ESTRUTURAS DE DADOS

Ponteiros e Referências

Roteiro

- Definição
- Declaração de Ponteiros
- Inicialização de Ponteiros
- Utilização de Memória
- Cuidados com Ponteiros

Definição de Ponteiros

- **Um ponteiro é uma variável cujo conteúdo é um endereço de memória, não um valor no sentido tradicional.**

- **Os endereços podem ser a localização na memória de uma variável ou função.**

- **Se a variável *x* tiver como valor o endereço da variável *y*, então dizemos:**
  - **"*x* aponta para *y*".**

- **O endereço de uma variável (ou função) é a localização na memória do primeiro byte ocupado por ela.**

- **Conhecer o endereço de uma variável permite criar estruturas complexas.**
  - **Listas Encadeadas são implementadas com um item conhecendo o endereço do item seguinte.**

- **A possibilidade de trabalhar diretamente com a memória permite criar programas mais eficientes.**

Declaração de Ponteiros

Para declarar um ponteiro, usamos a seguinte sintaxe:

```
tipo *ponteiro;
```

- **tipo: se refere para qual tipo de dados o ponteiro estará apontando.**
- **ponteiro: é o nome da variável.**

- **Por exemplo, a seguir declaramos um variável chamada intPointer que aponta para um valor do tipo inteiro.**

```
int* intPointer;
```

- **Como a variável acima não foi inicializada, o seu conteúdo será undefined.**

- **A pergunta agora é: "Como obter um endereço de memória?"**
  - Isso pode ser feito de maneira estática ou dinâmica.

Inicialização de Ponteiros

O operador & nos permite obter o endereço de memória de uma variável. Feito isso, podemos inicializar um ponteiro

```
// Declarando variáveis
int alpha;
int* intPointer;

// Inicializando ponteiro
intPointer = &alpha;
```

- **Uma segunda maneira de inicializar ponteiros é com alocação dinâmica, um mecanismo pelo qual um programa aloca e libera memória em tempo de execução.**

- **Vantagens**
  - **Elimina a necessidade de definir a priori o tamanho da memória a ser utilizada.**
  - **É possível aumentar ou diminuir o tamanho da memória utilizada em tempo de execução.**

- **Os operadores new e delete são utilizados para efetuar a alocação e desalocação de memória, respectivamente.**

- **Por exemplo, alocando memória dinamicamente para armazenar um inteiro.**

```
int *intPointer;
intPointer = new int;
```

- **Características da Alocação Dinâmica**
  - **As variáveis residem em um local diferente das que foram alocadas estaticamente.**
  - **Uma variável alocada de forma dinâmica com new não possui nome.**
  - **Essa variável precisa ser acessada indiretamente pelo ponteiro retornado por new.**

Utilização de Memória

- Temos um ponteiro e queremos acessar o valor que está na memória. Nesse caso, usamos o operador * como um prefixo para o nome da variável.

- O operador * é um operador unário que retorna o conteúdo da variável localizada no endereço especificado.

- **Para obter o conteúdo que está localizado no endereço apontado por intPointer:**

```
anotherInt = *intPointer;
```

- **Para alterar o conteúdo que está localizado no endereço apontado por intPointer:**

```
*intPointer = 25;
```

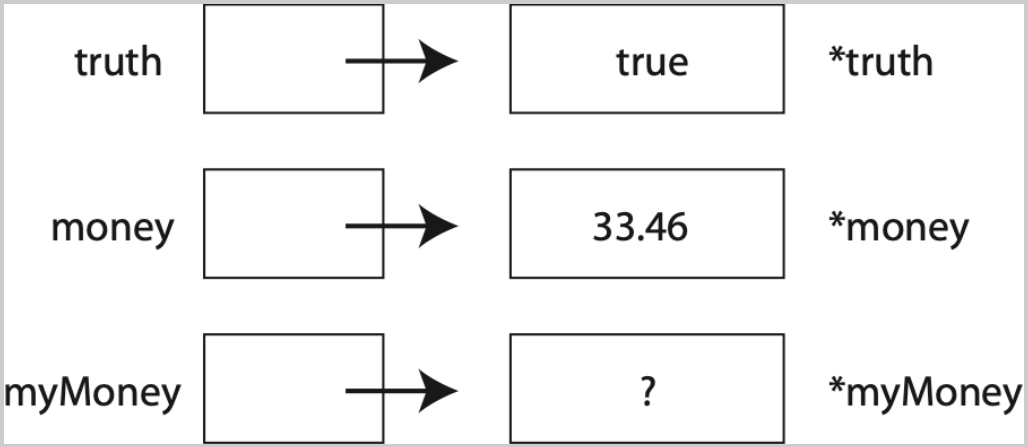
- **Um ponteiro com valor 0 (zero), por definição, aponta para o vazio, mas não queremos confundir com o inteiro zero.**

