Linguagem de Programação C++

O que acontecerá se um programa *driver* para a classe Matriz efetuasse a seguinte sequência de instruções:

```
#include "Matriz.h"
...
int main()
{
  char opcao;
  ...
  switch (opcao)
  {
    case 1:
    {
      int l, c;
      cout << endl << "Encontre com o numero de linhas da matriz: ";
      cin >> l;
      cout << endl << "Encontre com o numero de colunas da matriz: ";
      cin >> c;
      Matriz m(l, c);
      ...
    }
    ...
  }
  ...
}
```

Destrutor

Para resolvermos este problema, existe, em cada classe, uma função membro especial, denominada Destrutor, que permitem providenciar a desalocação controlada dos objetos instanciados.

Estas funções membros são nomeadas na forma ~<nome_da_classe> (nome da classe prefixado com til).

Exemplo:

```
class Matriz
{
   ...
   public:
      ~Matriz();
}
```

Linguagem de Programação C++

Sendo assim, uma função-membro destrutor, poderá tomar as providências necessárias para a desalocação adequada do objeto receptor.

Por exemplo: desalocar nodo a nodo uma lista encadeada ou um vetor alocado dinamicamente.

É muito relevante frisar que um destrutor **não recebe parâmetro** e **nem retorna um valor**. Não sendo possível a especificação de nenhum tipo de retorno, nem mesmo o void. Cada classe possui **apenas um destrutor**.

Uma classe sempre possui um destrutor, se o programador não fornecer um destrutor explicitamente, o compilador cria um destrutor "vazio", que desempenha um papel importante em objetos criados por herança e composição.

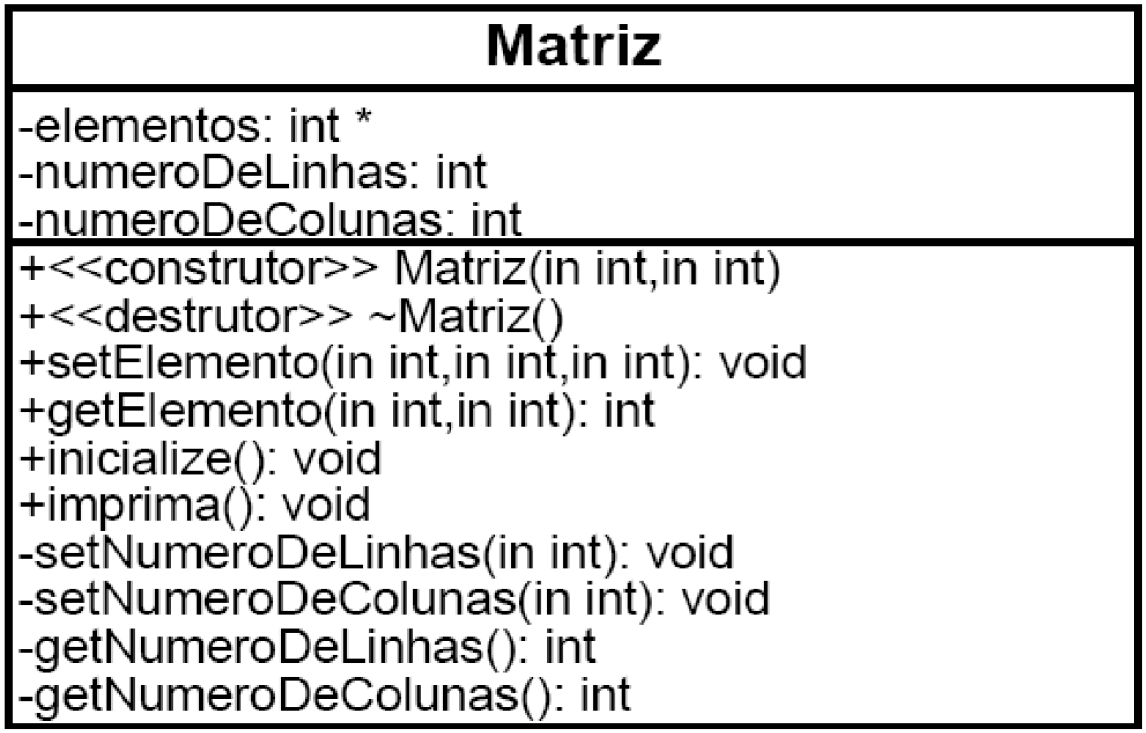
Observação: Construtoras e destrutoras não são herdadas!

Exercício:

Agora, com base no conceito anterior adapte sua solução para o exercício do slide 193 sobre a classe Matriz.

Exercício:

Com base no que vimos a respeito da representação do construtor de uma classe em um diagrama de classes em UML siga sua lógica e construa um diagrama de classe em UML que represente a classe Matriz.



Lembre-se que neste caso o diagrama de classes apresentado está direcionado para a linguagem C++ o que foi mencionado não ser aconselhável.

