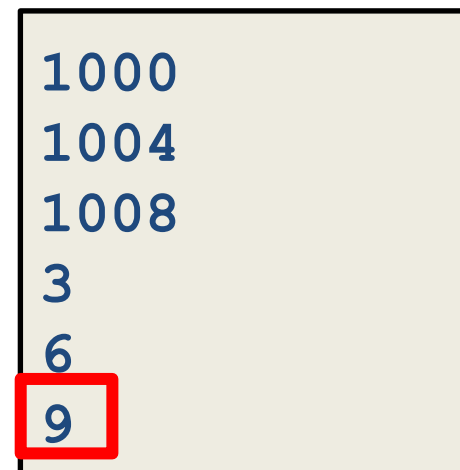
```
int v[3] = { 3, 6, 9 };  
printf("%d\n", v ); // v+0 ou &v[0]  
printf("%d\n", v + 1 ); // &v[1]  
printf("%d\n", v + 2 ); // &v[2]
```

```
printf("%d\n", *v ); // v[0]  
printf("%d\n", *(v + 1) ); // v[1]  
printf("%d\n", *(v + 2) ); // v[2]
```

```
// Endereços variam a cada 4 bytes  
// sizeof(int) -> 4
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008



## Exemplo 4: aritmética de ponteiros

```
int v[3] = { 3, 6, 9 };  
printf("%d\n", v ); // v+0 ou &v[0]  
printf("%d\n", v + 1 ); // &v[1]  
printf("%d\n", v + 2 ); // &v[2]
```

```
printf("%d\n", *v ); // v[0]  
printf("%d\n", *(v + 1) ); // v[1]  
printf("%d\n", *(v + 2) ); // v[2]
```

```
// Endereços variam a cada 4 bytes  
// sizeof(int) -> 4
```

### Modelo da Memória

v[0]	3	1000
v[1]	6	1004
v[2]	9	1008

1000
1004
1008
3
6
9

# Considerações

- Generalizando em função de um índice  $i$ , tem-se as seguintes equivalências:
- Endereço:

$$\textcolor{red}{\&v[i]} \textcolor{blue}{\leftrightarrow} v + i$$

# Considerações

- Generalizando em função de um índice  $i$ , tem-se as seguintes equivalências:

- Endereço:

$$\&v[i] \leftrightarrow v + i$$

- Valor:

$$v[i] \leftrightarrow *(v + i)$$



# **EXEMPLO 5: ARITMÉTICA DE PONTEIROS**

# Exemplo 5: aritmética de ponteiros

```
int v[5];
```

```
int i;
```

```
for( i = 0; i < 5 ; i++ ){
```

```
    scanf("%d", v + i ); // &v[i]
```

```
}
```

# Exemplo 5: aritmética de ponteiros

```
int v[5];
```

```
int i;
```

```
for( i = 0; i < 5 ; i++ ){  
    scanf("%d", v + i ); // &v[i]  
}
```

```
for( i = 0; i < 5 ; i++ ){  
    printf("%d\n", *(v + i) ); // v[i]  
}
```

# Considerações

- Em C, ponteiros e vetores podem ser tratados com os mesmos operadores;

# Considerações

- Em C, ponteiros e vetores podem ser tratados com os mesmos operadores;
  - Na prática, servem para calcular endereços;

# Considerações

- Em C, ponteiros e vetores podem ser tratados com os mesmos operadores;
  - Na prática, servem para calcular endereços;
- Todavia, lembre-se que os endereços de vetores são imutáveis!

# Considerações

- Em C, ponteiros e vetores podem ser tratados com os mesmos operadores;
  - Na prática, servem para calcular endereços;
- Todavia, lembre-se que os endereços de vetores são imutáveis!
- Para o processamento de vetores, índices podem ser substituídos por ponteiros.

# **EXEMPLO 6: TROCANDO ÍNDICE POR PONTEIRO**



## Exemplo 6: trocando índice por ponteiro

```
int v[4] = { 1, 4, 7, 11 };  
int i; // índice  
for( i = 0; i < 4 ; i++ ){  
    printf("V[%d] ", i );  
    printf("( %d) ", &v[i] );  
    printf("= %d\n", v[i] );  
}
```

### **Modelo da Memória**

v[0]	1	1000
v[1]	4	1004
v[2]	7	1008
v[3]	11	1012

## Exemplo 6: trocando índice por ponteiro

