Introdução à Programação C

Fabio Mascarenhas - 2014.2

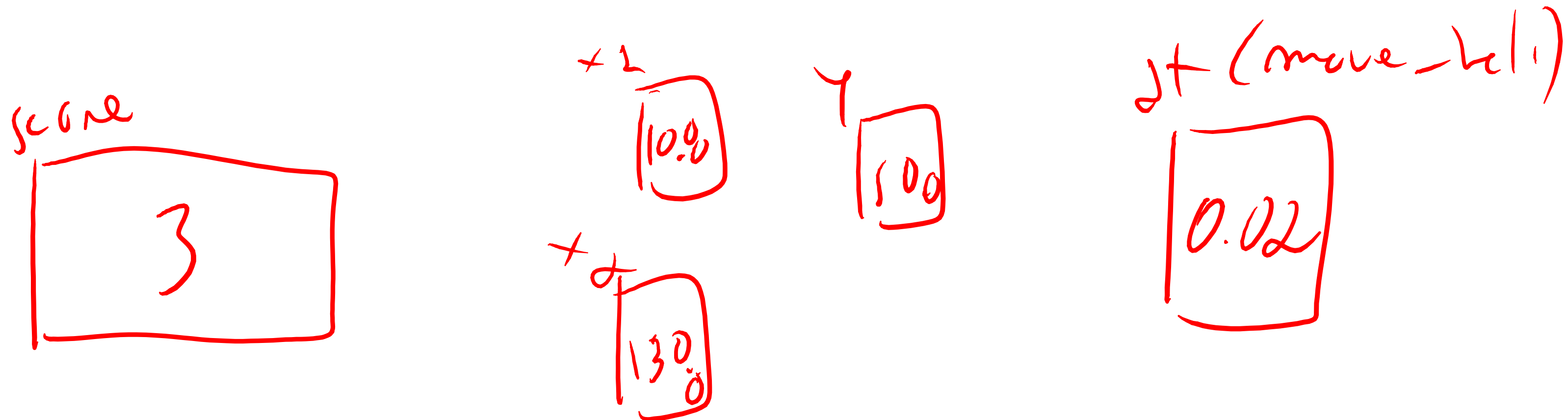
http://www.dcc.ufrj.br/~fabiom/introc

Caixas

- Podemos pensar em uma variável como uma caixa que guarda um valor

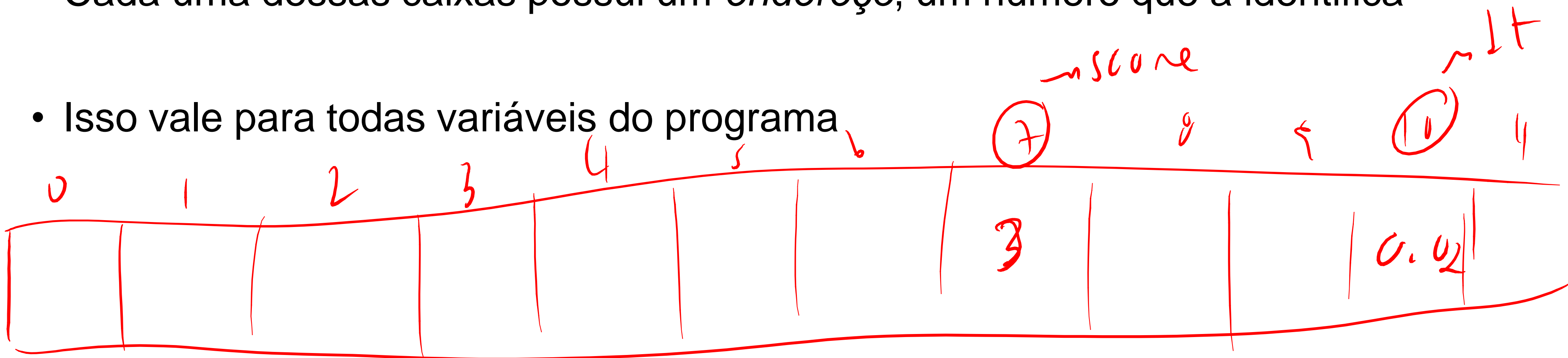
- Quando atribuímos à variável, guardamos um novo valor na caixa, e quando usamos lemos o valor que está lá naquele momento

- Parâmetros de funções também são variáveis; quando chamamos uma função, colocamos o valor do argumento na caixa do parâmetro correspondente



Endereços

- A memória do computador é uma coleção de inúmeras caixas
- Cada uma dessas caixas possui um *endereço*, um número que a identifica
- Isso vale para todas variáveis do programa



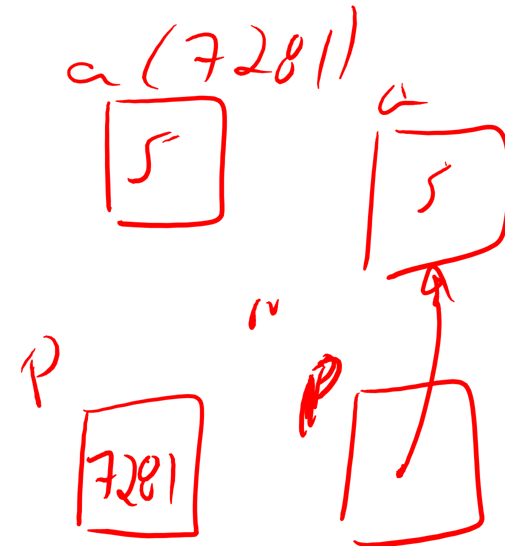
Ponteiros

- 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*`

- Mas como inicializamos uma variável ponteiro? Uma maneira é pegar o endereço de alguma outra variável:

```
int a = 5;
int *p = &a; /* p é um ponteiro */
```



