Programowanie 2

Zadanie 6

Piotr Błaszyński

30 kwietnia 2019

Przygotować klasę do obsługi macierzy liczb **rzeczywistych**. Klasa ma mieć metody pozwalające na:

- utworzenie macierzy o podanych (dowolnych) rozmiarach, pamięc przydzielana dynamicznie, wartości w komórkach domyślnie 0, rozmiar macierzy podawany jako parametry konstruktora,
- zmianę wartości dowolnej komórki (podajemy położenie modyfikowanej komórki i jej nową wartość),
- zwrócenie reprezentacji w postaci stringa dla:
  - podanej komórki,
  - podanego wiersza,
  - podanej kolumny,
  - całej macierzy (korzystając z metody zwracającej wiersze).
- dodanie jednej liczby do wszystkich elementów macierzy,
- zamianę na macierz transponowaną (NIE TYLKO dla macierzy kwadratowych),

Klasa ma mieć także funkcje do zapisu i odczytu (konstruktor) danych do i z pliku tekstowego o podanej nazwie.

Przykładowy kod demonstracyjny:

- z przygotowanego ręcznie pliku wczytać macierz o rozmiarach 4x3,
- zmodyfikować w niej kilka wartości,

- zapisać do innego pliku (można wykorzystać czas jako część nazwy pliku, jak w zad. 4),
- utworzyć nową macierz na podstawie zapisanego przed chwilą pliku,
- transponować macierz,
- zapisać do kolejnego pliku,
- utworzyć kolejną nową macierz na podstawie zapisanego przed chwilą pliku,
- dodać jakąś liczbę do wszystkich elementów macierzy,
- zapisać do kolejnego pliku.

Dla przypomnienia przykładowe użycie klas obsługujących pliki:

```
#include <ctime>
#include <iostream>
#include <istream>
#include <fstream>
#include <string>

using namespace std;

void file_read ()
{
  ifstream inputFile("foobar.txt");
  if (!inputFile.is_open())
    throw std::exception("Can't␣read␣foobar.txt");
  string lineAsString;
  while (!inputFile.eof())
  {
    inputFile >> lineAsString;
    cout << lineAsString << endl;
  }
}

void file_write ()
{
  ofstream outputFile("fooboo.txt");
  if (!outputFile.is_open())
    throw std::exception("Can't␣write␣fooboo.txt");
  outputFile <<"Line␣1" <<endl;
  string lineAsString="LineAsString";
  outputFile << lineAsString <<endl;
  char *lineAsCharPointer="LineAsChar*";
  outputFile <<lineAsCharPointer <<endl;
}
```

Przykładowa alokacja pamięci na dane macierzy, wyjątkowo (i po raz ostatni) można użyć double ** (preferowane użycie klasy vector):

```
class Matrix
{
.
.
        std::vector<std::vector<double> > data;
.
.
};
void Matrix::allocate_data ()
{
        //obie linie niepotrzebne
  data.reserve(width);//tylko alokacja,
  data.resize(width);//alokacja i inicjalizacja
}
```

```
class Matrix
{
.
.
        double **data;
.
.
};

void Matrix::allocate_data ()
{
  data = new double *[width];
  for (int i = 0 ; i<width ; i++)
  {
        data[i] = new double [height];
  }
}

void Matrix::free_data ()
{
  for (int i = 0 ; i<width ; i++)
  {
        delete [] data[i];
  }
  delete [] data;
}
```

Dla szybkich i łagodnych: przygotować 2 wersje klasy: korzystającą z klasy vector i korzystającą z klasycznych wskaźników. Porównać czas wykonywania operacji transponowania dla obu rodzajów macierzy. Do porównania potrzebne będzie wykonanie transponowania wielokrotnie dla odpowiednio dużej macierzy.