Ponteiros e Referências

Linguagem de Programação I

Waldson Patrício

Conteúdo

- Introdução
- Operador `address-of`
- Operador de dereferência
- Ponteiros
- Referências
- Ponteiros e *Arrays*
- Ponteiros e Variáveis
- `this`
- Erros Comuns utilizando ponteiros

Introdução

- Quando declaramos uma variável, o computador associa o nome dela à uma posição de memória
- Quando utilizamos a variável pelo nome, o computador deve fazer duas coisas:
  1. Verificar qual o endereço de memória que o nome da variável está associado
  2. Acessar aquela posição de memória e retornar ou modificar seu valor
- Em C++, podemos fazer essas duas operações de forma independente utilizando os operadores de `address-of` e de dereferência

Operador `address-of`

- Retorna o endereço de memória de uma variável (&)
- Em C++, um endereço de memória é um ponteiro para aquela localização de memória

```
1    int a = 10;
2    int* b = &a;//b recebe o endereço de memória de a
```

Operador de Dereferência

- Acessa ou modifica um valor em uma posição de memória (*)

```
1  #include <iostream>
2
3  int main(int argc, char *argv[]) {
4    int a = 10;
5    int* b = &a;
6
7    std::cout << a << std::endl;
8
9    int c = *b;
10
11   *b = 20;
12
13   std::cout << a << std::endl;
14   std::cout << c << std::endl;
15 }
```

Ponteiros

- Ponteiros são variáveis que armazenam posições de memória (valores inteiros)
- Como visto anteriormente, ponteiros são declarados utilizando o caractere asterisco (*), assim como o operador de dereferência
- A diferença está que ele é utilizado na **declaração de uma variável**

Ponteiros

- Ponteiros geralmente apontam para:
  - Outra variável
  - Endereço de memória alocado dinamicamente
  - Nulo (`nullptr`)

```
1    int number = 10;
2    int* a = &number;
3    int *b = new int;
4    int* c = nullptr;
```

Referências

- Referências são outros nomes que damos para uma mesma posição de memória
- São os “apelidos” de uma variável
- Utilizam o mesmo caractere “e comercial” (&), assim como o operador `address-of`
- A diferença é que ele é utilizado na `declaração de uma variável`

Referências

- Referências não podem ser nulas

```
1    int a  = 10;
2    int &b = a;
3    int& c = a;
4
5    c = 20;
6
7    std::cout << a << std::endl; // 20
8    std::cout << b << std::endl; // 20
9    std::cout << c << std::endl; // 20
```

Referências

- Por padrão, o C++ passa os valores por cópia quando chamamos uma função
- Podemos passar parâmetros como referência

Referências

```
1   #include <iostream>
2   void por_copia(int a) {
3       a = 20;
4   }
5
6   void por_referencia(int& a) {
7       a = 30;
8   }
9
10  int main() {
11      int n = 10;
12      por_copia(n);
13      std::cout << n << std::endl;//10
14
15      por_referencia(n);
16      std::cout << n << std::endl;//30
17
18      return 0;
19  }
```

Ponteiros e Arrays

- Uma variável array é, na verdade, um ponteiro para o primeiro elemento do array
- Arrays são blocos de memória contíguas
- Quando acessamos `array[3]`, estamos acessando 4 posições de memória em relação à posição inicial do array

Ponteiros e Arrays

- Podemos utilizar aritmética com ponteiros para acessar seus elementos:

```
1  #include <iostream>
2
3  int main() {
4    int a[] = {10, 20, 30, 40};
5    int *ptr = a;
6    std::cout << a[3] << std::endl;
7    std::cout << *(ptr + 3) << std::endl;
8    return 0;
9  }
```

Ponteiros e Variáveis

- Se o ponteiro aponta para um objeto (`struct` ou classe), devemos acessar os valores através do operador de dereferência de structs (->)

```
1    MyStruct s;
2    MyStruct* p = &s;
3    std::cout << p->variable << std::endl;
4    std::cout << s.variable << std::endl;
```

`this`

- Quando criamos um método em uma `struct` ou classe, dentro dos métodos há sempre uma variável implícita chamada `this`
- `this` é um ponteiro do próprio tipo que é utilizado para referenciar outros valores do próprio objeto

this

Quando fazemos isso:

```
1    MyStruct::method(int a, int b)
2    {
3        //implementação
4    }
```

internamente o compilador faz isso:

```
1    MyStruct::method(MyStruct* this, int a, int b)
2    {
3        //implementação
4        //podemos utilizar this aqui. Ex:
5        //Ex: this->variable,
6        //    this->otherMethod()
7    }
```

Erros Comuns ao Utilizar Ponteiros

- Utilização de ponteiros não inicializados
- Apontamento para locais inválidos de memória
- Utilização de ponteiros depois de desalocação de memória

Erros Comuns ao Utilizar Ponteiros

Utilização de Ponteiros não inicializados

```
1       MyStruct* s;
2       MyStruct* s2 = nullptr;
3
4       //trecho de código grande pra nos distrair
5
6       s->variable; //s não foi inicializada
7       s2->variable; //s2 aponta para nulo
```

Erros Comuns ao Utilizar Ponteiros

Apontamento para locais inválidos de memória

```cpp
#include <iostream>

int* funcao() {
    int a = 10;
    return &a;
    //a é desalocado quando a função termina
}

int main() {
    int* ptr = funcao();

    std::cout << *ptr << std::endl;

    return 0;
}
```

Erros Comuns ao Utilizar Ponteiros

Utilização de ponteiros depois de desalocação de memória

```
1   #include <iostream>
2
3   int main() {
4       MyStruct* s = new MyStruct();
5       //usa s
6       delete s;
7
8       s->variable; //a memória que s apontava já foi desalocada
9
10      return 0;
11  }
```

Dúvidas

?

INSTITUTO METRÓPOLE DIGITAL