```
int v[4] = { 1, 4, 7, 11 };  
int i; // índice  
for( i = 0; i < 4 ; i++ ) {  
    printf("V[%d] ", i );  
    printf("( %d) ", &v[i] );  
    printf("= %d\n", v[i] );  
}
```

### Modelo da Memória

v[0]	1	1000
v[1]	4	1004
v[2]	7	1008
v[3]	11	1012

V[0]	(1000)	=	1
V[1]	(1004)	=	4
V[2]	(1008)	=	7
V[3]	(1012)	=	11

## Exemplo 6: trocando índice por ponteiro

```
int v[4] = { 1, 4, 7, 11 };
```

```
int *i; // ponteiro
```

```
for(          ;          ) {
```

```
    printf("V[%d] ",
```

```
    printf("( %d) ",    );
```

```
    printf("= %d\n",    );
```

```
}
```

```
/* i torna-se um  
   ponteiro */
```

### Modelo da Memória

v[0]	1	1000
v[1]	4	1004
v[2]	7	1008
v[3]	11	1012

V[0]	(1000)	=	1
V[1]	(1004)	=	4
V[2]	(1008)	=	7
V[3]	(1012)	=	11

## Exemplo 6: trocando índice por ponteiro

```
int v[4] = { 1, 4, 7, 11 };  
int *i; // ponteiro  
for(   ;                    ) {  
    printf("V[%d] ",  
    printf("( %d) ",        );  
    printf("= %d\n",        );  
}
```

*/\* Como inicializar i? \*/*

### Modelo da Memória

v[0]	1	1000
v[1]	4	1004
v[2]	7	1008
v[3]	11	1012

V[0]	(1000)	=	1
V[1]	(1004)	=	4
V[2]	(1008)	=	7
V[3]	(1012)	=	11

## Exemplo 6: trocando índice por ponteiro

```
int v[4] = { 1, 4, 7, 11 };  
int *i; // ponteiro  
for( i = v;          ) {  
    printf("V[%d] ",  
    printf("( %d) ",  
    printf("= %d\n",  
}
```

```
/* i começa em v */
```

### Modelo da Memória

v[0]	1	1000
v[1]	4	1004
v[2]	7	1008
v[3]	11	1012

V[0]	(1000)	=	1
V[1]	(1004)	=	4
V[2]	(1008)	=	7
V[3]	(1012)	=	11

## Exemplo 6: trocando índice por ponteiro

```
int v[4] = { 1, 4, 7, 11 };  
int *i; // ponteiro  
for( i = v; i < ; ) {  
    printf("V[%d] ",  
    printf("( %d) ", );  
    printf("= %d\n", );  
}
```

```
/* i deve ser  
   menor que ...? */
```

### Modelo da Memória

v[0]	1	1000
v[1]	4	1004
v[2]	7	1008
v[3]	11	1012

V[0]	(1000)	=	1
V[1]	(1004)	=	4
V[2]	(1008)	=	7
V[3]	(1012)	=	11

## Exemplo 6: trocando índice por ponteiro

```
int v[4] = { 1, 4, 7, 11 };
int *i; // ponteiro
for( i = v; i < v + 4;      ) {
    printf("V[%d] ",
    printf("( %d) ",      );
    printf("= %d\n",      );
}
```

```
/* v + 4 = 1016 */
```

### Modelo da Memória

v[0]	1	1000
v[1]	4	1004
v[2]	7	1008
v[3]	11	1012

V[0]	(1000)	=	1
V[1]	(1004)	=	4
V[2]	(1008)	=	7
V[3]	(1012)	=	11

## Exemplo 6: trocando índice por ponteiro

```
int v[4] = { 1, 4, 7, 11 };  
int *i; // ponteiro  
for( i = v; i < v + 4 ;   ) {  
    printf("V[%d] ",  
    printf("( %d) ",    );  
    printf("= %d\n",    );  
}
```

*/\* Como incrementar i? \*/*

### Modelo da Memória

v[0]	1	1000
v[1]	4	1004
v[2]	7	1008
v[3]	11	1012

V[0]	(1000)	=	1
V[1]	(1004)	=	4
V[2]	(1008)	=	7
V[3]	(1012)	=	11



## Exemplo 6: trocando índice por ponteiro

