Programação Orientada a Objetos em C++

Ponteiros em C++

Agostinho Brito

2020

Agenda

Classes e Ponteiros

Classes e Ponteiros

- O uso de ponteiros em C e C++ é idêntico, de sorte que conhecimentos adquiridos em C acerca do assunto são perfeitamente aplicáveis em C++.

```
1  class Vetor2d{
2  ...
3  };
4
5  Vetor2d *v1;
6  float a;
7  v1->setX(3);
8  a = v2->getX();
```

- O acesso às propriedades e funções-membro da classe é feito pelo operador `->`, assim como se dá nas structs da linguagem C.
- A linguagem provê um ponteiro especial `this` que aponta para O PRÓPRIO OBJETO.

Alocação dinâmica de memória

Alocação dinâmica de memória

- Embora a sintaxe de acesso a ponteiros seja semelhante a C, os processos de alocação e liberação de memória são bem diferentes.
- Enquanto que em C a alocação e liberação de memória é feita pelas funções `malloc()` e `free()`, em C++ são feitas por operadores.

	alocação individual	alocação bloco
C	`malloc()/free()`	`malloc()/free()`
C++	`new/delete`	`new[]/delete[]`

- Além disso, os operadores também invocam dois importantes elementos da linguagem: o **construtor** e o **destrutor**.

Alocação dinâmica com tipos primitivos

Alocação de tipos primitivos

- O operador `new` recebe o tipo a ser alocado.
- O operador `delete` recebe o ponteiro com o endereço do bloco previamente alocado por `new`.

```
int *x;
x = new int;
*x = 3;
delete x;
```

- O operador `new[]` recebe o tipo e a quantidade de elementos a serem alocados.
- O operador `delete[]` recebe o ponteiro com o endereço do bloco previamente alocado por `new[]`.

```
int *x, n=10;
x = new int[n];
x[2] = 3;
delete [] x;
```

Alocação de tipos primitivos

- O operador `new` recebe o tipo a ser alocado.
- O operador `delete` recebe o ponteiro com o endereço do bloco previamente alocado por `new`.

```
int *x;
x = new int;
*x = 3;
delete x;
```

- O operador `new[]` recebe o tipo e a quantidade de elementos a serem alocados.
- O operador `delete[]` recebe o ponteiro com o endereço do bloco previamente alocado por `new[]`.

```
int *x, n=10;
x = new int[n];
x[2] = 3;
delete [] x;
```

Alocação dinâmica com tipos primitivos

!

NUNCA libere blocos de memória alocados com `new` usando `delete[]`.

!

NUNCA libere blocos de memória alocados com `new[]` usando `delete`.

!

O comportamento é indefinido!

Praticando alocação dinâmica em C++...

Obrigado

Alocação dinâmica com Classes

Alocação dinâmica com Classes

- Os mesmos cuidados com a combinação `new`/`delete` e `new[]`/`delete[]` também vale para alocação dinâmica de memória com classes.
- A sintaxe da alocação, entretanto, é diferente.

```
1  class Vetor2d{
2    ...
3  public:
4    Vetor2d();
5    Vetor2d(float a_, float _b);
6  };
7  Vetor2d *v1, *v2, *v3;
8  v1 = new Vetor2d;
9  v2 = new Vetor2d(3,4);
10 v3 = new Vetor2d[3];
11 delete v1;
12 delete v2;
13 delete [] v3;
```

Praticando alocação dinâmica em C++...

Obrigado