Programação Procedimental

Ponteiros

Aula 07

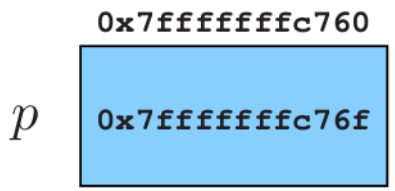
Prof. Felipe A. Louza

Roteiro

Ponteiros

Um ponteiro (apontador) é um tipo especial de variável que armazena um endereço.

- Um ponteiro pode ter o valor `NULL` (ou zero) que é um endereço inválido.



Ponteiros

Declaramos uma variável ponteiro com o operador *:

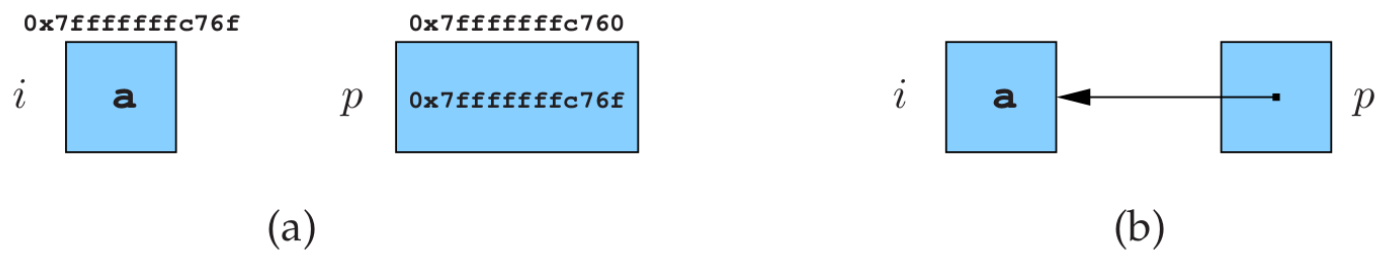
```c
#include <stdio.h>

int main(){
   ...
   int *p;
   ...
return 0;
}
```

- Precisamos especificar o tipo do apontador.
- Essa declaração indica que p é uma variável ponteiro capaz de apontar para variáveis do tipo `int`.

Ponteiros

Se um ponteiro `p` armazena o endereço de uma variável `i`, dizemos que `p` aponta para `i`.

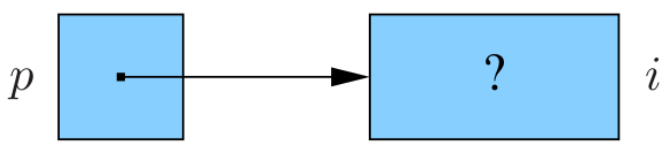
```
1  #include <stdio.h>
2
3  int main(){
4    ...
5    char i, *p;
6    ...
7    p = &i;
8    ...
9  return 0;
10 }
```



Ponteiros

Para acessar o valor da variável apontada por um ponteiro, utilizamos o operador *:

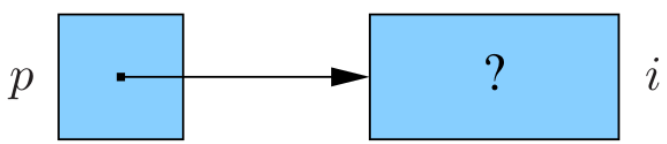
```c
#include <stdio.h>

int main(){
   ...
   int i=10, *p;
   p = &i;
   ...
   printf("%d\n", *p);
   ...
return 0;
}
```



Ponteiros

Todas as alterações feitas em *p são feitas, na verdade, na variável apontada:

```c
#include <stdio.h>

int main(){
   ...
   int i=10, *p;
   p = &i;
   *p = 101;
   printf("%d\n", i);
   ...
return 0;
}
```



Operadores de Ponteiro

Exemplo 1:

```c
#include <stdio.h>

int main(){
  int b;
  int *c;

  b=10;
  c=&b;
  *c=11;

  printf("\n%d\n",b);

return 0;
}
```

- O que será impresso??

Operadores de Ponteiro

Exemplo 2:

```c
#include <stdio.h>

int main(){
  int num, q=1;
  int *p;

  num=100;
  p = &num;
  q = *p;

  printf("%d",q);

return 0;
}
```

- O que será impresso??

Cuidado!