Empacotador de pré-processador

Creio que alguns de vocês já devem ter se perguntado sobre a possibilidade de inserir, por acidente, a definição de uma classe mais de uma vez em um programa, pois em um programa grande ocorrem muitas inclusões de arquivos cabeçalhos que por sua vez podem incluir outros arquivos cabeçalhos.

Para evitar a ocorrência deste erro devemos utilizar o empacotador de pré-processador **#ifndef**. O qual representa 'se não definido'.

Sua sintaxe é:

```
#ifndef <ROTULO>
#define <ROTULO>
   ...
#endif
```

Para uma compreensão adequada analisaremos sua utilização na classe Matriz.

```
//conteúdo do arquivo Matriz.h
#ifndef MATRIZ_H
#define MATRIZ_H
  class Matriz
  {
    private:
      int *elementos;
      int numeroDeLinhas;
      int numeroDeColunas;
      void setNumeroDeLinhas(int);
      void setNumeroDeColunas(int);
      int getNumeroDeLinhas();
      int getNumeroDeColunas();
    public:
      Matriz(int, int);
      ~ Matriz();
      void setElemento(int, int, int);
      int getElemento(int, int);
      void inicialize();
      void imprima();
  };
#endif
```

Linguagem de Programação C++

Exercício:

Visando melhor fixar a utilização da alocação dinâmica de memória, explorando as funções-membros construtor e destrutor, explorar o princípio do ocultamento de implementação e demonstrar as vantagens na manutenibilidade de sistemas gerados com a utilização da OO, adapte a solução do exercício do slide 193 sobre a classe Matriz. Faça com que este armazene os elementos da matriz através de um membro de dados int ** ao invés de int * ou vise e versa para quem gerou inicialmente sua solução com int **. Utilize o empacotador de pré-processador.

UNIVASF

Formas de utilização de uma classe

Para exemplificar as formas de utilização de uma classe, nos utilizaremos da classe Ponto2D que definimos anteriormente.

Uma vez que a classe foi definida, ela pode ser utilizada como um tipo em declarações de objeto, array, ponteiro e referência, como mostrado a seguir:

```
Ponto2D ponto;
Ponto2D vetorDePontos[10];
Ponto2D *ponteiroParaPonto = &ponto;
Ponto2D &referenciaAPonto = ponto;
...
ponto.mostraCoordenadas();
vetorDePontos[0].mostraCoordenadas();
(*ponteiroParaPonto).mostraCoordenadas();
ponteiroParaPonto->mostraCoordenadas();
referenciaAPonto.mostraCoordenadas();
```

Escopo de variáveis

Variáveis declaradas em uma função-membro têm escopo de bloco e são conhecidas apenas por esta função.

Uma função membro pode definir uma variável com o mesmo nome de uma variável com escopo de classe, a variável de escopo de classe é ocultada pela variável de escopo de bloco no escopo do bloco. Esta variável oculta pode ser acessada colocando-se o nome da classe seguido pelo operador de resolução de escopo binário antes do nome da variável.

Um detalhe importante é que variáveis globais ocultas podem ser acessadas utilizando-se o operador de resolução de escopo unário.

Vamos analisar um exemplo.

Linguagem de Programação C++

```
int var;
class Exemplo
{
   private:
      int var;
   public:
      void teste()
      {
         int var;
         var = 3;
         Exemplo::var = 2;
         ::var = 1;
      }
};
...
```

Argumentos-padrão

A linguagem C++ possibilita a atribuição de um valor padrão para parâmetros de funções membros.

Uma grande utilidade para este recurso é a possibilidade de definirmos um construtor padrão com parâmetros. Pois, não existirá mais a obrigatoriedade da definição de argumentos na instanciação de objetos, já que se nenhum valor for fornecido na chamada de construtor, o mesmo ainda inicializará os membros de dados mantendo o objeto instanciado em um estado consistente.

Exemplo:

```
...
class Ponto2D
{
  ...
  public:
    Ponto2D (float = 0, float = 0);
    ...
};
```

Linguagem de Programação C++

Podemos então instanciar objetos da classe Ponto2D da seguinte forma:

```
...
Ponto2D ponto1; //x e y receberão 0
Ponto2D ponto2(4); //x receberá 4 e y receberá 0
Ponto2D ponto3(7,3); //x receberá 7 e y receberá 3
...
```

Exercício:

Crie uma classe Jogo que permitirá escrever um programa completo para jogar o jogo-da-velha. A classe contém como membros de dados privados um vetor bidimensional 3 x 3 de inteiros. O construtor deve inicializar a grade vazia com todos os valores como zero. Permita dois jogadores humanos. Para onde quer que o primeiro jogador se mova, coloque 1 no quadrado especificado. Coloque 2 para onde quer que o segundo jogador se mova. Todo movimento deve ocorrer em um quadrado vazio. Depois de cada movimento, determine se houve uma derrota ou um empate. Se você se sentir motivado, modifique seu programa de modo que o computador faça o movimento para um dos jogadores. Além disso, permita que o jogador humano especifique se quer ser o primeiro ou o segundo a jogar.