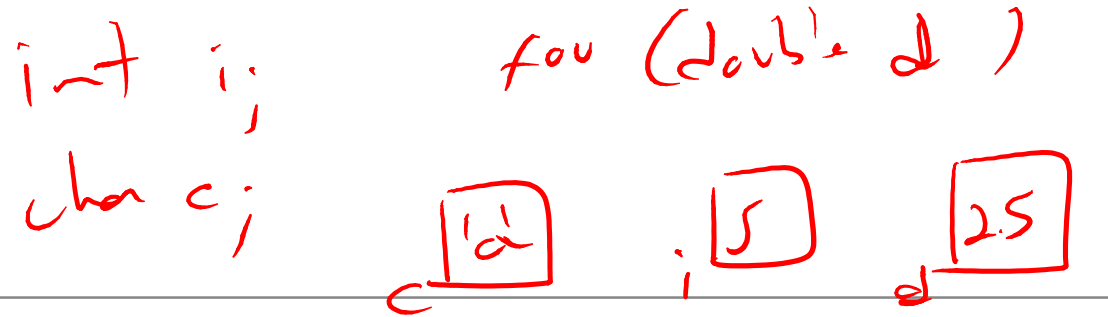


Introdução à Programação C

Fabio Mascarenhas - 2014.2

<http://www.dcc.ufrj.br/~fabiom/introc>

Ponteiros

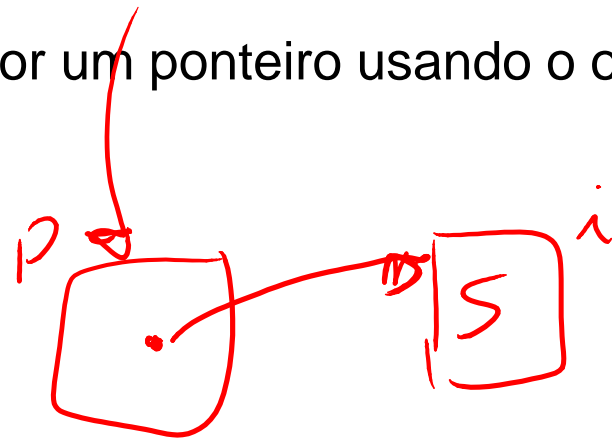


- Um *ponteiro* é uma variável que guarda um *endereço* para uma das caixas da memória
- Ponteiros para caixas de tipo `int` têm tipo `int*`, ponteiros para caixas de tipo `char` têm tipo `char*`, ponteiros para caixas de tipo `double` têm tipo `double*`
- Podemos pegar um ponteiro para qualquer variável ou posição de um vetor usando o operador `&` (“endereço de”)

`int *p = &i;` `char *pc = &s[10];`

- Podemos ler ou escrever o valor apontado por um ponteiro usando o operador `*` (“dereferência”)

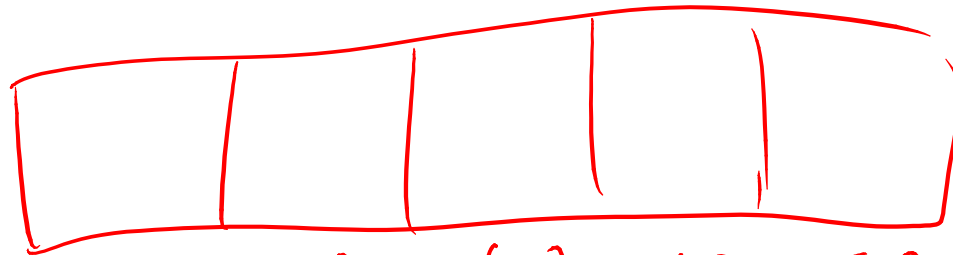
`printf("%d\n", *p);`
`*p = 10;` /* i agora é 10 */



Vetores

- Uma variável vetor não é uma única caixa, mas uma coleção de caixas, uma para cada elemento do vetor
- Os endereços dessas caixas são consecutivos

