

APONTANDO E APRONTANDO:

ALTAS CONFUSÕES COM ENDEREÇOS E PONTEIROS

Encontro 2 de Desafios de Programação

```
#include <stdio.h>
```

```
void flip(int n) {  
    n = 1 - n;  
}
```

```
int main(void) {  
    int n = 0;  
    flip(n);  
    printf("%d", n);  
  
    return 0;  
}
```

Qual é a saída?

a) 0

b) 1

```
#include <stdio.h>
```

```
void flip(int n) {  
    n = 1 - n;  
}
```

```
int main(void) {  
    int n = 0;  
    flip(n);  
    printf("%d", n);  
  
    return 0;  
}
```

Qual é a saída?

a) 0 ✓

b) 1

```
#include <stdio.h>
```

```
void flip(int v[], int i) {  
    v[i] = 1 - v[i];  
}
```

```
int main(void) {  
    int v[] = {0, 0, 0, 0, 0};  
    flip(v, 2);  
    printf("%d", v[2]);  
  
    return 0;  
}
```

```
#include <stdio.h>

void flip(int v[], int i) {
    v[i] = 1 - v[i];
}

int main(void) {
    int v[] = {0, 0, 0, 0, 0};
    flip(v, 2);
    printf("%d", v[2]);

    return 0;
}
```

inicialização com inferência
automática de tamanho

```
#include <stdio.h>
```

```
void flip(int v[], int i) {  
    v[i] = 1 - v[i];  
}
```

```
int main(void) {  
    int v[] = {0, 0, 0, 0, 0};  
    flip(v, 2);  
    printf("%d", v[2]);  
  
    return 0;  
}
```

Qual é a saída?

a) 0

b) 1

```
#include <stdio.h>
```

```
void flip(int v[], int i) {  
    v[i] = 1 - v[i];  
}
```

```
int main(void) {  
    int v[] = {0, 0, 0, 0, 0};  
    flip(v, 2);  
    printf("%d", v[2]);  
  
    return 0;  
}
```

Qual é a saída?

a) 0

b) 1 ✓

```
#include <stdio.h>

void flip(int v[], int i) {
    v[i] = 1 - v[i];
}

int main(void) {
    int v[] = {0, 0, 0, 0, 0};
    flip(v, 2);
    printf("%d", v[2]);

    return 0;
}
```

Qual é a saída?

a) 0

b) 1 ✓

¿qué pasa?

Quando um parâmetro é vetor,
ele é tratado de “maneira especial”?

Quando um parâmetro é vetor,
ele é tratado de “maneira especial”?

Não!

```
#include <stdio.h>
```

```
void flip(int n) {  
    n = 1 - n;  
}
```

```
int main(void) {  
    int n = 0;  
    flip(n);  
    printf("%d", n);  
  
    return 0;  
}
```

E se *quisermos* que `flip` tenha de fato o poder de alterar `n`?

```
#include <stdio.h>
```

```
void flip(int n) {  
    n = 1 - n;  
}
```

```
int main(void) {  
    int n = 0;  
    flip(n);  
    printf("%d", n);  
  
    return 0;  
}
```

E se *quisermos* que *flip* tenha de fato o poder de alterar *n*?

(ou seja, “tratar variável como trata vetor”)

```
#include <stdio.h>
```

```
void flip(int *n) {  
    *n = 1 - *n;  
}
```

```
int main(void) {  
    int n = 0;  
    flip(&n);  
    printf("%d", n);  
  
    return 0;  
}
```

Eis o famoso asterisco!

```
#include <stdio.h>

void flip(int *n) {
    *n = 1 - *n;
}

int main(void) {
    int n = 0;
    flip(&n);
    printf("%d", n);

    return 0;
}
```

Opa, mudou aqui também!

```
#include <stdio.h>

void flip(int *n) {
    *n = 1 - *n;
}

int main(void) {
    int n = 0;
    flip(&n);
    printf("%d", n);

    return 0;
}
```

Opa, mudou aqui também!
E isso parece familiar...

```
#include <stdio.h>
```

```
void flip(int n) {  
    n = 1 - n;  
}
```

```
int main(void) {  
    int n = 0;  
    flip(n);  
    printf("%d", n);  
  
    return 0;  
}
```



```
#include <stdio.h>
```

```
void flip(int n) {  
    n = 1 - n;  
}
```

```
int main(void) {  
    int n = 0;  
    flip(n);  
    printf("%d", n);  
  
    return 0;  
}
```

O que é uma variável?

```
#include <stdio.h>
```

```
void flip(int n) {  
    n = 1 - n;  
}
```

```
int main(void) {  
    int n = 0;  
    flip(n);  
    printf("%d", n);  
  
    return 0;  
}
```

O que é uma variável?