- **Nesse caso, usaremos a constante NULL que está no pacote csttdev.**

```
#include <cstddef>
bool* truth = NULL;
float* money = NULL;
```

Vamos observar a memória após algumas operações:

```
bool* truth = new bool;
*truth = true;
float* money = new float;
*money = 33.46;
float* myMoney = new float;
```



- **Qualquer operação que pode ser aplicada a uma variável do tipo int pode ser aplicada a *intPointer.**

- **Qualquer operação que pode ser aplicada a uma variável do tipo float pode ser aplicada a *money.**

- **Qualquer operação que pode ser aplicada a uma variável do tipo bool pode ser aplicada a *truth.**

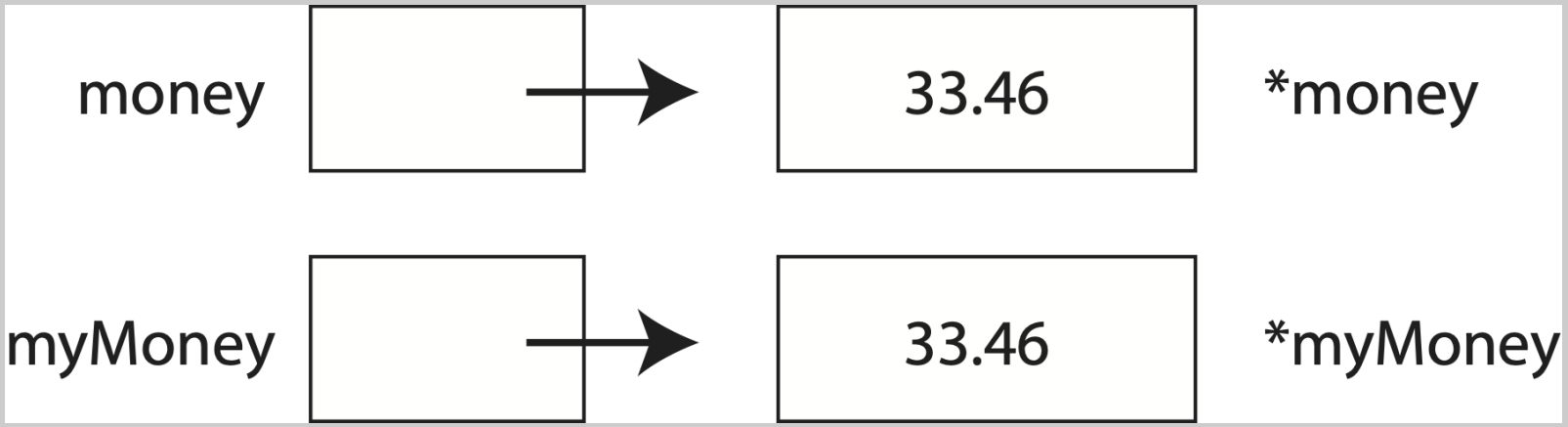
Cuidados com Ponteiros

- As duas operações a seguir são completamente diferentes.
  - Na primeira, o conteúdo de memória apontado por money é copiado para a região apontada por myMoney.
  - Na segunda, myMoney passa a apontar para a mesma região apontada por money.