Cuidado:

- Não se pode atribuir um valor para o endereço apontado pelo ponteiro sem antes ter certeza de que o endereço é válido:

```
1  int a,b;
2  int *c;
3
4  b=10;
5  *c=13; //Vai armazenar 13 em qual endereço?
```

Ponteiros

Em geral, sempre inicializamos um ponteiro com o valor NULL.

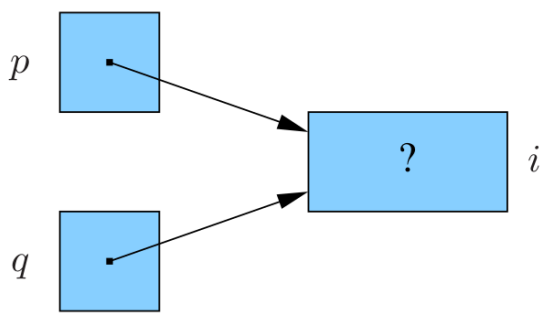
```
1  int a,b;
2  int *c = NULL;
3
4  b=10;
5  if(c!=NULL) *c=13;
```

- Com isso, podemos verificar se o ponteiro é válido:

Ponteiros

Ponteiros podem ser copiados:

```c
#include <stdio.h>

int main(){
   ...
   int i = 10, *p = NULL, *q = NULL;
   p = &i;
   q = p;
   *q = 101;
   ...
return 0;
}
```



Declaração de apontadores em C

Cuidado ao declarar vários apontadores em uma única linha.

```
1 int *ap;
2 int *ap1, *ap_2, *ap_3;
```

```
1 int* ap;
2 int* ap1, ap_2, ap_3;
```

Exemplo 3

```c
#include <stdio.h>

int main() {
  char c, * p;
  p = &c;
  c = 'a';

  printf("&c = %p   c = %c\n", &c, c);
  printf("&p = %p   p = %p    *p = %c\n\n", &p, p, *p);

  c = '/';
  printf("&c = %p   c = %c\n", &c, c);
  printf("&p = %p   p = %p    *p = %c\n\n", &p, p, *p);

  *p = 'Z';
  printf("&c = %p   c = %c\n", &c, c);
  printf("&p = %p   p = %p    *p = %c\n\n", &p, p, *p);

return 0;
}
```

Exemplo 4

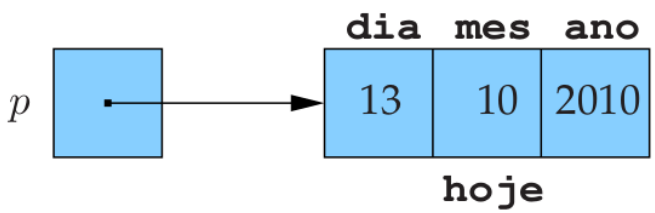
```c
#include <stdio.h>

int main(){
  int a, b, *ptr1, *ptr2;
  ptr1 = &a;
  ptr2 = &b;
  a = 1;

  (*ptr1)++;
  b = a + *ptr1;
  *ptr2 = *ptr1 * *ptr2;

  printf("a=%d, b=%d, *ptr1=%d, *ptr2=%d\n", a, b, *ptr1, *ptr2);

return 0;
}
```

Roteiro

Ponteiros

Ponteiros podem apontar para registros (structs).

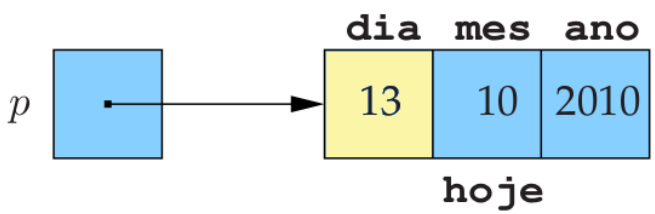
```
1  typedef struct{
2    int dia, mes, ano;
3  } t_data;
4  ...
5  t_data hoje;
6  t_data *p = &hoje;
7  ...
```



Ponteiros

Para acessar os campos da estrutura utilizamos o operador ->.

```
1 typedef struct{
2   int dia, mes, ano;
3 } t_data;
4 ...
5 t_data hoje;
6 t_data *p = &hoje;
7 p->dia = 13;
```

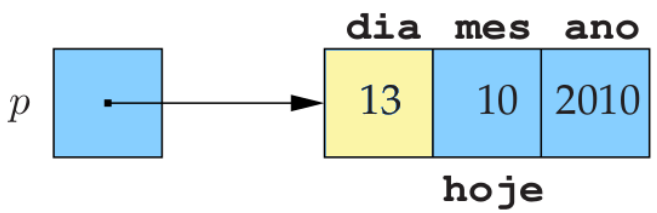


Ponteiros

O operador -> é equivalente à `(*p).dia`.

```
1 typedef struct{
2   int dia, mes, ano;
3 } t_data;
4 ...
5 t_data hoje;
6 t_data *p = &hoje;
7 (*p).dia = 13 //p->dia = 13;
```

– Os parênteses são necessários pois o operador * tem prioridade menor que o operador .



