Estruturas de Dados I

Introdução

Prof. Bruno Azevedo

Instituto Federal de São Paulo

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus Catanduva

Ponteiros para Ponteiros

- Já vimos como criar vetores unidimensionais dinamicamente, mas e quanto a vetores multidimensionais?
- Por exemplo, matrizes.
- Para este fim, usaremos ponteiros para ponteiros.
- Ponteiros para ponteiros são ponteiros que apontam para outros ponteiros.
- Ou seja, são ponteiros que armazenam endereços para ponteiros.
- Um ponteiro para ponteiro é declarado da seguinte forma: tipodedado **nomeDoPonteiroParaPonteiro.
- Por exemplo:

```
int **ponteiroParaPonteiro;
```

- Utilizamos o operador de desreferência (*) da mesma forma que com ponteiros; entretanto, quantos operadores usarmos determina o que obteremos.

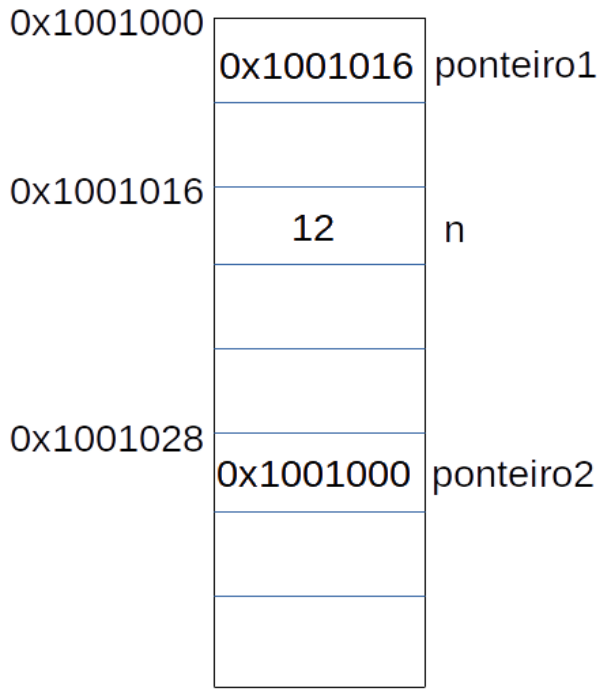
Ponteiros para Ponteiros

```
#include <iostream>
using namespace std;
int main() {
    int n = 12;
    int *pont1 = &n;
    int **pont2 = &pont1;
    cout << "Valor: " << **pont2 << endl;
    cout << "End. de n: " << *pont2 << endl;
    cout << "End. de pont1: " << pont2 << endl;
    return 0;
}
```

- No código acima, se utilizarmos dois operadores de desreferência, temos acesso ao valor de n.
- Se utilizarmos um operador de desreferência, temos acesso ao endereço da variável, que está armazenado em ponteiro1.
- Se não utilizarmos operadores de desreferência, temos acesso ao endereço do ponteiro1.