```
int v[4] = { 1, 4, 7, 11 };  
int *i; // ponteiro  
for( i = v; i < v + 4 ; i++) {  
    printf("V[%d] ",  
    printf("( %d) ",  
    printf("= %d\n",  
    );  
};
```

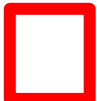
```
/* Incremento de ponteiro  
de acordo com o  
sizeof(int) */
```

### Modelo da Memória

v[0]	1	1000
v[1]	4	1004
v[2]	7	1008
v[3]	11	1012

V[0]	(1000)	=	1
V[1]	(1004)	=	4
V[2]	(1008)	=	7
V[3]	(1012)	=	11

## Exemplo 6: trocando índice por ponteiro

```
int v[4] = { 1, 4, 7, 11 };  
int *i; // ponteiro  
for( i = v; i < v + 4 ; i++ ){  
    printf("V[%d] ",  
    printf("( %d) ",  );  
    printf("= %d\n",      );  
}
```

```
/* Como determinar o  
   endereço? */
```

### Modelo da Memória

v[0]	1	1000
v[1]	4	1004
) ; v[2]	7	1008
v[3]	11	1012

V[0]	(1000)	=	1
V[1]	(1004)	=	4
V[2]	(1008)	=	7
V[3]	(1012)	=	11

## Exemplo 6: trocando índice por ponteiro

```
int v[4] = { 1, 4, 7, 11 };  
int *i; // ponteiro  
for( i = v; i < v + 4 ; i++ ){  
    printf("V[%d] ",  
    printf("( %d) ", i );  
    printf("= %d\n",      );  
}
```

```
/* i é o próprio  
   endereço */
```

### Modelo da Memória

v[0]	1	1000
v[1]	4	1004
) ; v[2]	7	1008
v[3]	11	1012

V[0]	(1000)	=	1
V[1]	(1004)	=	4
V[2]	(1008)	=	7
V[3]	(1012)	=	11

## Exemplo 6: trocando índice por ponteiro

```
int v[4] = { 1, 4, 7, 11 };  
int *i; // ponteiro  
for( i = v; i < v + 4 ; i++ ){  
    printf("V[%d] ",  
    printf("( %d) ", i );  
    printf("= %d\n",  );  
}
```

```
/* Como determinar o  
   valor?  
*/
```

### Modelo da Memória

v[0]	1	1000
v[1]	4	1004
) ; v[2]	7	1008
v[3]	11	1012

V[0]	(1000)	=	1
V[1]	(1004)	=	4
V[2]	(1008)	=	7
V[3]	(1012)	=	11

## Exemplo 6: trocando índice por ponteiro

```
int v[4] = { 1, 4, 7, 11 };  
int *i; // ponteiro  
for( i = v; i < v + 4 ; i++ ){  
    printf("V[%d] ",  
    printf("( %d) ", i );  
    printf("= %d\n", *i );  
}
```

```
/* Valor é determinado  
   pelo operador *  
*/
```

### Modelo da Memória

v[0]	1	1000
v[1]	4	1004
v[2]	7	1008
v[3]	11	1012

V[0]	(1000)	=	1
V[1]	(1004)	=	4
V[2]	(1008)	=	7
V[3]	(1012)	=	11

## Exemplo 6: trocando índice por ponteiro

```
int v[4] = { 1, 4, 7, 11 };  
int *i; // ponteiro  
for( i = v; i < v + 4 ; i++ ){  
    printf("V[%d] ",  );  
    printf("( %d) ", i );  
    printf("= %d\n", *i );  
}
```

```
/* Como determinar  
   o índice a partir  
   de um endereço? */
```

### Modelo da Memória

v[0]	1	1000
v[1]	4	1004
v[2]	7	1008
v[3]	11	1012

V[0]	(1000)	=	1
V[1]	(1004)	=	4
V[2]	(1008)	=	7
V[3]	(1012)	=	11

## Exemplo 6: trocando índice por ponteiro

```
int v[4] = { 1, 4, 7, 11 };
int *i; // ponteiro
for( i = v; i < v + 4 ; i++ ){
    printf("V[%d] ", (i-v) / sizeof(int) );
    printf("(%d) ", i );
    printf("= %d\n", *i );
}
```

```
/* Resultado da subtração
dividido pelo
sizeof(int) */
```

### Modelo da Memória

v[0]	1	1000
v[1]	4	1004
v[2]	7	1008
v[3]	11	1012

V[0]	(1000)	=	1
V[1]	(1004)	=	4
V[2]	(1008)	=	7
V[3]	(1012)	=	11

## Exemplo 6: trocando índice por ponteiro