Dereferência

- O valor de uma variável ponteiro é o endereço de alguma caixa na memória

- Podemos acessar a caixa que corresponde a esse ponteiro com operações de *dereferência*

  "valor apontado por"

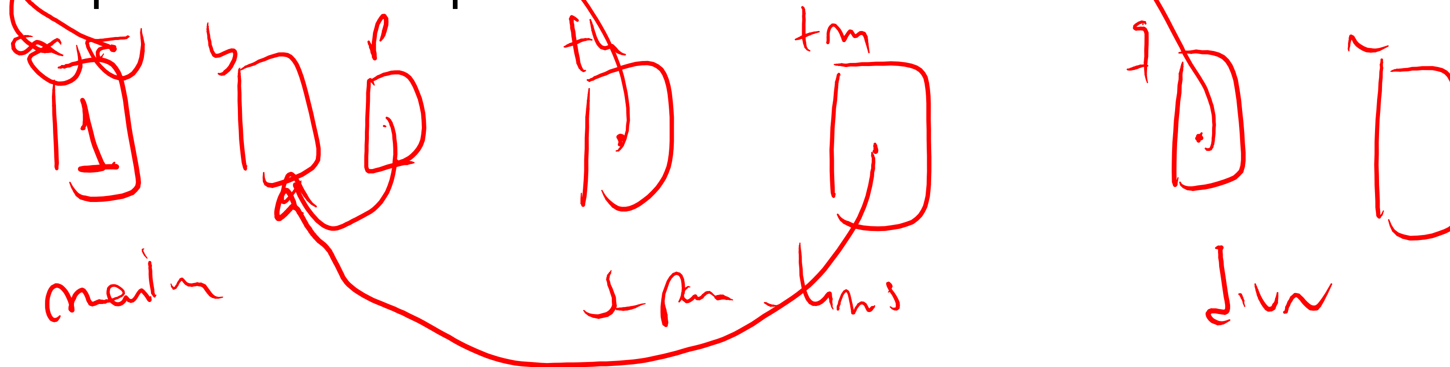
- Se p é um ponteiro, usamos *p em uma expressão para ler o valor da caixa para a qual ele aponta

- Usamos *p do lado esquerdo de uma atribução para guardar um novo valor nessa caixa

```
printf("%d\n", *p); /* imprime 5 */
*p = 2;
printf("%d\n", a); /* imprime 2 */
```

Parâmetros de saída, de novo

- Os “parâmetros de saída” que vimos anteriormente na verdade são ponteiros!
- Em C, uma função sempre só retorna no máximo um valor, mas podemos contornar isso passando o *endereço* de alguma variável para a função
- De posse desse endereço, a função pode atribuir um novo valor a essa variável, e assim “retornar” vários valores
- Por isso usamos * na declaração e uso de “parâmetros de saída”, e usamos & para dizer em quais variáveis esses valores deveriam ser armazenados!



c (15) b (50)

nome-igual

1400 s (120)

h m

15 50

h-para-hms

c 5000 b 3600

15 120

div

Erros com ponteiros

- Um erro comum com ponteiros é usar um ponteiro sem inicializá-lo:

```
int *p;
*p = 2;
```

→ CRASH!

- Isso quase sempre faz o programa terminar de modo abrupto

- Outro erro comum é usar um ponteiro depois que a caixa para a qual ele aponta já não existe mais

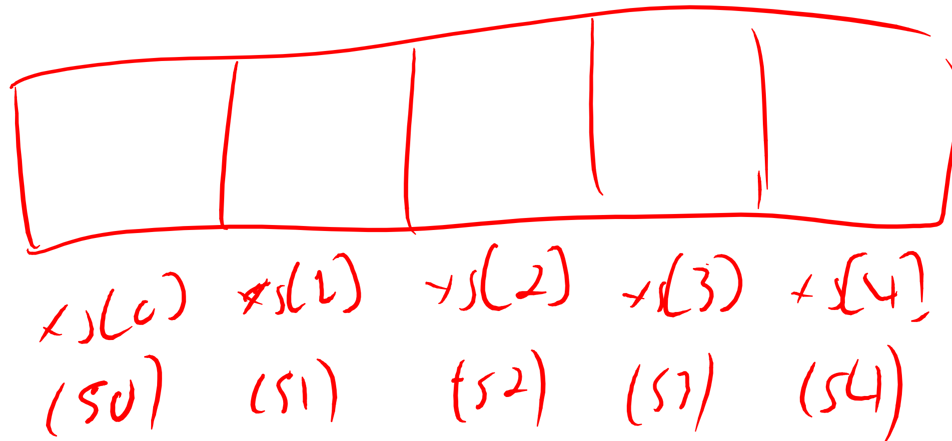
- Quando saímos de uma função as caixas correspondentes às variáveis locais e parâmetros deixam de existir, então quaisquer ponteiros para elas ficam inválidos

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];



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:

sentinela

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

aritmética de ponteiros

Alocação dinâmica

- Até agora, toda memória do programa ou está em variáveis globais, que existe enquanto o programa está executando, mas em um número fixo, ou em variáveis locais, que só existem enquanto a função correspondente está executando

- Não podemos fazer uma função que cria e retorna um vetor, por exemplo, pois se usarmos uma variável local para o vetor e retornarmos o endereço dela esse endereço não será mais válido no instante em que retornarmos da função

- Além disso, o tamanho dos nosso vetores está sendo sempre pré-determinado

- Podemos contornar essas limitações com o uso de *alocação dinâmica*

`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 aquele 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)

- Essas 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