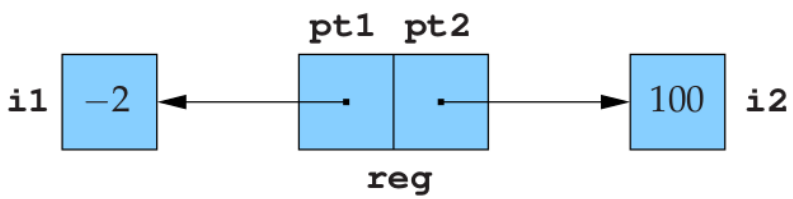
Exemplo 5

```c
#include <stdio.h>

typedef struct{
  int dia, mes, ano;
} t_data;

int main(){

  t_data hoje, * p;

  p = &hoje;
  p->dia = 13;
  p->mes = 10;
  p->ano = 2010;

  printf("A data de hoje é %d/%d/%d\n", hoje.dia, hoje.mes, hoje.ano);

return 0;
}
```

Ponteiros

Registros podem conter ponteiros:

```
1  typedef struct{
2    int *pt1, *pt2;
3  } t_reg;
4  ...
5  int i1 = -2, i2 = 100;
6  ...
7  t_reg reg;
8  reg.pt1 = &i1;
9  reg.pt2 = &i2;
10 ...
```



Exemplo 5

```c
#include <stdio.h>

typedef struct{
  int *pt1, *pt2;
} t_reg;

int main(){

  int i1 = 0, i2 = 100;

  t_reg reg;
  reg.pt1 = &i1;
  reg.pt2 = &i2;

  *reg.pt1 = -2;// equivalente *(reg.pt1) = -2, prioridade . *

  printf("i1 = %d, *reg.pt1 = %d\n", i1, *reg.pt1);
  printf("i2 = %d, *reg.pt2 = %d\n", i2, *reg.pt2);

return 0;
}
```

Roteiro

Passagem de argumentos por valor

Quando passamos argumentos a uma função, os valores fornecidos são copiados para os parâmetros (idêntico a uma atribuição).

```
1  void funcao(int a) {
2    a = a + 1;
3  }
4  ...
5  int main(){
6  ...
7    int a=1;
8    funcao(a);
9  ...
10 }
```

- Alterações nos parâmetros dentro da função não alteram os valores que foram passados.