```
int v[4] = { 1, 4, 7, 11 };
int *i; // ponteiro
for( i = v; i < v + 4 ; i++ ){
    printf("V[%d] ", (i-v) / sizeof(int) );
    printf("(%d) ", i );
    printf("= %d\n", *i );
}
```

```
/* Resultado da subtração
dividido pelo
sizeof(int) */
```

### Modelo da Memória

v[0]	1	1000
v[1]	4	1004
v[2]	7	1008
v[3]	11	1012

V[0]	(1000)	=	1
V[1]	(1004)	=	4
V[2]	(1008)	=	7
V[3]	(1012)	=	11



# COMPARANDO EXEMPLOS

# Comparação entre ponteiro e índice

```
int v[4] = { 1, 4, 7, 11 };
int *i; // ponteiro
for(i = v; i < v+4 ; i++){
    printf("V[%d] ", (i-v)/4);
    printf("( %d) ", i );
    printf("= %d\n", *i );
}
```

```
int v[4] = { 1, 4, 7, 11 };
int i; // índice
for(i = 0; i < 4 ; i++){
    printf("V[%d] ", i);
    printf("( %d) ", &v[i] );
    printf("= %d\n", v[i] );
}
```

## Modelo da Memória

v[0]	1	1000
v[1]	4	1004
v[2]	7	1008
v[3]	11	1012

v[0]	(1000)	=	1
v[1]	(1004)	=	4
v[2]	(1008)	=	7
v[3]	(1012)	=	11

# **EXEMPLO 7: INVERTENDO UM VETOR**

# Exemplo 7: invertendo um vetor

```
→ int v[7] = {4, 3, 6, 10, 2, -1, 9};  
int n = 7;  
int *i = v;           // primeiro  
int *j = v + n - 1;   // último  
while( i < j ){  
    int aux = *i;  
    *i = *j;  
    *j = aux;  
    i++; // incremento do ponteiro  
    j--; // decremento do ponteiro  
}  
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	4	1000
v[1]	3	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	-1	1020
v[6]	9	1024
n		1028
i		1032
j		1038
aux		1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};
```

→ 

```
int n = 7;
```

```
int *i = v;           // primeiro
```

```
int *j = v + n - 1;  // último
```

```
while( i < j ){
```

```
    int aux = *i;
```

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

```
    *j = aux;
```

```
    i++; // incremento do ponteiro
```

```
    j--; // decremento do ponteiro
```

```
}
```

```
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	4	1000
v[1]	3	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	-1	1020
v[6]	9	1024
n	7	1028
i		1032
j		1038
aux		1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};
```

```
int n = 7;
```

```
→ int *i = v;           // primeiro
```

```
int *j = v + n - 1;    // último
```

```
while( i < j ){
```

```
    int aux = *i;
```

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

```
    *j = aux;
```

```
    i++; // incremento do ponteiro
```

```
    j--; // decremento do ponteiro
```

```
}
```

```
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	4	1000
v[1]	3	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	-1	1020
v[6]	9	1024
n	7	1028
i	1000	1032
j		1038
aux		1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};
int n = 7;
int *i = v;           // primeiro
→ int *j = v + n - 1; // último
while( i < j ){
    int aux = *i;
    *i = *j;
    *j = aux;
    i++; // incremento do ponteiro
    j--; // decremento do ponteiro
}
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	4	1000
v[1]	3	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	-1	1020
v[6]	9	1024
n	7	1028
i	1000	1032
j	1024	1038
aux		1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};
int n = 7;
int *i = v;           // primeiro
int *j = v + n - 1;   // último
→ while( i < j ){
    int aux = *i;
    *i = *j;
    *j = aux;
    i++; // incremento do ponteiro
    j--; // decremento do ponteiro
}
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	4	1000
v[1]	3	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	-1	1020
v[6]	9	1024
n	7	1028
i	1000	1032
j	1024	1038
aux		1042



# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};  
int n = 7;  
int *i = v;           // primeiro  
int *j = v + n - 1;   // último  
while( i < j ){  
    → int aux = *i;  
      *i = *j;  
      *j = aux;  
      i++; // incremento do ponteiro  
      j--; // decremento do ponteiro  
}  
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	4	1000
v[1]	3	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	-1	1020
v[6]	9	1024
n	7	1028
i	1000	1032
j	1024	1038
aux	4	1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};
int n = 7;
int *i = v;           // primeiro
int *j = v + n - 1;   // último
while( i < j ){
    int aux = *i;
    *i = *j;
    *j = aux;
    i++; // incremento do ponteiro
    j--; // decremento do ponteiro
}
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	9	1000
v[1]	3	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	-1	1020
v[6]	9	1024
n	7	1028
i	1000	1032
j	1024	1038
aux	4	1042