Um espaço na memória...

```
#include <stdio.h>
```

```
void flip(int n) {  
    n = 1 - n;  
}
```

```
int main(void) {  
    int n = 0;  
    flip(n);  
    printf("%d", n);  
  
    return 0;  
}
```

O que é uma variável?

Um espaço na memória, para guardar um valor de um certo tipo...

```
#include <stdio.h>
```

```
void flip(int n) {  
    n = 1 - n;  
}
```

```
int main(void) {  
    int n = 0;  
    flip(n);  
    printf("%d", n);  
  
    return 0;  
}
```

O que é uma variável?

Um espaço na memória, para guardar um valor de um certo tipo, que podemos acessar através de um nome.

```
#include <stdio.h>
```

```
void flip(int n) {  
    n = 1 - n;  
}
```

```
int main(void) {  
    int n = 0;  
    flip(n);  
    printf("%d", n);  
  
    return 0;  
}
```

O nome só vale localmente!

```
#include <stdio.h>
```

```
void flip(int n) {  
    n = 1 - n;  
}
```

```
int main(void) {  
    int n = 0;  
    flip(n);  
    printf("%d", n);  
  
    return 0;  
}
```

Esse n...

...não tem nada a ver com esse n.

```
int n;  
scanf("%d", n);
```

Qual é o problema com esse código?

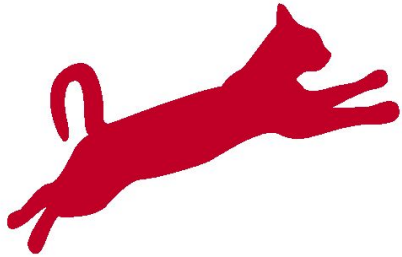
```
int n;  
scanf("%d", n);
```

Qual é o problema com esse código?
scanf não tem o poder de alterar n!


```
int n;  
scanf("%d", &n);
```

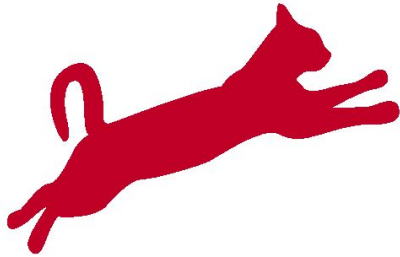
Além do nome, também podemos acessar uma variável através de um endereço.

Além do nome, também podemos acessar uma variável através de um endereço.



E o endereço vale globalmente!

Além do nome, também podemos acessar uma variável através de um endereço.



E o endereço vale globalmente!

Uma função pode alterar uma variável de outra função se souber o endereço dela!

como podemos passar
endereços para lá e para cá?

apontadores (ou ponteiros):
variáveis que guardam endereços



```
int x;
```

variável inteira



X


```
int x;  
x = 5;
```

5

x

```
int x;  
x = 5;
```

```
int *p;
```

apontador para inteiro



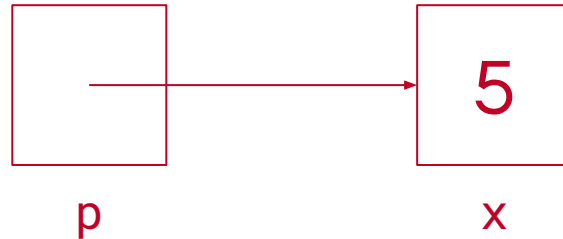
p



x

```
int x;  
x = 5;
```

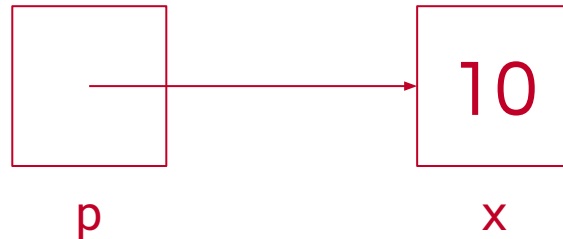
```
int *p;  
p = &x;
```



```
int x;  
x = 5;
```

```
int *p;  
p = &x;
```

```
*p = 10;
```



Duas regrinhas que facilitarão sua vida:

Duas regrinhas que facilitarão sua vida:

1) leia & como *endereço de*;

Duas regrinhas que facilitarão sua vida:

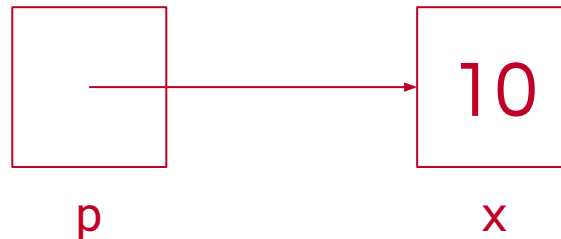
1) leia & como *endereço de*;