Passagem de argumentos por valor

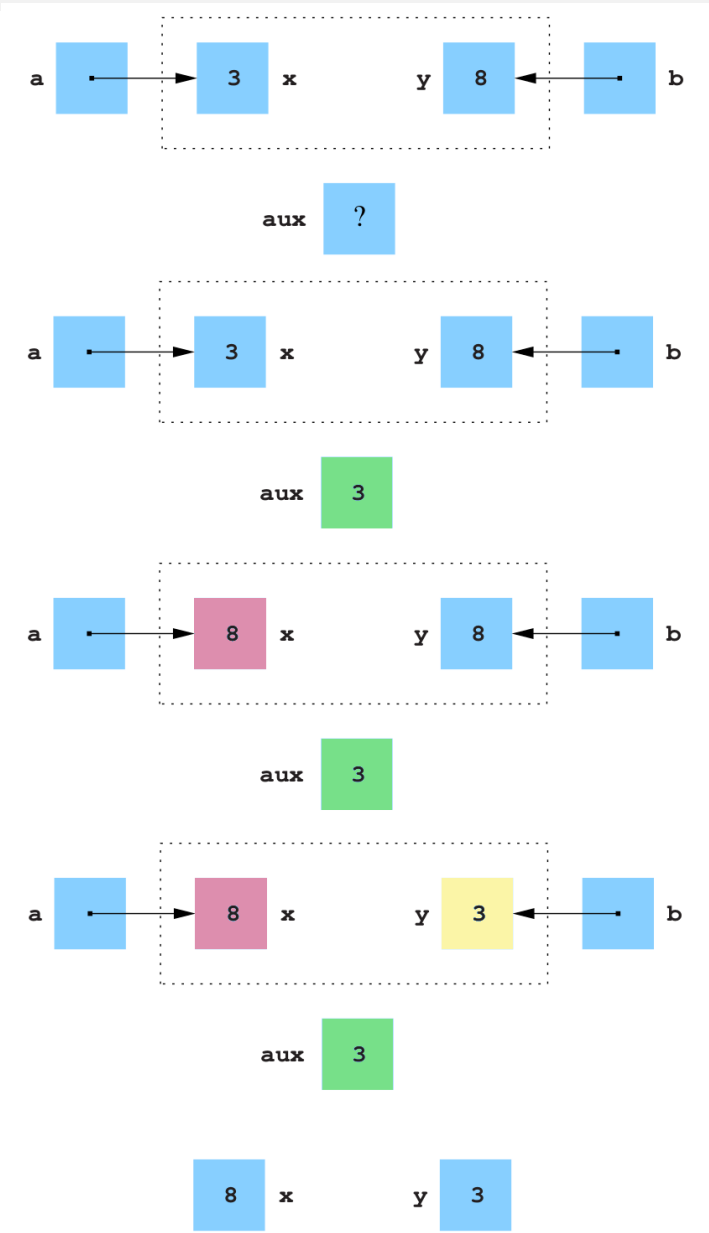
Podemos passar o endereço de uma variável como argumento para uma função:

```
1  void funcao(int *a) {
2    *a = *a + 1;
3  }
4  ...
5  int main(){
6  ...
7    int a=0;
8    funcao(&a);
9  ...
10 }
```

- Alterações realizadas no conteúdo das variáveis (não é feita cópia de valores).

Passagem de argumentos por valor

```c
#include <stdio.h>

void troca(int *a, int *b){
  int aux = *a;
  *a = *b;
  *b = aux;
}

int main(){

  int x=3, y=8;
  troca(&x, &y);
  printf("%d\t%d\n", x, y);

return 0;
}
```



Roteiro

Funções que retornam ponteiros

Funções podem retornar variáveis do tipo ponteiro:

```
1 int *max(int *a, int *b){
2
3   if (*a > *b)
4     return a;
5   else
6     return b;
7 }
```

- Precisamos especificar o tipo do ponteiro.

Funções que retornam ponteiros

Exemplo:

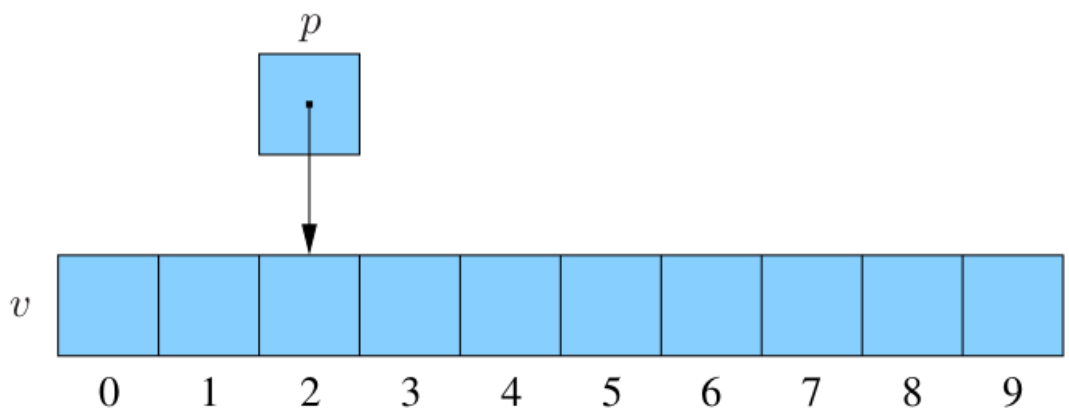
```c
#include <stdio.h>

int *max(int *a, int *b){
  if (*a > *b)
    return a;
  else
    return b;
}

int main(){

  int i=0, j=10;
  int *p = max(&i, &j);
  *p = -1;
  printf("%d\t%d\n", i, j);

return 0;
}
```