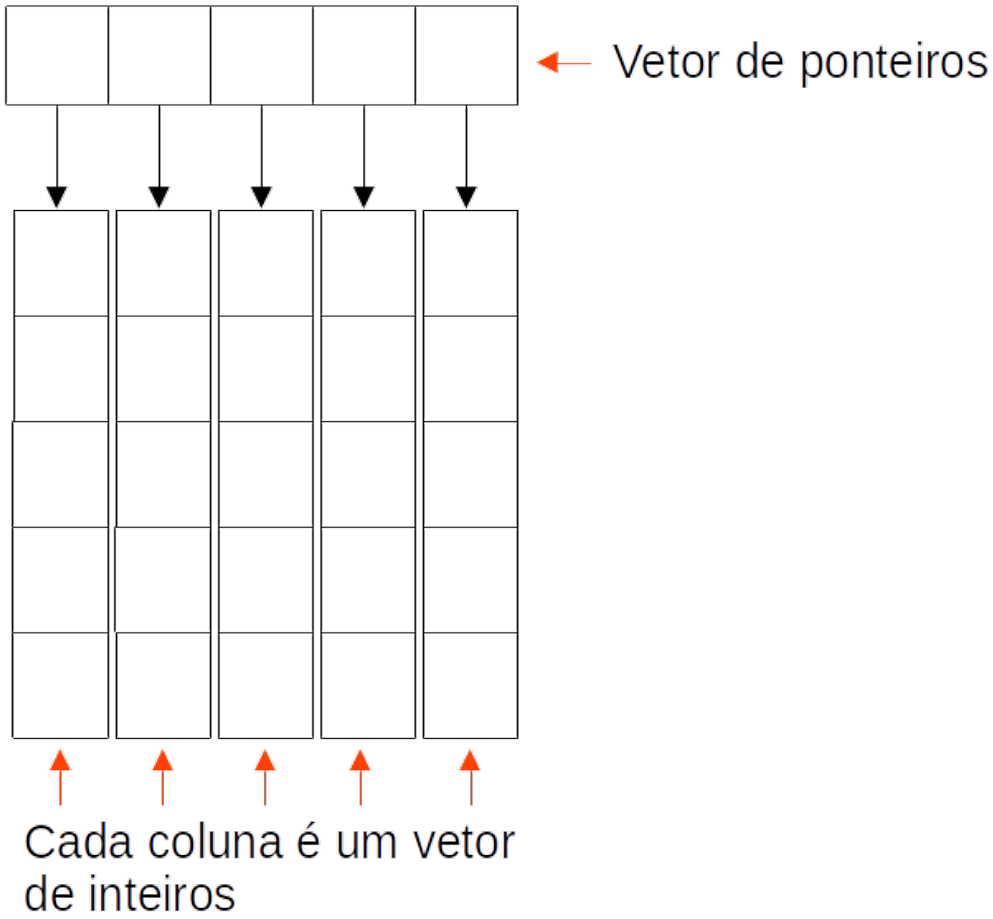
Ponteiros para Ponteiros

```
#include <iostream>
using namespace std;
int main() {
    int n = 12;
    int *pont1 = &n;
    int **pont2 = &pont1;
    cout << "Valor: " << **pont2 << endl;
    cout << "End. de n: " << *pont2 << endl;
    cout << "End. de pont1: " << pont2 <<
     endl;
    return 0;
}
```



Matrizes Alocadas Dinamicamente

- Usaremos ponteiros para ponteiros para alocar dinamicamente uma matriz.
- Alocaremos dinamicamente um vetor de ponteiros, e alocaremos um vetor de inteiros em cada posição deste vetor.



Matrizes Alocadas Dinamicamente

```
#include <iostream>
using namespace std;
int main() {
    int linhas, colunas;
    cout << "Digite o número de linhas: ";
    cin >> linhas;
    cout << "\nDigite o número de colunas: ";
    cin >> colunas;
    int **matriz = new int*[linhas]; // Alocando um vetor de ponteiros
    for (int i = 0; i < linhas; i++)
        matriz[i] = new int[colunas];  // Alocando um vetor de inteiros em cada posição do vetor
    cout << endl;
    // Preenchendo e imprimindo a matriz resultante
    for (int i = 0; i < linhas; i++) {
        for (int j = 0; j < colunas; j++) {
            matriz[i][j] = i * colunas + j;
            cout << matriz[i][j] << "\t";
        }
        cout << endl;
    }
    // Liberando a memória
    for (int i = 0; i < linhas; i++) {
        delete[] matriz[i];
    }
    delete[] matriz;
    return 0;
}
```

- O código acima aloca uma matriz dinamicamente.
- A operação delete[] libera a memória alocada pelo vetor. Usamos delete para apenas uma variável alocada e delete[] para vetores.

Referências

- Uma variável de referência (ou referência) é um "apelido".
- Quando criamos uma referência, e a inicializamos com o nome de outro objeto, a referência age como um nome alternativo para o objeto.
- Ou seja, qualquer coisa que você faça com a referência é feita no objeto original.
- Uma referência é declarada da seguinte forma: tipodedado &nomeReferência.
- Por exemplo:

```
int original = 12;
int &rNovaReferencia = original;
```

- Referências precisam ser inicializadas em sua criação.
- Variáveis de referência não podem ser reatribuídas.
- Geralmente, prefixamos a letra r no nome da variável de referência de modo a identificá-la com maior facilidade.