2) leia * como *variável apontada por*.

```
int x;  
x = 5;
```

```
int *p;  
p = &x;
```

```
*p = 10;
```

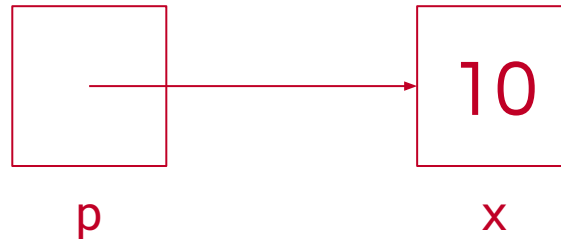



```
int x;  
x = 5;
```

```
int *p;  
p = &x;
```

```
*p = 10;
```

“x é inteira”

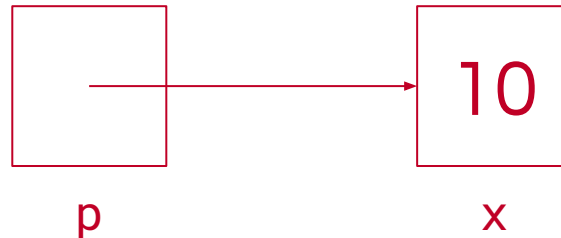


```
int x;  
x = 5;
```

```
int *p;  
p = &x;
```

```
*p = 10;
```

“x é inteira”
“x recebe 5”



```
int x;  
x = 5;
```

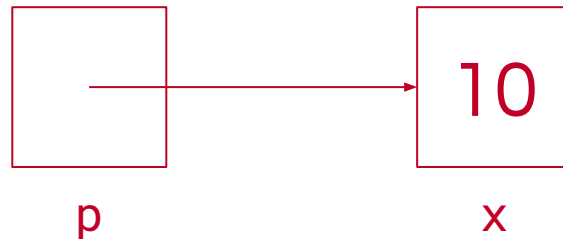
```
int *p;  
p = &x;
```

```
*p = 10;
```

“x é inteira”

“x recebe 5”

“variável apontada por p é inteira”



```
int x;  
x = 5;
```

```
int *p;  
p = &x;
```

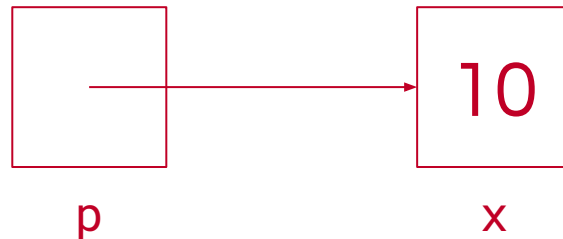
```
*p = 10;
```

“x é inteira”

“x recebe 5”

“variável apontada por p é inteira”

“p recebe endereço de x”



```
int x;  
x = 5;
```

```
int *p;  
p = &x;
```

```
*p = 10;
```

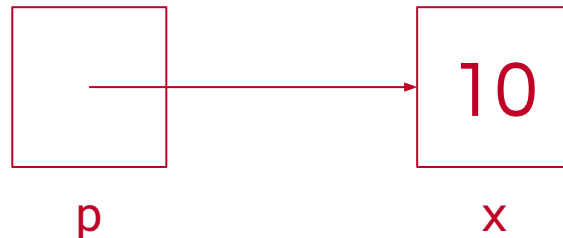
“x é inteira”

“x recebe 5”

“variável apontada por p é inteira”

“p recebe endereço de x”

“variável apontada por p recebe 10”



```
int &x;  
x = 5;
```

```
int *p;  
p = &x;
```

```
*p = 10;
```

isso faz sentido?

```
int x;  
&x = 5;
```

```
int *p;  
p = &x;
```

```
*p = 10;
```

isso faz sentido?

```
int *x;  
x = 5;
```

```
int *p;  
p = &x;
```

```
*p = 10;
```

isso faz sentido?


```
int x;  
*x = 5;
```

```
int *p;  
p = &x;
```

```
*p = 10;
```

isso faz sentido?

```
int x;  
x = 5;
```

```
int p;  
p = &x;
```

```
*p = 10;
```

isso faz sentido?

```
int x;  
x = 5;
```

```
int &p;  
p = &x;
```

```
*p = 10;
```

isso faz sentido?

```
int x;  
x = 5;
```

```
int *p;  
&p = &x;
```

```
*p = 10;
```

isso faz sentido?

```
int x;  
x = 5;
```

```
int *p;  
*p = &x;
```

```
*p = 10;
```

isso faz sentido?

```
int x;  
x = 5;
```

```
int *p;  
p = x;
```

```
*p = 10;
```

isso faz sentido?

```
int x;  
x = 5;
```

```
int *p;  
p = *x;
```

```
*p = 10;
```

isso faz sentido?

```
int x;  
x = 5;
```

```
int *p;  
p = &x;
```

```
p = 10;
```

isso faz sentido?


```
int x;  
x = 5;
```

```
int *p;  
p = &x;
```

```
&p = 10;
```

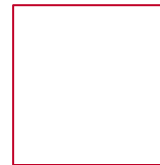
isso faz sentido?

```
**p = 10;
```

isso pode fazer sentido?

```
**p = 10;
```

“variável apontada por...”