# Exemplo 7: invertendo um vetor

```

int v[7] = {4, 3, 6, 10, 2, -1, 9};
int n = 7;
int *i = v;           // primeiro
int *j = v + n - 1;   // último
while( i < j ){
    int aux = *i;
    *i = *j;
    *j = aux;
    i++; // incremento do ponteiro
    j--; // decremento do ponteiro
}
mostra_vetor( v , n );
  
```

## Modelo da Memória

v[0]	9	1000
v[1]	3	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	-1	1020
v[6]	4	1024
n	7	1028
i	1000	1032
j	1024	1038
aux	4	1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};  
int n = 7;  
int *i = v;           // primeiro  
int *j = v + n - 1;   // último  
while( i < j ){  
    int aux = *i;  
    *i = *j;  
    *j = aux;  
    i++; // incremento do ponteiro  
    j--; // decremento do ponteiro  
}  
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	9	1000
v[1]	3	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	-1	1020
v[6]	4	1024
n	7	1028
i	1004	1032
j	1024	1038
aux	4	1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};
int n = 7;
int *i = v;           // primeiro
int *j = v + n - 1;   // último
while( i < j ){
    int aux = *i;
    *i = *j;
    *j = aux;
    i++; // incremento do ponteiro
    j--; // decremento do ponteiro
}
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	9	1000
v[1]	3	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	-1	1020
v[6]	4	1024
n	7	1028
i	1004	1032
j	1020	1038
aux	4	1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};
```

```
int n = 7;
```

```
int *i = v;           // primeiro
```

```
int *j = v + n - 1;  // último
```

```
→ while( i < j ){
    int aux = *i;
    *i = *j;
    *j = aux;
    i++; // incremento do ponteiro
    j--; // decremento do ponteiro
}
```

```
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	9	1000
v[1]	3	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	-1	1020
v[6]	4	1024
n	7	1028
i	1004	1032
j	1020	1038
aux	4	1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};
int n = 7;
int *i = v;           // primeiro
int *j = v + n - 1;   // último
while( i < j ){
    int aux = *i;
    *i = *j;
    *j = aux;
    i++; // incremento do ponteiro
    j--; // decremento do ponteiro
}
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	9	1000
v[1]	3	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	-1	1020
v[6]	4	1024
n	7	1028
i	1004	1032
j	1020	1038
aux	3	1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};
int n = 7;
int *i = v;           // primeiro
int *j = v + n - 1;   // último
while( i < j ){
    int aux = *i;
    *i = *j;
    *j = aux;
    i++; // incremento do ponteiro
    j--; // decremento do ponteiro
}
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	9	1000
v[1]	-1	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	-1	1020
v[6]	4	1024
n	7	1028
i	1004	1032
j	1020	1038
aux	3	1042



# Exemplo 7: invertendo um vetor

```

int v[7] = {4, 3, 6, 10, 2, -1, 9};
int n = 7;
int *i = v;           // primeiro
int *j = v + n - 1;   // último
while( i < j ){
    int aux = *i;
    *i = *j;
    *j = aux;
    i++; // incremento do ponteiro
    j--; // decremento do ponteiro
}
mostra_vetor( v , n );
  
```

## Modelo da Memória

v[0]	9	1000
v[1]	-1	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	3	1020
v[6]	4	1024
n	7	1028
i	1004	1032
j	1020	1038
aux	3	1042

# Exemplo 7: invertendo um vetor

```

int v[7] = {4, 3, 6, 10, 2, -1, 9};
int n = 7;
int *i = v;           // primeiro
int *j = v + n - 1;   // último
while( i < j ){
    int aux = *i;
    *i = *j;
    *j = aux;
    i++; // incremento do ponteiro
    j--; // decremento do ponteiro
}
mostra_vetor( v , n );
  
```

## Modelo da Memória

v[0]	9	1000
v[1]	-1	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	3	1020
v[6]	4	1024
n	7	1028
i	1008	1032
j	1020	1038
aux	3	1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};  
int n = 7;  
int *i = v;           // primeiro  
int *j = v + n - 1;   // último  
while( i < j ){  
    int aux = *i;  
    *i = *j;  
    *j = aux;  
    i++; // incremento do ponteiro  
    j--; // decremento do ponteiro  
}  
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	9	1000
v[1]	-1	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	3	1020
v[6]	4	1024
n	7	1028
i	1008	1032
j	1016	1038
aux	3	1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};
```

