ФГАОУ ВО Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа киберфизических систем и управления

**ОТЧЕТ**

**по лабораторной работе №9**

**по теме «Boost»**

по дисциплине «Практикум по программированию»

Выполнил

студент гр.23533/2 А.Д. Шурак

Проверил

ассистент В.Э. Ковалевский

Санкт-Петербург

2019

Задание

На основе лабораторной работы прошлого семестра продемонстрировать работу библиотеки Boost.

**Разработка программы**

Для начала необходимо подключить библиотеку Boost к проекту, чтобы использовать её методы и классы.

Необходимо:

1. Скачать нужные файлы.
2. В созданном проекте, в обозревателе решений выбрать «Свойства», «Свойства конфигурации».
3. В разделе «С/С++» выбрать «Общие», «Дополнительные каталоги включаемых файлов», подключить нужную папку, прописать её адрес в виде: D:\boost\_1\_69\_0; %(AdditionalIncludeDirectories)
4. Выбрать графу «Предварительно откомпилированный заголовок» и в ней указать «Не использовать предварительно скомпилированные заголовки».
5. Далее можно приступать к реализации программы.

Реализуем транспонирование матрицы, заполненной рандомными комплексными числами.

Для работы с матрицами и генерации рандомных значений необходимо подключить:

#include <boost/numeric/ublas/matrix.hpp>//подключаем библиотеку для работы с матрицами

#include <boost/numeric/ublas/io.hpp>

#include <boost/random.hpp>//библиотека для генерации рандомных значений

#include <iostream>

#include "complex.h"

В основной функции программы пропишем заполнение матрицы рандомными числами.   
  
matrix<Complex> m(3, 3);//матрица комплексных

for (int i=0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

//заполнение рандомными

m(i, j).SetRe(gen(rand));

m(i, j).SetIm(gen(rand));

}

}

А также создадим функцию для транспонирования:

matrix<Complex> c = trans(m);//транспонирование

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

std::cout << c(i, j) << "\t";

std::cout << std::endl;

}

Важно выбрать генератор чисел и распределение.

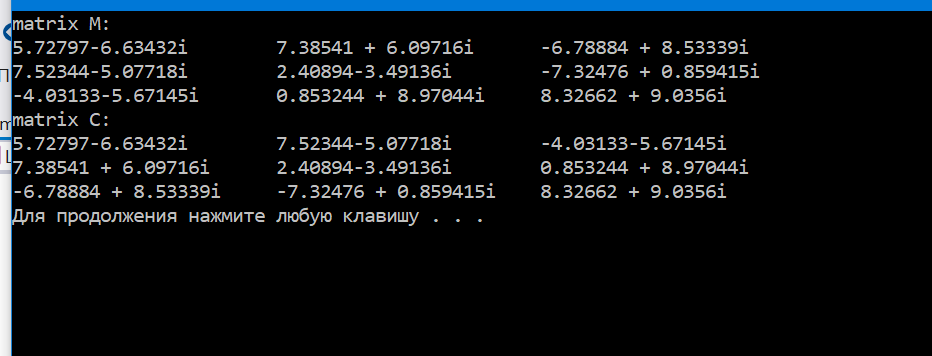
using namespace boost::numeric::ublas;

boost::random::mt19937 rand;//выбор генератора случайных чисел

boost::random::uniform\_real\_distribution<double> gen(-9.1, 9.1);//выбор распределения

После этого, соединив данные нововведения с предыдущим проектом, можно запускать программу.

**Демонстрация работы программы**



**Заключение**

В ходе выполнения лабораторной работы на основе лабораторной работы прошлого семестра была продемонстрирована работа библиотеки Boost.

**Приложение**

#include <boost/numeric/ublas/matrix.hpp>//подключаем библиотеку для работы с матрицами

#include <boost/numeric/ublas/io.hpp>

#include <boost/random.hpp>//библиотека для генерации рандомных значений

#include <iostream>

#include "complex.h"

int main()

{

using namespace boost::numeric::ublas;

boost::random::mt19937 rand;//выбор генератора случайных чисел

boost::random::uniform\_real\_distribution<double> gen(-9.1, 9.1);//выбор распределения

matrix<Complex> m(3, 3);//матрица комплексных

for (int i=0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

//заполнение рандомными

m(i, j).SetRe(gen(rand));

m(i, j).SetIm(gen(rand));

}

}

std::cout << "matrix M:" << std::endl;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

std::cout << m(i, j) << "\t";

std::cout << std::endl;

}

std::cout << "matrix C:" << std::endl;

matrix<Complex> c = trans(m);//транспонирование

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

std::cout << c(i, j) << "\t";

std::cout << std::endl;

}

system("pause");

}

#pragma once

#ifndef ComplexClasses\_H

#define ComplexClasses\_H

#include <iostream>

class Complex

{

float re;

float im;

public:

Complex();

Complex(float x, float y);

~Complex();

Complex operator=(float B);

Complex operator+(Complex B);

Complex operator\*(Complex B);

Complex operator-(Complex B);

Complex operator/(Complex B);

Complex operator+(float B);

Complex operator/(float B);

Complex operator\*(float B);

Complex operator^(int degree);

friend std::ostream & operator<<(std::ostream & os, const Complex & c);

void SetRe(float x) { re = x; };

void SetIm(float x) { im = x; };

void sopr();

};

#endif

#include "complex.h"

Complex::Complex()

{

re = 0.0;

im = 0.0;

}

Complex::~Complex()

{

}

Complex::Complex(float x, float y)

{

re = x;

im = y;

}

Complex Complex::operator+(Complex B)

{

return Complex(re + B.re, im + B.im);

}

Complex Complex::operator+(float B)

{

return Complex(re + B, im);

}

Complex Complex::operator\*(Complex B)

{

Complex X;

X.re = (re\*B.re) - (im\*B.im);

X.im = (re\*B.im) + (im\*B.re);

return X;

}

Complex Complex::operator/(Complex y)

{

Complex x;

x.re = ((re\*y.re) + (im\*y.im)) / ((y.re\*y.re) + (y.im\*y.im));

x.im = ((im\*y.re) - (re\*y.im)) / ((y.re\*y.re) + (y.im\*y.im));

return x;

}

Complex Complex::operator/(float y)

{

Complex x;

x.re = (re\*y) / (y\*y);

x.im = (im\*y) / (y\*y);

return x;

}

Complex Complex::operator-(Complex y)

{

Complex x;

x.re = re - y.re;

x.im = im - y.im;

return x;

}

Complex Complex::operator\*(float B)

{

Complex X;

X.re \*= B;

X.im \*= B;

return X;

}

Complex Complex::operator=(float B)

{

return Complex(re = B, 0.0);

}

Complex Complex::operator^(int degree)

{

Complex X;

X.re = re;

X.im = im;

for (int i = 1; i < degree; i++)

{

X = X \* (\*this);

}

return X;

}

std::ostream& operator<<(std::ostream& os, const Complex &c)

{

os << c.re;

if (c.im > 0) os << " + ";

if (c.im != 0) os << c.im << "i";

return os;

}

void Complex::sopr()

{

if (im > 0)

{

im = im \* (-1);

}

else

{

im = im \* (-1);

}

}