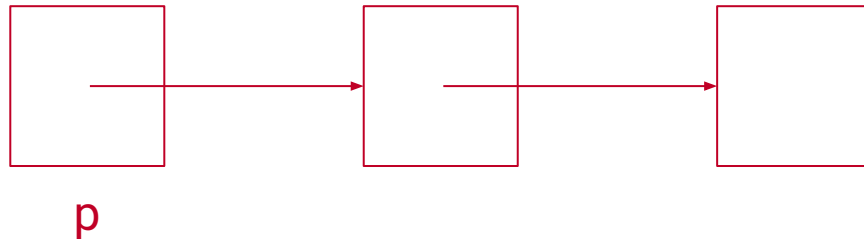
```
**p = 10;
```

“variável apontada por...
...variável apontada por...”



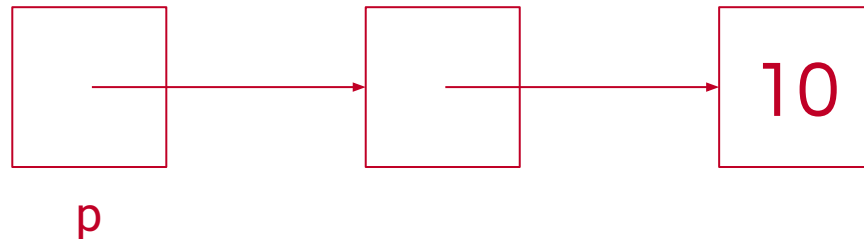
```
**p = 10;
```

“variável apontada por...
...variável apontada por...
...p...”



```
**p = 10;
```

“variável apontada por...
...variável apontada por...
...p...
...recebe 10”

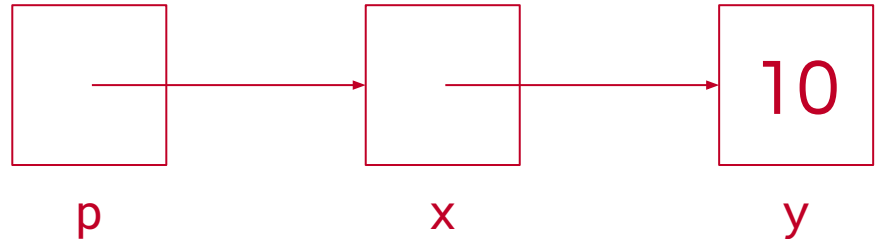


```
int y;
```

```
int *x;  
x = &y;
```

```
int **p;  
p = &x;
```

```
**p = 10;
```



Duas regrinhas que facilitarão sua vida:

1) leia & como *endereço de*;

2) leia * como *variável apontada por*.

Duas regrinhas que facilitarão sua vida:

1) leia & como *endereço de*;

2) leia * como *variável apontada por*.

Só há uma exceção:

```
int *p = &x;
```

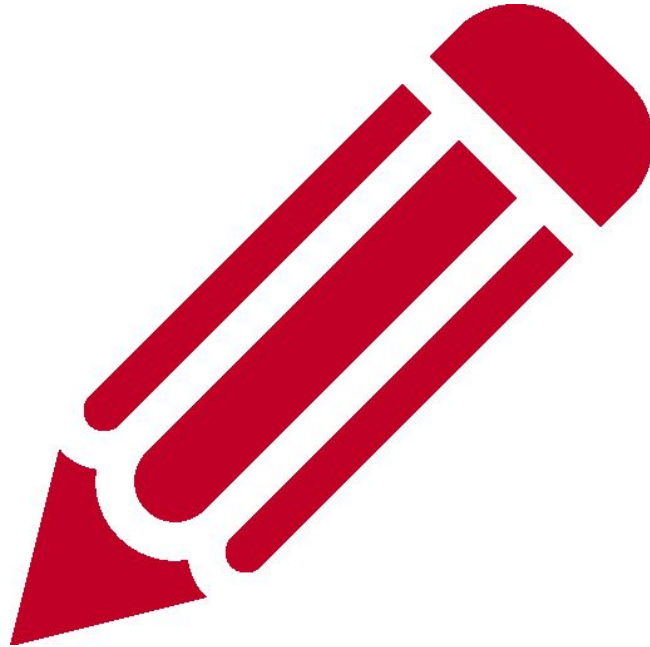
(alguma coisa como
`pointer<int> p = &x;`
teria sido melhor, mas não existe)

O MINISTÉRIO DA PROGRAMAÇÃO ADVERTE:

em C++, os significados de & e * são diferentes dependendo do contexto. Por enquanto, pensar nisso pode causar confusão, insanidade temporária e perda de vida social. Persistindo os sintomas, contate o atendimento.