Roteiro

Vetores e ponteiros

Ponteiros podem apontar para uma posição qualquer de um vetor:

```
1 int v[10];
2 ...
3 int *p;
4 p = &v[2];
```

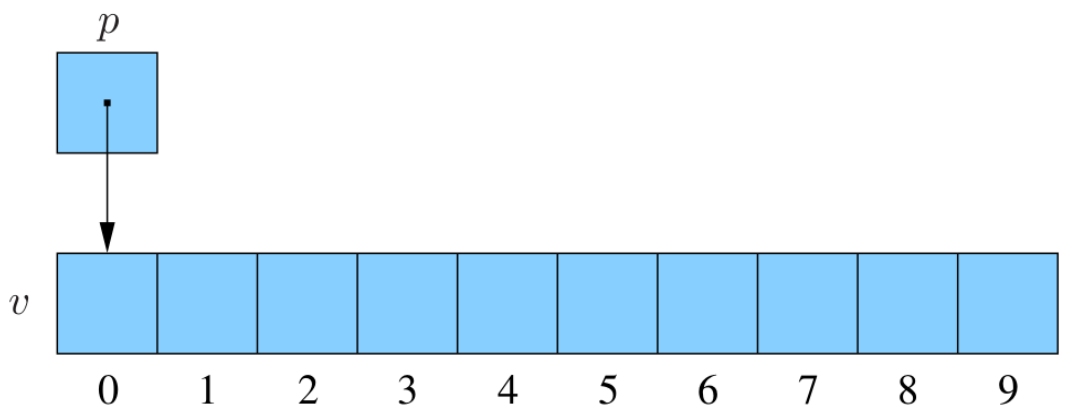


Vetores e ponteiros

O endereço da primeira posição de um vetor (`&v[0]`) pode ser abreviado pelo nome do vetor (`v`).

```
1 int *p;
2 p = &v[0];
```

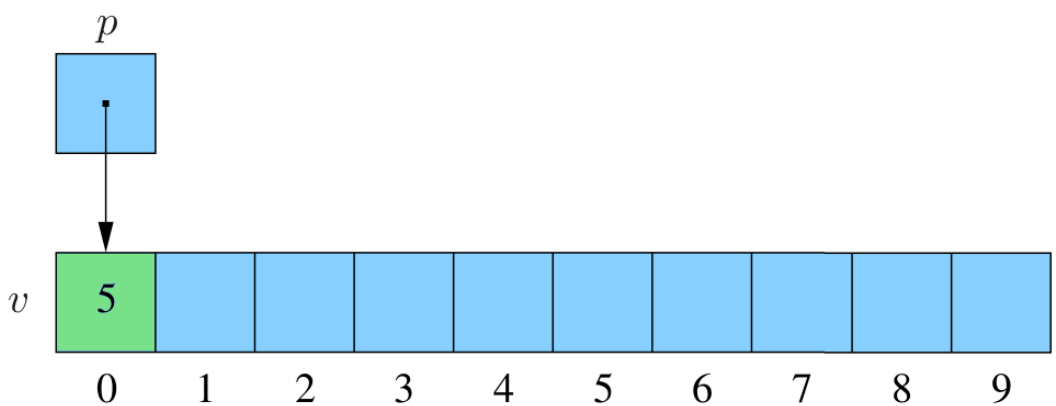
```
1 int *p;
2 p = v;
```



Vetores e ponteiros

Podemos acessar/alterar o valor de uma posição apontada com o operador *.

```
1 int v[10];
2 int *p = v;
3 ...
4 *p = 5;
```



Roteiro

Aritmética de ponteiros

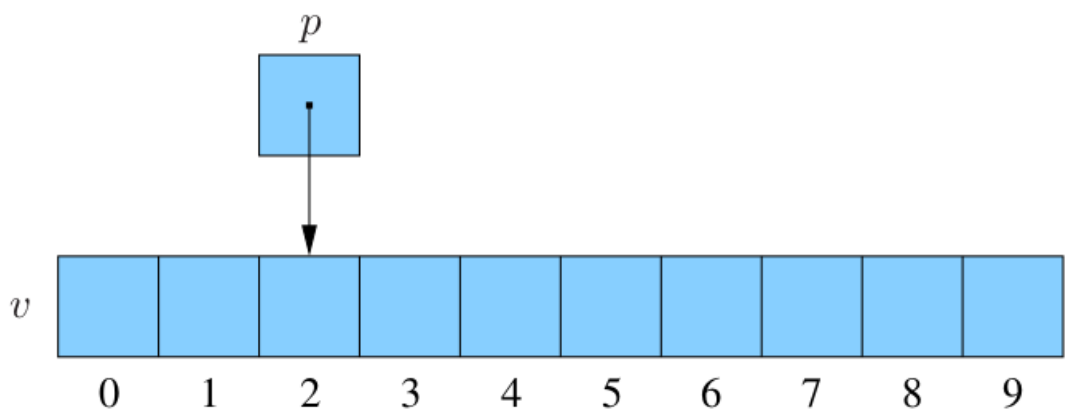
Podemos somar/subtrair valores inteiros com um ponteiro `p`:

- Lembre-se: vetores são armazenados em posições consecutivos na memória do computador.
- Se `p` aponta para `&v[i]`, então `p+j` aponta para `&v[i+j]`.

Aritmética de ponteiros

Exemplo:

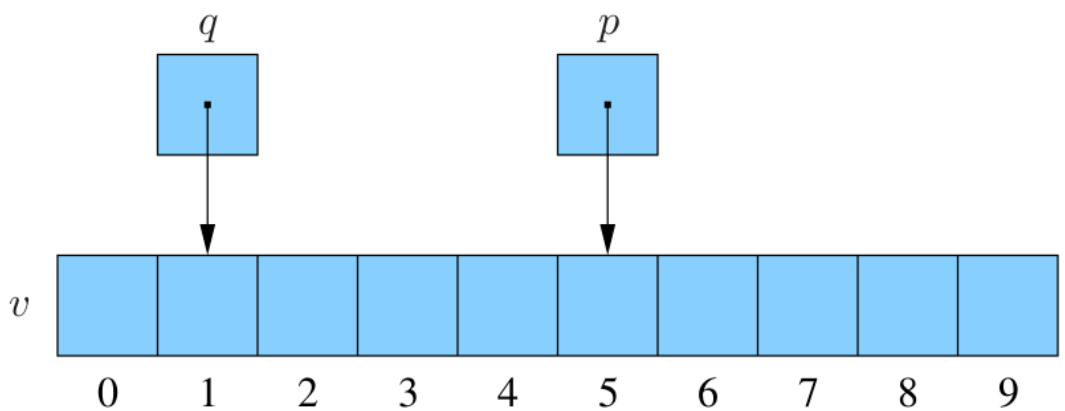
```
1 int v[10];
2 int *p = v;
3 p = p+2;
```



Aritmética de ponteiros

Outro exemplo:

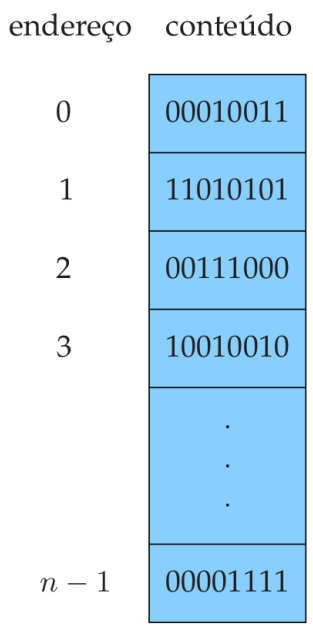
```
1 int v[10];
2 int *q = v;
3 q = q+1;
4 ...
5 int *p = q+4;
```



Vetores e ponteiros

Memória do Computador:

- Cada elemento de um vetor de `int` ocupa 4 bytes na memória.
- Ao somar `1` em um ponteiro do tipo `int *`, estamos "pulando" 4 bytes na memória.

