Programowanie 2

Zadanie 7

Piotr Błaszyński

15 maja 2019

Do klasy opisującej macierz (L6.pdf) dodać nowe funkcjonalności.

- jeżeli ktoś tego jeszcze nie zrobił, to zamienić używanie double ** na vector, przykład uzycia vector na końcu,
- dodawanie macierzy, zwracające nowy obiekt, (nowa metoda z nazwą, nie operator),
- odejmowanie macierzy, zwracające nowy obiekt, (nowa metoda, nie operator),
- mnożenie macierzy, zwracające nowy obiekt, (nowa metoda, nie operator),
- w powyższych metodach sprawdzać poprawność wymiarów macierzy, jeżeli są niepoprawne do wykonania działania, rzucamy wyjątek (przykład w dalszej cześci zadania),
- zmodyfikować metodę dodającą liczbę do macierzy, tak aby zwracała macierz (nowy obiekt),

Proszę nie robić żadnego menu i pobierania informacji od użytkownika, również nazwy pliku, wartości elementów macierzy przechowywać w kodzie źródłowym i tam je ewentualnie modyfikować. Pliki mogą mieć stałą nazwę (podawaną oczywiście przy wywołaniu konstruktora, a nie w jego środku).

Dodatkowo wydzielić oddzielny plik na część nagłówkową klasy i jej część implementacyjną. **Zwracam uwagę**, żeby w pliku nagłówkowym nie używać konstrukcji using namespace std (i oczywiście innych using namespace). Powoduje to zaburzenie prawidłowego działania mechanizmu przestrzeni nazw (zainteresowanych odsyłam do szerszej dyskusji na stackoverflow.

Przykładowy kod demonstracyjny (macierze powinny zawierać liczby rzeczywiste, ale wartości powinny być łatwe do weryfikacji):

- wczytać macierz 4x3 z pliku (dalej opisywaną jako A),
- wczytać macierz 3x4 z pliku (dalej opisywaną jako B),
- pomnożyć macierze A i B i zapisać wynik w macierzy C,
- wyświetlić macierz A wierszami, każdy wiersz poprzedzając kilkoma spacjami
- wyświetlić macierz B i C wierszami, obok siebie (wiersz B - wiersz C) (te dwa punkty mają zobrazować schemat mnożenia),
- zapisać macierz C do pliku
- pomnożyć macierze w nieprawidłowy sposób, powinien zostać rzucony wyjątek, który należy przechwycić w funkcji main (lub innej funkcji programu, która wywoła to mnożenie) i w tej funkcji wypisac stosowny komunikat,
- wcześniej zapisaną macierz C odczytać do macierzy D,
- do wszystkich elementów macierzy D dodać wartość,
- dodać macierze C i D, wynik przechować w macierzy E,
- dodać macierze C i D, wynik przechować w macierzy F,
- odjąć macierze E i F, wynik przechować w macierzy G,
- wyświetlić macierz C, D i E wierszami, obok siebie (wiersz C - wiersz D - wiersz E),
- wyświetlić macierze F i G, pod sobą,
- spróbować dodać macierze A i B, powinno to też spowodować wygenerowanie wyjątku, umieścić w głównej funkcji jego obsługę.

Przykładowy kod prezentujący użycie wyjątków (zalecam używanie std::exception lub pochodnych):

```
int RecomputePositiveNumbers(int value)
{
  if (value<=0)
    throw std::exception("This␣should␣never␣happen␣:(.␣
       Param␣value␣should␣be␣positive");
  RestOfComputations();
}
void Computations()
{
  try
  {
    RecomputePositiveNumbers(-1);
  }
  catch (int e)
  {
    cout << "Exception␣Nr.␣" << e << endl;
  }
  catch (std::exception& e)
  {
    cout << "Ex.␣desc:␣" << e.what() << endl;
  }
}
```

Przykładowe operacje na dwóch obiektach tej samej klasy (proszę zwrócić uwagę na niedawno wprowadzony do C++ ciekawy sposób deklarowania zmiennej, której typ jest automatycznie określony - słowo kluczowe **auto**):

```
#include "stdafx.h"
#include <iostream>
#include <sstream>
#include <string>

using namespace std;
class Complex
{
private:
  double real, im;
public:
  Complex add(Complex&);
  Complex (double real, double im): real(real), im(im)
     {;}
  string toString();
};
Complex Complex::add(Complex& param)
{
  Complex result(real + param.real, im + param.im);
  return result;
}

string Complex::toString()
{
  stringstream result;
  result << real << " + " << im << "i";
  return result.str();
}

int main(int argc, char* argv[])
{
  Complex x (1.2, 3.4);
  Complex y (1.5, 7.6);
  auto z = x.add(y);
  cout << z.toString() << endl;
  return 0;
}
```

Przykładowa deklaracja obiektu przechowującego vector i podstawowe operacje na nim, zwracam uwagę na dodatkową spację przed drugim nawiasem trójkątnym (w kompilatorach obsługujących standard C++11 nie jest już wymagana):

```
#include <vector>
#include <iostream>

using namespace std;

int main(int argc, char* argv[])
{
        vector <vector<double> > vOut;
        vector<double> vIn;
        vIn.push_back(1);
        vIn.push_back(2);
        vIn[0]=3;
        vOut.push_back(vIn);
        cout << vOut[0][0] << endl;
}
```