**E não esqueça da ferramenta mais poderosa
do mundo para entender apontadores!**

**E não esqueça da ferramenta mais poderosa
do mundo para entender apontadores!**



Le Quiz 1

```
int i = 5;  
int *p;  
p = &i;  
i += 2;  
printf( "%d", *p );
```

- a) 5
 - b) 7
 - c) endereço de i
 - d) endereço de p
- mãos vazias) hein?**

Le Quiz 1

```
int i = 5;  
int *p;  
p = &i;  
i += 2;  
printf( "%d", *p );
```

- a) 5
 - ✓ b) 7
 - c) endereço de i
 - d) endereço de p
- mãos vazias) hein?**

Le Quiz 2

```
int i = 5;  
int *p;  
*p = &i;  
i += 2;  
printf( "%d", *p );
```

- a) 5
 - b) 7
 - c) endereço de i
 - d) endereço de p
- mãos vazias) hein?**

Le Quiz 2

```
int i = 5;  
int *p;  
*p = &i;  
i += 2;  
printf( "%d", *p );
```

- a) 5
- b) 7
- c) endereço de i
- d) endereço de p
- ✓ **mãos vazias)** hein?


Le Quiz 3

```
int i = 5;  
int *p;  
p = &i;  
i += 2;  
printf( "%d", p );
```

- a) 5
 - b) 7
 - c) endereço de i
 - d) endereço de p
- mãos vazias) hein?**

Le Quiz 3

```
int i = 5;  
int *p;  
p = &i;  
i += 2;  
printf( "%d", p );
```

- a) 5
 - b) 7
 -  c) endereço de i
 - d) endereço de p
- mãos vazias) hein?**

Le Quiz 4

```
int i = 5;  
int *p;  
p = &i;  
p += 2;  
printf( "%d", *p );
```

- a) 5
 - b) 7
 - c) endereço de i
 - d) endereço de p
- mãos vazias) hein?**

Le Quiz 4

```
int i = 5;  
int *p;  
p = &i;  
p += 2;  
printf( "%d", *p );
```

- a) 5
- b) 7
- c) endereço de i
- d) endereço de p
- ✓ **mãos vazias)** hein?

Le Quiz 5

```
int i = 5;  
int *p;  
p = &i;  
*p += 2;  
printf( "%d", *p );
```

- a) 5
 - b) 7
 - c) endereço de i
 - d) endereço de p
- mãos vazias) hein?**

Le Quiz 5

```
int i = 5;  
int *p;  
p = &i;  
*p += 2;  
printf( "%d", *p );
```

- a) 5
 - ✓ b) 7
 - c) endereço de i
 - d) endereço de p
- mãos vazias) hein?**

Le Quiz 6

```
void troca(int *a, int *b) {  
    int temp;  
    temp = *a;  
    *a = *b;  
    *b = temp;  
    printf("%d %d\n", *a, *b);  
}
```

```
int main(void) {  
    int x = 1;  
    int y = 2;  
    troca(&x, &y);  
    printf("%d %d\n", x, y);  
    return 0;  
}
```

a) 1 2 e 1 2

b) 1 2 e 2 1

c) 2 1 e 1 2

d) 2 1 e 2 1

mãos vazias) hein?

Le Quiz 6

```
void troca(int *a, int *b) {  
    int temp;  
    temp = *a;  
    *a = *b;  
    *b = temp;  
    printf("%d %d\n", *a, *b);  
}
```

```
int main(void) {  
    int x = 1;  
    int y = 2;  
    troca(&x, &y);  
    printf("%d %d\n", x, y);  
    return 0;  
}
```

a) 1 2 e 1 2

b) 1 2 e 2 1

c) 2 1 e 1 2

 d) 2 1 e 2 1

mãos vazias) hein?

Le Quiz 7

```
void troca(int *a, int *b) {  
    int *temp;  
    *temp = *a;  
    *a = *b;  
    *b = *temp;  
    printf("%d %d\n", *a, *b);  
}
```

```
int main(void) {  
    int x = 1;  
    int y = 2;  
    troca(&x, &y);  
    printf("%d %d\n", x, y);  
    return 0;  
}
```

a) 1 2 e 1 2

b) 1 2 e 2 1

c) 2 1 e 1 2

d) 2 1 e 2 1

mãos vazias) hein?