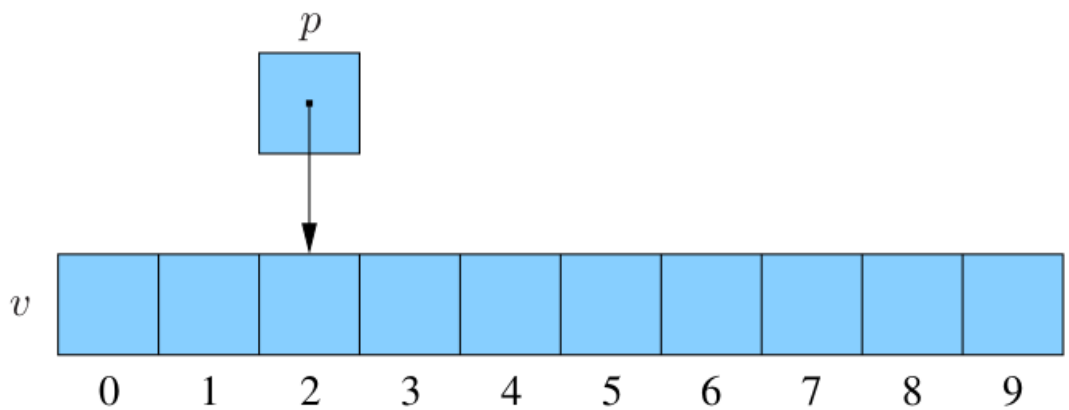


Vetores e ponteiros

Compilador:

- O compilador ajusta os detalhes internos de modo a criar uma ilusão de que a diferença entre `v[i]` e `v[i+1]` seja 1, qualquer que seja o valor em bytes.

```
1 int v[10];
2 int *p = v;
3 p = p+2;
```



Vetores e ponteiros

- As expressões `*(v + i)` e `v[i]` têm exatamente o mesmo valor.

```
1 int i=10;
2 *(v+i) = 789;
3 v[i] = 789;
```

- Da mesma forma, `(v + i)` e `&v[i]` são equivalente

```
1 for (i = 0; i < 100; i++)  scanf("%d", &v[i]);
2 for (i = 0; i < 100; i++)  scanf("%d", v + i);
```

Vetores e ponteiros

Dado um vetor `v[100]` e um ponteiro `int *p=v;`

```
1 int v[100];
2 int *p = v;
```

- Qual a diferença?

```
1 for (i = 0; i < 100; i++)
2   scanf("%d", &v[i]);
```

```
1 for (i = 0; i < 100; i++)
2   scanf("%d", v + i);
```

```
1 for (p = v; p < &v[100]; p++)
2   scanf("%d", p);
```

Roteiro

Passagem de argumentos por valor

Passagem de um vetor como argumento para uma função:

```
1 int soma(int v[], int n) {
2   int i, sum=0;
3   for(i=0; i<n; i++) sum+=v[i];
4 return sum;
5 }
```

```
1 int v[100];
2 ...
3 soma(v, n);
```

Passagem de argumentos por valor

Alternativamente, podemos passar o endereço de um vetor como argumento para uma função:

```
1 int soma(int *v, int n) {
2   int i, sum=0;
3   for(i=0; i<n; i++) sum+=v[i]; //v[i] == *(v+i)
4 return sum;
5 }
```

```
1 int v[100];
2 ...
3 soma(v, n);
```

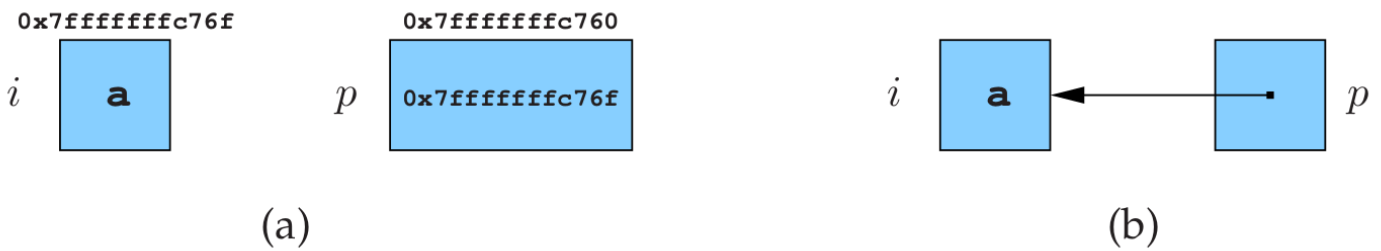
Roteiro

Ponteiros para ponteiros

Um ponteiro é um tipo especial de variável que armazena um endereço.

- Podemos acessar o conteúdo de uma posição de memória através de forma indireta.