`int xs[5];`



`x[0] x[1] x[2] x[3] x[4]`
`(50) (51) (52) (53) (54)`

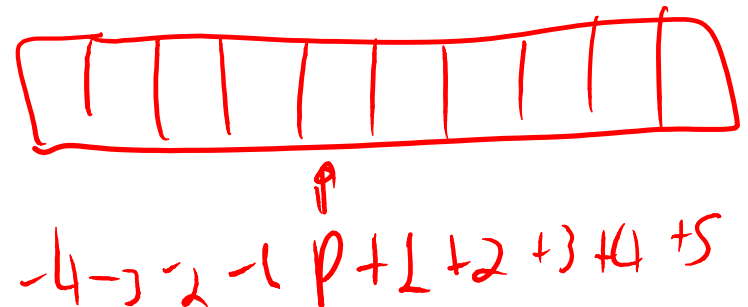
Ponteiros e vetores

- Podemos percorrer um vetor usando um ponteiro, inicializando ele com o endereço do primeiro elemento e depois incrementando o ponteiro
- Como uma conveniência, a variável do vetor também é o endereço do primeiro elemento:

```
int v[] = { 1, 2, 3, 4, 0 };
int *p = v;
int s = 0;
while(*p != 0) {
    s = s + *p;
    p = p + 1;
}
```

p é o endereço do 1º elemento

avance p p/ o próximo el.



Ponteiros e vetores

- Se uma função tem um ponteiro como parâmetro, podemos passar um vetor para ela

```
static void print_vetor(int *p, int n);
```

```
int v[] = { 1, 2, 3, 4 };  
print_vetor(v, 4);
```

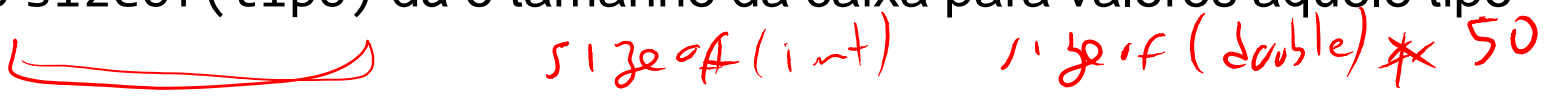

- Também podemos passar um ponteiro para uma função que espera um vetor sem indicação do tamanho

```
static void print_vetor(int v[], int n);
```

```
int *p = naturais(4);  
print_vetor(p, 4);
```

- Usar a forma `int v[]` deixa mais claro que a função espera um vetor

malloc e free

- Cada caixa da memória tem um tamanho, dado pelo tipo dos valores que aquela caixa guarda
- A operação `sizeof(tipo)` dá o tamanho da caixa para valores daquele tipo

- A função `malloc` pede para o sistema uma nova caixa (ou coleção de caixas) de determinado tamanho, e retorna o endereço para essa caixa (ou para a primeira caixa da coleção)
- Essa caixa (ou coleção de caixas) existe até que o endereço dela (ou da primeira caixa da coleção) seja passado para a função `free`


Alocando e retornando um “vetor”

- Podemos simular a criação de um vetor com tamanho determinado em tempo de execução usando um ponteiro:

```
/* retorna um vetor com os primeiros n
   números naturais */
static int* naturais(int n) {
    int *v = malloc(n * sizeof(int));
    int i = 0;
    while(i < n) {
        v[i] = i + 1;
        i = i + 1;
    }
    return v;
}
```

- Apesar de v não ser um vetor, podemos indexá-lo como se fosse, e passar v para funções que esperam um vetor

Cadeias de caracteres

- Cadeias de caracteres são vetores de caracteres terminados pelo caractere 0, mas funções de cadeias de caracteres normalmente usam `char*` para indicar uma cadeia de caracteres
- Muitas vezes as funções também retornam cadeias, então usar sempre `char*` deixa a assinatura mais consistente, já que não se pode declarar um tipo de retorno vetor
- Como cadeias são sempre terminadas com 0, uma forma normal de percorrer os caracteres de uma cadeia nessas funções é incrementar o próprio ponteiro até encontrar 0

Exemplo - comprimento

- A função abaixo é equivalente a `strlen`, e retorna o comprimento da cadeia de caracteres passada como argumento:

```
static int comprimento(char *s) {  
    int n = 0;  
    while(*s) {  
        n = n + 1;  
        s = s + 1;  
    }  
    return n;  
}
```

- Podemos usar um laço `for` para escrever a função de modo mais compacto:

```
static int comprimento(char *s) {  
    int n;  
    for(n = 0; *s; n++, s++) {}  
    return n;  
}
```



Exemplo - busca

- Usar `char*` para cadeias de caracteres tem a vantagem de deixar o código mais uniforme quando temos funções que recebem e retornam cadeias
- Um exemplo é uma função que busca um caractere em uma cadeia, retornando o sufixo da cadeia começando naquele caractere (ou uma cadeia vazia se o caractere não foi encontrado):

```
static char* busca(char *s, char c) {  
    for(; *s && *s != c; s++) {}  
    return s;  
}
```

→ o endereço de c em s ou o endereço do \emptyset

busca("batata", 't') → "tata"

- Se `s` é uma cadeia não vazia, então `(s+1)` também é uma cadeia!