Le Quiz 7

```
void troca(int *a, int *b) {  
    int *temp;  
    *temp = *a;  
    *a = *b;  
    *b = *temp;  
    printf("%d %d\n", *a, *b);  
}
```

```
int main(void) {  
    int x = 1;  
    int y = 2;  
    troca(&x, &y);  
    printf("%d %d\n", x, y);  
    return 0;  
}
```

a) 1 2 e 1 2

b) 1 2 e 2 1

c) 2 1 e 1 2

d) 2 1 e 2 1

 **mãos vazias) hein?**

Le Quiz 8

```
void troca(int *a, int *b) {  
    int *temp;  
    temp = a;  
    a = b;  
    b = temp;  
    printf("%d %d\n", *a, *b);  
}
```

```
int main(void) {  
    int x = 1;  
    int y = 2;  
    troca(&x, &y);  
    printf("%d %d\n", x, y);  
    return 0;  
}
```

a) 1 2 e 1 2

b) 1 2 e 2 1

c) 2 1 e 1 2

d) 2 1 e 2 1

mãos vazias) hein?

Le Quiz 8

```
void troca(int *a, int *b) {  
    int *temp;  
    temp = a;  
    a = b;  
    b = temp;  
    printf("%d %d\n", *a, *b);  
}
```

```
int main(void) {  
    int x = 1;  
    int y = 2;  
    troca(&x, &y);  
    printf("%d %d\n", x, y);  
    return 0;  
}
```

a) 1 2 e 1 2

b) 1 2 e 2 1

 c) 2 1 e 1 2

d) 2 1 e 2 1

mãos vazias) hein?

Tipos compostos em C

```
struct ponto {  
    int x;  
    int y;  
}
```

```
int main(void) {
```

```
    return 0;  
}
```

um struct é bem mais simples que um objeto

```
struct ponto {  
    int x;  
    int y;  
}
```

```
int main(void) {  
    struct ponto a;
```

```
    return 0;  
}
```

a palavra `struct` é necessária na declaração


```
struct ponto {  
    int x;  
    int y;  
}
```

```
int main(void) {  
    struct ponto a;
```

```
    a.x = 1;  
    a.y = 2;
```

```
    return 0;  
}
```

acessamos através do operador .

```
struct ponto {  
    int x;  
    int y;  
}
```

```
int main(void) {  
    struct ponto a;
```

```
    a.x = 1;
```

```
    a.y = 2;
```

```
    printf("%d %d\n", a.x, a.y);
```

```
    return 0;
```

```
}
```

igualzinho a usar uma variável

```
struct ponto {  
    int x;  
    int y;  
}
```

```
int main(void) {  
    struct ponto a;  
  
    a.x = 1;  
    scanf("%d", &a.y);  
    printf("%d %d\n", a.x, a.y);  
  
    return 0;  
}
```

igualzinho mesmo!

```
struct ponto {  
    int x;  
    int y;  
}
```

```
int main(void) {  
    struct ponto a;  
  
    a.x = 1;  
    scanf("%d", &a.y);  
    printf("%d %d\n", a.x, a.y);  
  
    struct ponto b = a;  
    printf("%d %d\n", b.x, b.y);  
  
    return 0;  
}
```

isso vale?

```
struct ponto {  
    int x;  
    int y;  
}
```

```
int main(void) {  
    struct ponto a;  
  
    a.x = 1;  
    scanf("%d", &a.y);  
    printf("%d %d\n", a.x, a.y);  
  
    struct ponto b = a;  
    printf("%d %d\n", b.x, b.y);  
  
    return 0;  
}
```

sim, isso vale!

```
struct ponto {  
    int x;  
    int y;  
}
```

```
typedef struct ponto tiponto;
```

```
int main(void) {  
    tiponto a;  
  
    a.x = 1;  
    scanf("%d", &a.y);  
    printf("%d %d\n", a.x, a.y);  
  
    tiponto b = a;  
    printf("%d %d\n", b.x, b.y);  
  
    return 0;  
}
```

podemos usar typedef para definir um apelido

```
typedef struct ponto {  
    int x;  
    int y;  
} tiponto;
```

versão compacta

```
int main(void) {  
    tiponto a;  
  
    a.x = 1;  
    scanf("%d", &a.y);  
    printf("%d %d\n", a.x, a.y);  
  
    tiponto b = a;  
    printf("%d %d\n", b.x, b.y);  
  
    return 0;  
}
```

```
typedef struct {  
    int x;  
    int y;  
} tiponto;
```

```
int main(void) {  
    tiponto a;  
  
    a.x = 1;  
    scanf("%d", &a.y);  
    printf("%d %d\n", a.x, a.y);  
  
    tiponto b = a;  
    printf("%d %d\n", b.x, b.y);  
  
    return 0;  
}
```

