CMP1056 - Técnicas de Programação II
Max Gontijo de Oliveira
1º Trabalho - Número complexo e matriz

1 Resumo

Implementar uma classe que represente um número imaginário e uma que represente uma matriz.

2 Número complexo

Deverá ser implementada uma classe que seja capaz de representar um número complexo (parte real e parte imaginária). Vamos considerar nessa documentação que essa classe irá se chamar `Complex`. Essa classe deverá possibilitar sua instanciação das seguintes maneiras:

- sem informar nenhum parâmetros (nesse caso a parte real e a parte imaginárias serão inicializadas com zero);
- informando apenas um parâmetro `float` (nesse caso a parte real será dada por esse parâmetro e a parte imaginária será inicializada com zero);
- informando dois parâmetros `float` (nesse caso, primeiro parâmetro será o valor da parte real e o segundo a parte imaginária)

A classe `Complex` deverá prover os seguintes métodos públicos:

- `add` - método para realizar soma e que deve ter duas sobrecargas.
  - recebe um valor `float`; cria um novo objeto `Complex`; a parte real desse novo objeto será a soma da parte real do próprio objeto atual e o parâmetro passado; a parte imaginária desse novo objeto será apenas a cópia da parte imaginária do próprio objeto atual.
  - recebe uma referência para um objeto `Complex` (`Complex &parametro`); a parte real desse novo objeto será a soma da parte real do próprio objeto atual e a parte real do objeto passado por parâmetro; a parte imaginária desse novo objeto será a soma da parte imaginária do próprio objeto atual e a parte imaginária do objeto passado por parâmetro.
- `sub` - método para realizar subtração e que deve ter duas sobrecargas. Ambas as sobrecargas deverão seguir a mesma linha dos métodos de adição, evidentemente substituindo tal operação pela subtração.
- `mul` - método para realizar multiplicação e que deve ter duas sobrecargas. Ambas as sobrecargas deverão seguir a mesma linha dos métodos de adição, evidentemente substituindo tal operação pela multiplicação. **Dica:** caso não se recorde, pesquise como se faz a multiplicação de números complexos.
- `print` - método para imprimir o número complexo. Essa impressão deverá seguir a forma $a + bi$ onde a se refere à parte real e b à parte imaginária do número complexo.

Evidentemente, nos métodos `add`, `sub` e `mul`, o novo objeto `Complex` criado deverá ser retornado.
Com essa implementação, será possível realizar operações como:

```
Complex a;
Complex b(5);
Complex c(3,2);
a = b.add(5);
a = a.sub(c);
a = a.mul(b);
a = a.add(c.mul(b.add(7)));
a.print();
```

Com a intenção de deixar o código mais limpo, deverão ser implementadas sobrecargas dos operadores de adição (+), de subtração (−) e de multiplicação (∗), onde suas implementações deverão ser idênticas aos métodos `add`, `sub` e `mul` respectivamente.

Com a implementação dessas sobrecargas de operadores, agora deverá ser possível realizar operações como:

```
Complex a;
Complex b(5);
Complex c(3,2);
a = b + 5;
a = a - c;
a = a * b;
a = a + c * (b + 7);
a.print();
```

3 Matriz

Deverá ser implementada uma classe que seja capaz de representar uma matriz. Vamos considerar nessa documentação que essa classe irá se chamar `Matrix`.

Essa classe deverá possibilitar sua instanciação através da informação de dois parâmetros: o número de linhas e o número de colunas. Quando instanciada, a classe deverá criar uma matriz dinâmica de valores inteiros com as dimensões informadas e inicializar todos os elementos dessa matriz dinâmica com 0 (zero). Além do construtor, a classe deverá ter um destrutor que garanta que a matriz dinamicamente instanciada seja desalocada da memória.

A classe `Matrix` deverá prover os seguintes métodos públicos:

- `add` - método para realizar a soma entre duas matrizes. Esse método irá receber a referência de um outro objeto `Matrix` (`Matrix` &parametro) que tenha as mesmas dimensões do objeto `Matrix` atual e deverá criar um novo objeto `Matrix` com as mesmas dimensões. Os valores dos elementos dessa nova matriz serão dados pela soma matricial da matriz do próprio objeto com a matriz do objeto passado por parâmetro.
- `sub` - método para realizar a subtração entre duas matrizes. A implementação deverá seguir a mesma linha do método de adição, evidentemente substituindo tal operação pela subtração.
- `mul` - método para realizar a multiplicação entre duas matrizes. A implementação deverá seguir a mesma linha do método de adição, evidentemente substituindo tal operação pela subtração. Lembre-se que para realizar a operação de multiplicação entre duas matrizes, a quantidade de colunas da primeira matriz deve ser igual à quantidade de linhas da segunda e a matriz resultando tem dimensão formada pela quantidade de linhas da primeira matriz e quantidade de colunas da segunda matriz.
- `print` - método para imprimir a matriz. Essa impressão deverá apresentar os valores de cada célula da matriz de modo que cada linha da matriz esteja em uma linha na saída. Se for fazer com saída no terminal, procure utilizar tabulações ou a formatação da função `printf` para garantir um mínimo de organização da matriz apresentada.

Evidentemente, nos métodos `add`, `sub` e `mul`, o novo objeto `Matrix` criado deverá ser retornado.

Com essa implementação, será possível realizar operações como:

```
1 Matrix a(4,5);
2 Matrix b(4,5);
3 Matrix c(5,2);
4 Matrix x = a.add(b);
5 Matrix y = a.sub(b);
6 Matrix z = (x.add(y).sub(a.add(b))).mul(c);
7 z.print();
```

Com a intenção de deixar o código mais limpo, deverão ser implementadas sobrecargas dos operadores de adição ($+$), de subtração ($-$) e de multiplicação ($*$), onde suas implementações deverão ser idênticas aos métodos `add`, `sub` e `mul` respectivamente.

Com a implementação dessas sobrecargas de operadores, agora deverá ser possível realizar operações como:

```
1 Matrix a(4,5);
2 Matrix b(4,5);
3 Matrix c(5,2);
4 Matrix x = a + b;
5 Matrix y = a - b;
6 Matrix z = ((x + y) - (a + b)) * c;
7 z.print();
```

3.1 Destrutor

Devido ao fato de que a matriz interna à classe é dinamicamente alocada, é evidente que seria necessário criar um destrutor que garanta que essa matriz não permaneça ocupando espaço na memória após o objeto `Matrix` ser desalocado. Contudo, ao implementar um destrutor que faça isso, toda vez que um objeto `Matrix` for passado por valor como parâmetro para alguma função ou método ou quando for retornado por uma função ou método, o objeto `Matrix` (cópia) será destruído ao fim do escopo onde foi criado, chamando o destrutor que, por sua vez irá, de forma equivocada, remover a amtriz dinamicamente instanciada.

Assim, após implementar a classe e realizar os testes, procure descobrir uma maneira de criar um destrutor de modo que esse problema seja resolvido.

4 Testes

Deverá ser criada uma aplicação para testar as classes.