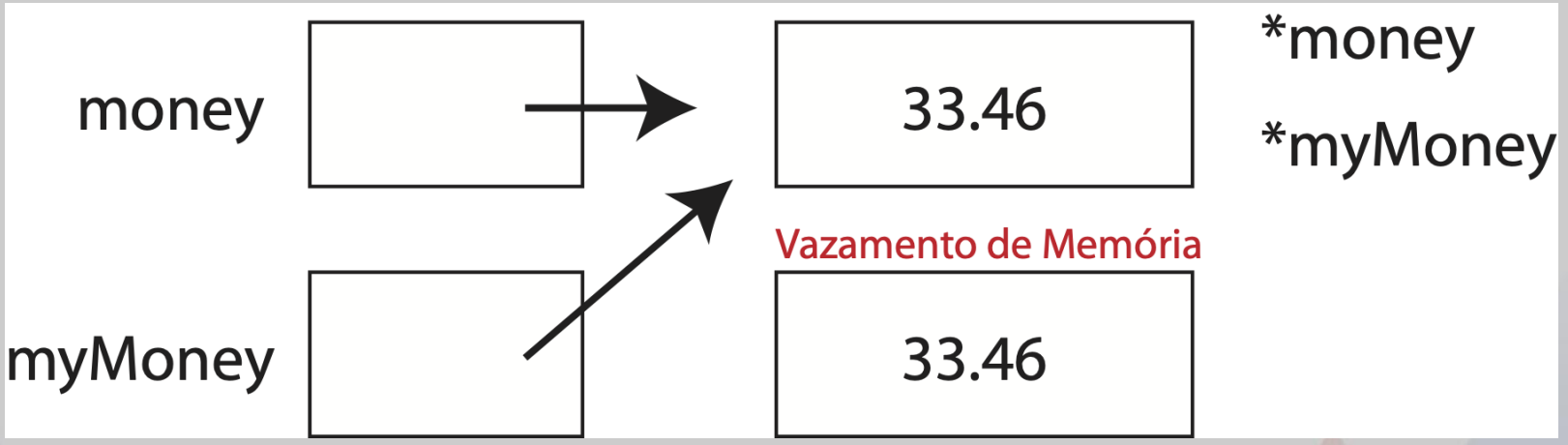
```
*myMoney = *money; // 1

myMoney = money; // 2
```

```
*myMoney = *money; // 1
```

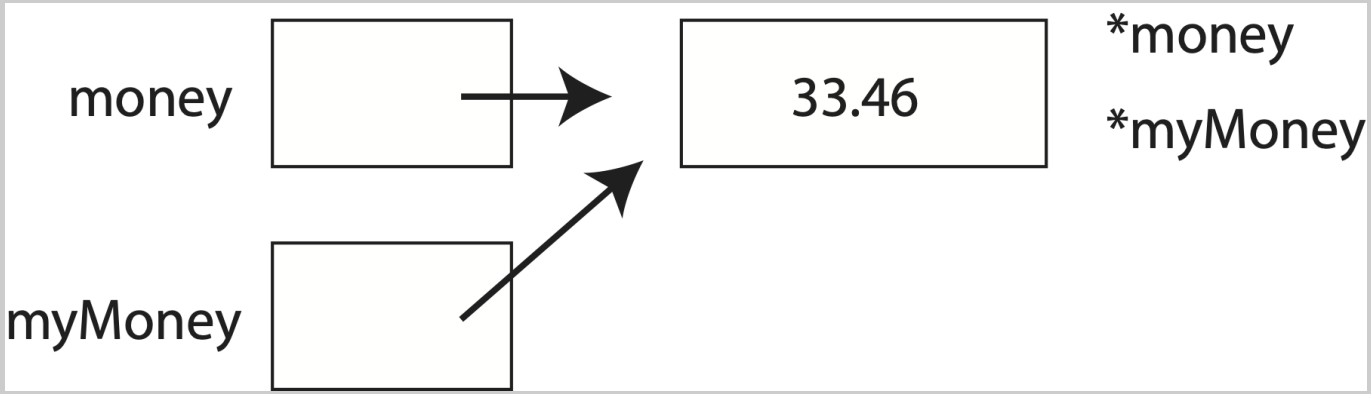



```
myMoney = money; // 2
```



- **Supondo que a segunda operação fosse a sua intenção, evite vazamento de memória com delete.**

```
delete myMoney;
myMoney = money;
```



- **Observe que delete não inutiliza a variável ponteiro, apenas libera a região que ela aponta.**

ESTRUTURAS DE DADOS

Ponteiros e Referências