nessa versão o struct pode ser anônimo


```
typedef struct {  
    int x;  
    int y;  
} ponto;
```

```
int main(void) {  
    ponto a;  
  
    a.x = 1;  
    scanf("%d", &a.y);  
    printf("%d %d\n", a.x, a.y);  
  
    ponto b = a;  
    printf("%d %d\n", b.x, b.y);  
  
    return 0;  
}
```

então podemos usar um nome menos feio

```
typedef struct {  
    int x;  
    int y;  
} ponto;
```

```
void inverte(ponto *a) {
```

```
}
```

agora o bicho pega

```
typedef struct {  
    int x;  
    int y;  
} ponto;
```

```
void inverte(ponto *a) {  
    int temp;  
    temp = a.x;  
    a.x = a.y;  
    a.y = temp;  
}
```

isso faz sentido?

```
typedef struct {  
    int x;  
    int y;  
} ponto;
```

```
void inverte(ponto *a) {  
    int temp;  
    temp = (*a).x;  
    (*a).x = (*a).y;  
    (*a).y = temp;  
}
```

agora faz sentido?

```
typedef struct {  
    int x;  
    int y;  
} ponto;  
  
void inverte(ponto *a) {  
    int temp;  
    temp = (*a).x;  
    (*a).x = (*a).y;  
    (*a).y = temp;  
}
```

não só faz sentido, como é muito comum

```
typedef struct {  
    int x;  
    int y;  
} ponto;
```

```
void inverte(ponto *a) {  
    int temp;  
    temp = a->x;  
    a->x = a->y;  
    a->y = temp;  
}
```

versão compacta

Le Quiz 9

```
typedef struct {  
    int *x;  
    int *y;  
} ponto;
```

```
void inverte(ponto *a) {  
    int *temp;  
    temp = a.x;  
    a.x = a.y;  
    a.y = temp;  
}
```


a) faz sentido

b) não faz sentido

Le Quiz 9

```
typedef struct {  
    int *x;  
    int *y;  
} ponto;
```

```
void inverte(ponto *a) {  
    int *temp;  
    temp = a.x;  
    a.x = a.y;  
    a.y = temp;  
}
```

- a)** faz sentido
-  **b)** não faz sentido

Le Quiz 10

```
typedef struct {  
    int *x;  
    int *y;  
} ponto;
```

```
void inverte(ponto *a) {  
    int *temp;  
    temp = *a.x;  
    *a.x = *a.y;  
    *a.y = temp;  
}
```


a) faz sentido

b) não faz sentido

Le Quiz 10

```
typedef struct {  
    int *x;  
    int *y;  
} ponto;
```

```
void inverte(ponto *a) {  
    int *temp;  
    temp = *a.x;  
    *a.x = *a.y;  
    *a.y = temp;  
}
```

- a)** faz sentido
-  **b)** não faz sentido

```
typedef struct {  
    int *x;  
    int *y;  
} ponto;
```

```
void inverte(ponto *a) {  
    int *temp;  
    temp = *(a.x);  
    *(a.x) = *(a.y);  
    *(a.y) = temp;  
}
```

Le Quiz 11

```
typedef struct {  
    int *x;  
    int *y;  
} ponto;
```

```
void inverte(ponto *a) {  
    int *temp;  
    temp = (*a).x;  
    (*a).x = (*a).y;  
    (*a).y = temp;  
}
```

a) faz sentido

b) não faz sentido

Le Quiz 11

```
typedef struct {  
    int *x;  
    int *y;  
} ponto;
```

```
void inverte(ponto *a) {  
    int *temp;  
    temp = (*a).x;  
    (*a).x = (*a).y;  
    (*a).y = temp;  
}
```

- ✓ a) faz sentido
b) não faz sentido

Le Quiz 12

```
typedef struct {  
    int *x;  
    int *y;  
} ponto;
```

```
void inverte(ponto *a) {  
    int *temp;  
    temp = *((*a).x);  
    *((*a).x) = *((*a).y);  
    *((*a).y) = temp;  
}
```


a) faz sentido

b) não faz sentido

Le Quiz 12

```
typedef struct {  
    int *x;  
    int *y;  
} ponto;
```

```
void inverte(ponto *a) {  
    int *temp;  
    temp = *((*a).x);  
    *((*a).x) = *((*a).y);  
    *((*a).y) = temp;  
}
```

- a) faz sentido
-  b) não faz sentido

```
typedef struct {  
    int *x;  
    int *y;  
} ponto;
```

```
void inverte(ponto *a) {  
    int temp;  
    temp = *((*a).x);  
    *((*a).x) = *((*a).y);  
    *((*a).y) = temp;  
}
```



```
typedef struct {  
    int *x;  
    int *y;  
} ponto;
```

```
void inverte(ponto *a) {  
    int temp;  
    temp = *(a->x);  
    *(a->x) = *(a->y);  
    *(a->y) = temp;  
}
```