```
int n = 7;
```

```
int *i = v;           // primeiro
```

```
int *j = v + n - 1;  // último
```

```
→ while( i < j ){  
    int aux = *i;  
    *i = *j;  
    *j = aux;  
    i++; // incremento do ponteiro  
    j--; // decremento do ponteiro  
}
```

```
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	9	1000
v[1]	-1	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	3	1020
v[6]	4	1024
n	7	1028
i	1008	1032
j	1016	1038
aux	3	1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};
int n = 7;
int *i = v;           // primeiro
int *j = v + n - 1;   // último
while( i < j ){
    int aux = *i;
    *i = *j;
    *j = aux;
    i++; // incremento do ponteiro
    j--; // decremento do ponteiro
}
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	9	1000
v[1]	-1	1004
v[2]	6	1008
v[3]	10	1012
v[4]	2	1016
v[5]	3	1020
v[6]	4	1024
n	7	1028
i	1008	1032
j	1016	1038
aux	6	1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};  
int n = 7;  
int *i = v;           // primeiro  
int *j = v + n - 1;   // último  
while( i < j ){  
    int aux = *i;  
    *i = *j;  
    *j = aux;  
    i++; // incremento do ponteiro  
    j--; // decremento do ponteiro  
}  
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	9	1000
v[1]	-1	1004
v[2]	2	1008
v[3]	10	1012
v[4]	2	1016
v[5]	3	1020
v[6]	4	1024
n	7	1028
i	1008	1032
j	1016	1038
aux	6	1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};
int n = 7;
int *i = v;           // primeiro
int *j = v + n - 1;   // último
while( i < j ){
    int aux = *i;
    *i = *j;
    *j = aux;
    i++; // incremento do ponteiro
    j--; // decremento do ponteiro
}
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	9	1000
v[1]	-1	1004
v[2]	2	1008
v[3]	10	1012
v[4]	6	1016
v[5]	3	1020
v[6]	4	1024
n	7	1028
i	1008	1032
j	1016	1038
aux	6	1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};
int n = 7;
int *i = v;           // primeiro
int *j = v + n - 1;   // último
while( i < j ){
    int aux = *i;
    *i = *j;
    *j = aux;
    i++; // incremento do ponteiro
    j--; // decremento do ponteiro
}
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	9	1000
v[1]	-1	1004
v[2]	2	1008
v[3]	10	1012
v[4]	6	1016
v[5]	3	1020
v[6]	4	1024
n	7	1028
i	1012	1032
j	1016	1038
aux	6	1042



# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};
int n = 7;
int *i = v;           // primeiro
int *j = v + n - 1;   // último
while( i < j ){
    int aux = *i;
    *i = *j;
    *j = aux;
    i++; // incremento do ponteiro
    j--; // decremento do ponteiro
}
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	9	1000
v[1]	-1	1004
v[2]	2	1008
v[3]	10	1012
v[4]	6	1016
v[5]	3	1020
v[6]	4	1024
n	7	1028
i	1012	1032
j	1012	1038
aux	6	1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};
```

```
int n = 7;
```

```
int *i = v;           // primeiro
```

```
int *j = v + n - 1;  // último
```

```
→ while( i < j ){  
    int aux = *i;  
    *i = *j;  
    *j = aux;  
    i++; // incremento do ponteiro  
    j--; // decremento do ponteiro  
}
```

```
mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	9	1000
v[1]	-1	1004
v[2]	2	1008
v[3]	10	1012
v[4]	6	1016
v[5]	3	1020
v[6]	4	1024
n	7	1028
i	1012	1032
j	1012	1038
aux	6	1042

# Exemplo 7: invertendo um vetor

```
int v[7] = {4, 3, 6, 10, 2, -1, 9};
int n = 7;
int *i = v;           // primeiro
int *j = v + n - 1;   // último
while( i < j ){
    int aux = *i;
    *i = *j;
    *j = aux;
    i++; // incremento do ponteiro
    j--; // decremento do ponteiro
}
→ mostra_vetor( v , n );
```

## Modelo da Memória

v[0]	9	1000
v[1]	-1	1004
v[2]	2	1008
v[3]	10	1012
v[4]	6	1016
v[5]	3	1020
v[6]	4	1024
n	7	1028
i	1012	1032
j	1012	1038
aux	6	1042