Programowanie 2

Zadanie 8

Piotr Błaszyński

26 maja 2019

Dodać do macierzy (zadania 6 i 7) operatory przeciążone i zademonstrować ich działanie. Przygotować wersję macierzy jako szablon (*template*). Wszystkie występujące w programie obiekty macierzy powinny byc tworzone jako macierze z elementami typu float. Przykład implementacji klas szablonowych podany poniżej.

Dodać następujące operatory:

- przypisania (operator =),
- dodawania (operator +),
- dodawania z przypisaniem (operator +=),
- mnożenia (operator *),
- mnożenia z przypisaniem (operator *=) (zwrócic uwagę na rzucane wyjątki),
- transponowania (wybrać operator (znaczek operatora) i zastanowić się, czy przeciążanie przy pomocy operatora takich operacji to poprawny sposób postępowania),

Ponadto proszę zmienić kopiowanie danych tak, aby korzystało z std::copy. Przykładowy kod demonstracyjny:

- wczytać z plików 3 macierze A, B, C o identycznych rozmiarach,
- D = A + B + C;,
- wyświetlić A, B, C, D wierszami (obok siebie)
- wczytać z plików macierze E i F, które w wyniku mnożenia dadzą macierz kwadratową,

- G = E *F;
- G += G * G;
- wyświetlić E, F, G,
- G= operator Transponowania G;
- wyświetlić G.

Jeżeli ktoś zrealizuje powyższe punkty, proszę dołożyc funkcje obliczania wyznacznika dla macierzy (np. tylko 2x2, dla innych można rzucać wyjątek).

Ważne zasady:

- jeżeli znaczenie operatora nie jest całkowicie jasne, nie powinien być przeciążany,
- zawsze należy sie trzymać ustalonego znaczenia operatora,
- jeżeli już przeciążamy to należy dostarczyć pełny zbiór operacji powiązanych (jeżeli działa + to powinien działać też +=)

Przykładowy kod prezentujący przeciążania operatora =, + i porównania (kontynuacja klasy z poprzednich zadań):

```
#include <algorithm>
using namespace std;

class Complex
{
public:
  friend void swap(Complex& first, Complex& second)
  {
    using std::swap;
    swap(first.real, second.real);
    swap(first.imaginary, second.imaginary);
    //swap(first.fieldArray, second.fieldArray);
  }
  Complex& operator=(Complex source)
  {
    swap(*this, source);
    return *this;
  }
  Complex operator+(Complex ) const;
  friend bool operator==(const Complex& , const Complex&
      );
  friend bool operator<(const Complex& , const Complex&
     );
};

Complex Complex::operator+(Complex right) const
{
  Complex result;
  result.real = real + right.real;
  result.imaginary = imaginary + right.imaginary;
  return result;
}
```

Przykładowy kod prezentujący przeciążania operatora =, + i porównania, ciąg dalszy:

```
inline bool operator==(const Complex& left, const
   Complex& right)
{
        return left.real==right.real && left.imaginary
           == right.imaginary;
}

inline bool operator!=(const Complex& left, const
   Complex& right)
{return !operator==(left,right);}

inline bool operator< (const Complex& left, const
   Complex& right)
{return left.real<right.real;}//TODO: imaginary

inline bool operator> (const Complex& left, const
   Complex& right)
{return  operator< (right,left);}

inline bool operator<=(const Complex& left, const
   Complex& right)
{return !operator> (left,right);}

inline bool operator>=(const Complex& left, const
   Complex& right)
{return !operator< (left,right);}
```

Przykładowy kod prezentujący deklarację i użycie klasy szablonowej (fragmenty klasy z poprzednich przykładów):

```
#include <algorithm>
using namespace std;

template <class T>
class Complex
{
public:
  friend void swap(Complex& first, Complex& second)
     noexcept
  {
    using std::swap;
    swap(first.real, second.real);
    swap(first.imaginary, second.imaginary);
  }
  Complex ():real(0),imaginary(0) { ; }
  Complex& operator=(Complex source)
  {
    swap(*this, source);
    return *this;
  }
  Complex(T real, T imaginary):
        real(real), imaginary(imaginary) { ; }
  Complex operator+(Complex) const;
private:
  T real;
  T imaginary;
};

template <class T>
Complex<T> Complex<T>::operator+(Complex<T> right) const
{
  Complex result;
  result.real = real + right.real;
  result.imaginary = imaginary + right.imaginary;
  return result;
}
int main(void)
{
  Complex<float> value = Complex<float>(1.1, 2.2);
  return 0;
}
```