**Agora preparem-se,
pois vai ficar bizarro**

```
int v[] = {11, 13, 17, 19, 23};
```

```
int *p;
```

```
int v[] = {11, 13, 17, 19, 23};
```

```
int *p;
```

```
p = v;
```

isso faz sentido?

```
int v[] = {11, 13, 17, 19, 23};  
  
int *p;  
  
p = v;  
  
printf("%d\n", p[0]);
```

qual será a saída?

```
int v[] = {11, 13, 17, 19, 23};  
  
int *p;  
  
p = v;  
  
printf("%d\n", p[0]);  
  
printf("%d\n", *p);
```

qual será a saída?

```
int v[] = {11, 13, 17, 19, 23};  
  
int *p;  
  
p = v;  
  
printf("%d\n", p[0]);  
  
printf("%d\n", *p);  
  
printf("%d\n", *(p + 2));
```

qual será a saída?

Quando um parâmetro é vetor,
ele é tratado de “maneira especial”?

Não!

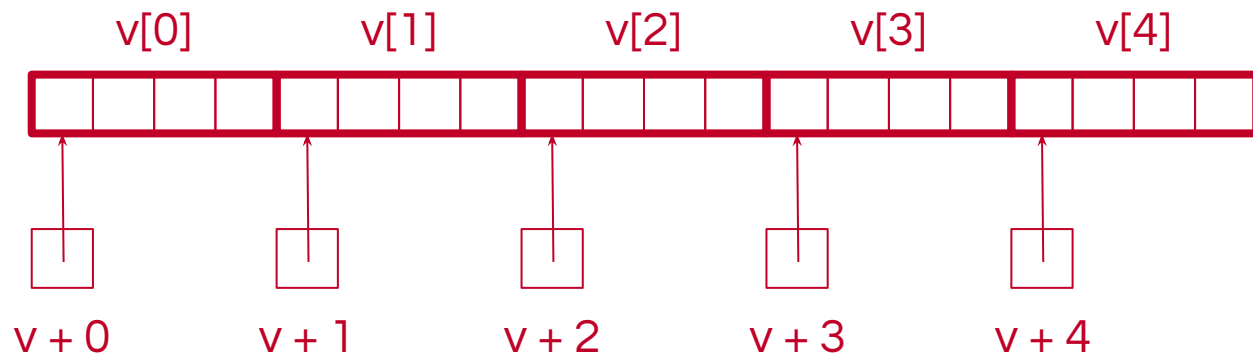
VETORES
são
APONTADORES

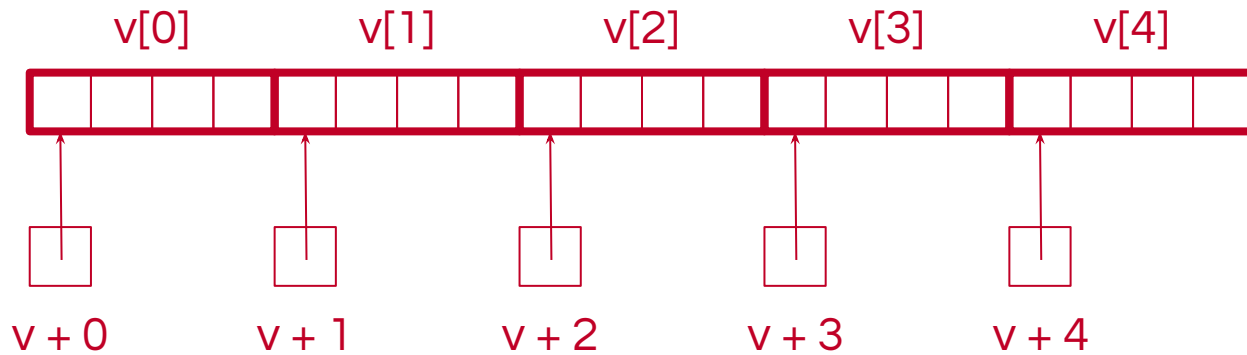


1 byte



4 bytes





somar `x` a um apontador significa somar `x * sizeof(int)`,
ou seja, deslocamos o apontador para o “próximo” inteiro

$v[0]$	equivale a	$*(v + 0)$
$v[1]$	equivale a	$*(v + 1)$
$v[2]$	equivale a	$*(v + 2)$
$v[3]$	equivale a	$*(v + 3)$
$v[4]$	equivale a	$*(v + 4)$

vocês conseguem entender agora o

```
int main(int argc, char **argv)
```

OBRIGADO