Referências

- Não confundam o operador AND Binário, o operador de endereço, e o operador de referência. Estes possuem o mesmo símbolo (&), mas são operadores distintos.
- Mas e se usarmos o operador de endereço com referências?
- Se você pedir o endereço de uma referência, ela retorna o endereço do seu alvo. Por exemplo:

```
#include <iostream>
using namespace std;
int main() {
    int intVar = 12;
    int &rRefVar = intVar;
    cout << "intVar: " << intVar << endl;
    cout << "rRefVar: " << rRefVar << endl;
    cout << "&intVar: " << &intVar << endl;
    cout << "&rRefVar: " << &rRefVar << endl;
    return 0;
}
```

- Este código exibirá o mesmo valor nas duas primeiras impressões e o mesmo endereço nas duas últimas impressões.

Referências

- Mencionei que não podemos reatribuir uma referência. Mas e se fizermos isso?

```
#include <iostream>
using namespace std;
int main() {
    int intVar = 12, intVar2 = 14;
    int &rRefVar = intVar;
    rRefVar = intVar2; // não é o que você pensa...
    cout << "intVar: " << intVar << endl;
    cout << "intVar2: " << intVar2 << endl;
    cout << "rRefVar: " << rRefVar << endl;
    cout << "&intVar: " << &intVar << endl;
    cout << "&intVar2: " << &intVar2 << endl;
    cout << "&rRefVar: " << &rRefVar << endl;
    return 0;
}
```

Referências

- Mencionei que não podemos reatribuir uma referência. Mas e se fizermos isso?

```
#include <iostream>
using namespace std;
int main() {
    int intVar = 12, intVar2 = 14;
    int &rRefVar = intVar;
    rRefVar = intVar2; // não é o que você pensa...
    cout << "intVar: " << intVar << endl; // intVar: 14
    cout << "intVar2: " << intVar2 << endl; // intVar2: 14
    cout << "rRefVar: " << rRefVar << endl; // rRefVar: 14
    cout << "&intVar: " << &intVar << endl; // &intVar: (endereço de intVar)
    cout << "&intVar2: " << &intVar2 << endl; // &intVar2: (endereço de intVar2
        )
    cout << "&rRefVar: " << &rRefVar << endl; //  &rRefVar: (endereço de
       intVar)
    return 0;
}
```

- Referências são apelidos para uma variável, são para efeitos práticos, a própria variável com outro nome.
- No código acima, o que parece ser uma reatribuição é na verdade a atribuição de um novo valor à variável.
- intVar recebeu o valor de intVar2.

Referências

- Podemos usar referências com ponteiros.

```
#include <iostream>
using namespace std;
int main() {
    int intVar = 12, intVar2 = 14;;
    int *intPtr = &intVar;
    int *&rRefVar = intPtr;
    rRefVar = &intVar2;
    cout << *rRefVar << endl;
    cout <<  rRefVar << endl;
    return 0;
}
```

- O que esse código imprime?

Referências

```
#include <iostream>
using namespace std;
int main() {
    int intVar = 12, intVar2 = 14;;
    int *intPtr = &intVar;
    int *&rRefVar = intPtr;
    rRefVar = &intVar2;
    cout << *rRefVar << endl; // 14
    cout <<  rRefVar << endl; // exibirá o endereço de intVar2
    return 0;
}
```

- rRefVar é uma referência para um ponteiro para inteiro.
- Como no exemplo anterior, temos **uma atribuição de valor** da variável original e não uma reatribuição da referência.
- Quando fazemos:

```
rRefVar = &intVar2;
```

- Estamos mudando a variável que o ponteiro original (intPtr) aponta. Estamos atribuindo um novo endereço para ele.
- Portanto, o programa exibe o valor e o endereço de intVar2.

Exercícios (13)

⇒ Ponteiros para Ponteiros

- Escreva um programa em C++ que declare uma variável inteira intVar, um ponteiro para inteiro ptr1, e um ponteiro para ponteiro para inteiro 'ptr2'. Atribua o endereço de intVar a ptr1 e o endereço de ptr1 a ptr2. Através de ptr2, atribua um novo valor a intVar e imprima o valor resultante.
- Escreva um programa em C++ que receba dois ponteiros para ponteiros para inteiros e realize a troca dos valores apontados por eles.
- Escreva um programa em C++ que solicita do usuário três valores inteiros e cria uma uma matriz dinâmica de inteiros de três dimensões usando alocação dinâmica de memória. Preencha a matriz com valores e imprima seus elementos.

⇒ Referências

- Escreva um programa em C++ que troque os valores de duas variáveis float usando referências.
- Escreva um programa em C++ que use referências para incrementar os valores de um vetor de inteiros.