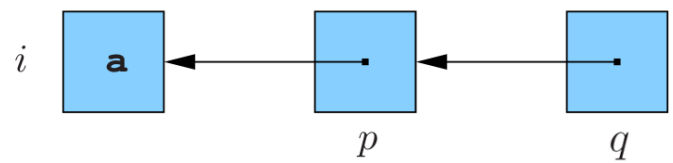
```
1  ...
2  char i, *p;
3  p = &i;
4  *p = 'a'
5  ...
```



Ponteiros para ponteiros

Podemos declarar um ponteiro para ponteiro:

```
1  ...
2  char i, *p;
3  p = &i;
4  char **q;
5  q = &p;
6  ...
```

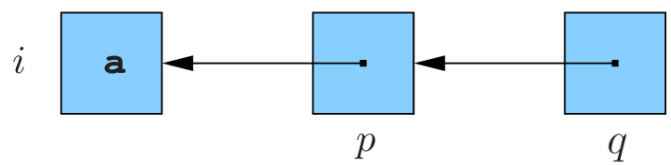


- O ponteiro `q` armazena o endereço de `p`,que por sua vez, armazena o endereço de `i`.

Ponteiros para ponteiros

Para acessar o conteúdo de um ponteiro para ponteiro utilizamos o operador **:

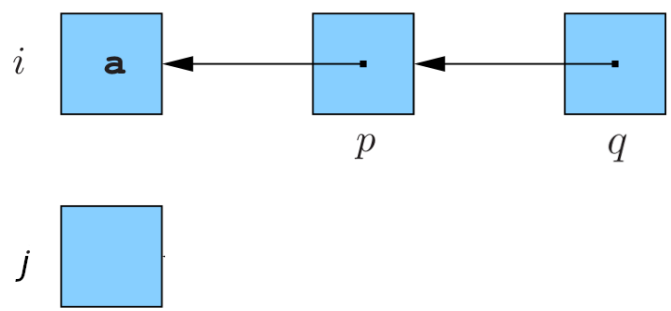
```
1  ...
2  char i, *p;
3  p = &i;
4  char **q;
5  q = &p;
6  ...
7  **q = 'a'; //equivalente à *p = 'a'
```



Ponteiros para ponteiros

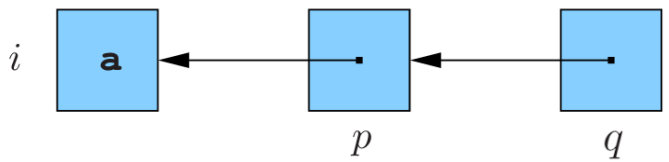
Cuidado ao alterar o valor de *q:

```
1  ...
2  char i, j, *p;
3  p = &i;
4  char **q;
5  q = &p;
6  ...
7  //muda o endereço que p está apontando
8  *q = &j; //equivalente à p = &j
```



Ponteiros para ponteiros

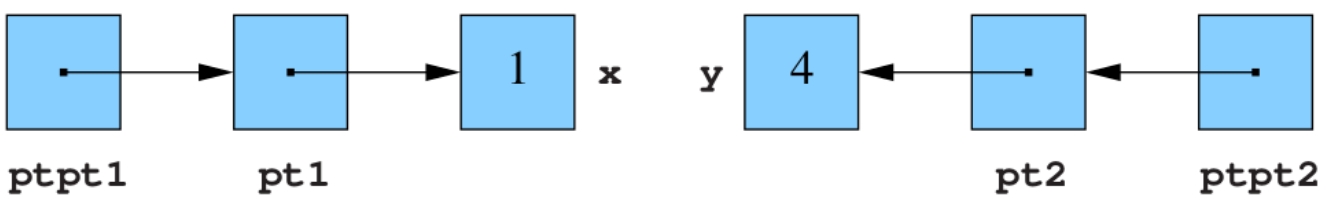
- Ponteiros para ponteiros têm diversas aplicações na linguagem C, especialmente no uso de listas ligadas.



Ponteiros para ponteiros

Exemplo:

```c
#include <stdio.h>

int main(void) {
  int x, y, *pt1, *pt2, **ptpt1, **ptpt2;
  x = 1;
  y = 4;
  printf("x=%d y=%d\n", x, y);
  pt1 = &x;
  pt2 = &y;
  printf("*pt1=%d *pt2=%d\n", *pt1, *pt2);
  ptpt1 = &pt1;
  ptpt2 = &pt2;
  printf("**ptpt1=%d **ptpt2=%d\n", **ptpt1, **ptpt2);
return 0;
}
```



Fim

Dúvidas?

Roteiro

Referências

1. Materiais adaptados da apostila de Programação de Computadores do Prof. Martinez, da UFMS.