Министерство образования Республики Беларусь

Учреждение образования

Брестский государственный технический университет

Кафедра ИИТ

Лабораторная работа №5

За 4 семестр

По дисциплине «ООТПиСП»

Тема: «Стандартная библиотека шаблонов»

Выполнил: студент 2 курса

Группы ПО-4(2)

Коташевич С.Н.

Проверил: Миндер А.В.

Брест 2021

Лабораторная работа №5

Стандартная библиотека шаблонов

**Цель:** освоить технологию обобщённого программирования с использованием библиотеки стандартных шаблонов (STL) языка C++.

**Основное содержание работы.**

Написать три программы с использованием STL. Первая и вторая программы должны демонстрировать работу с контейнерами STL, третья − использование алгоритмов STL.



**Вариант 13**

**Программа 1:**

**В программе № 1 выполнить следующее:**

1. Создать объект-контейнер в соответствии с вариантом задания и заполнить его данными, тип которых определяется вариантом задания.

2. Просмотреть контейнер.

3. Изменить контейнер, удалив из него одни элементы и заменив другие.

4. Просмотреть контейнер, используя для доступа к его элементам итераторы.

5. Создать второй контейнер этого же класса и заполнить его данными того же типа, что и первый контейнер.

6. Изменить первый контейнер, удалив из него n элементов после заданного и добавив затем в него все элементы из второго контейнера.

7. Просмотреть первый и второй контейнеры.

**Текст программы:**

#include <iostream>

#include <deque>

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

/\*1. Создать объект-контейнер в соответствии с вариантом задания

и заполнить его данными, тип которых определяется вариантом задания.\*/

deque<long> d;

d.push\_back(21);

d.push\_front(43534);

d.push\_back(3468);

d.push\_front(4351234);

d.push\_back(68);

/\*2. Просмотреть контейнер.\*/

cout << "Контейнер 1: ";

for (int i = 0; i < d.size(); i++) {

cout << d[i] << " ";

}

cout << endl;

/\*3. Изменить контейнер, удалив из него одни элементы и заменив другие.\*/

d.pop\_back();

for (int i = 0; i < d.size(); i++) {

d[i] = d[i] + 1;

}

/\*4. Просмотреть контейнер, используя для доступа к его элементам итераторы.\*/

deque<long>::iterator p = d.begin();

cout << "Контейнер 1: ";

while (p != d.end()) {

cout << \*p++ << " ";

}

cout << endl;

/\*5. Создать второй контейнер этого же класса

и заполнить его данными того же типа, что и первый контейнер.\*/

deque<long> e(d);

//6. Изменить первый контейнер, удалив из него n элементов после заданного

// и добавив затем в него все элементы из второго контейнера.

int n = 2;

int delAfter = 2;

p = d.begin();

p = p + delAfter;

d.erase(p, p + n);

for (int i = 0; i < e.size(); i++) {

d.push\_back(e[i]);

}

/\*7. Просмотреть первый и второй контейнеры.\*/

cout << "Контейнер 1: ";

for (int i = 0; i < d.size(); i++) {

cout << d[i] << " ";

}

cout << endl;

cout << "Контейнер 2: ";

for (int i = 0; i < e.size(); i++) {

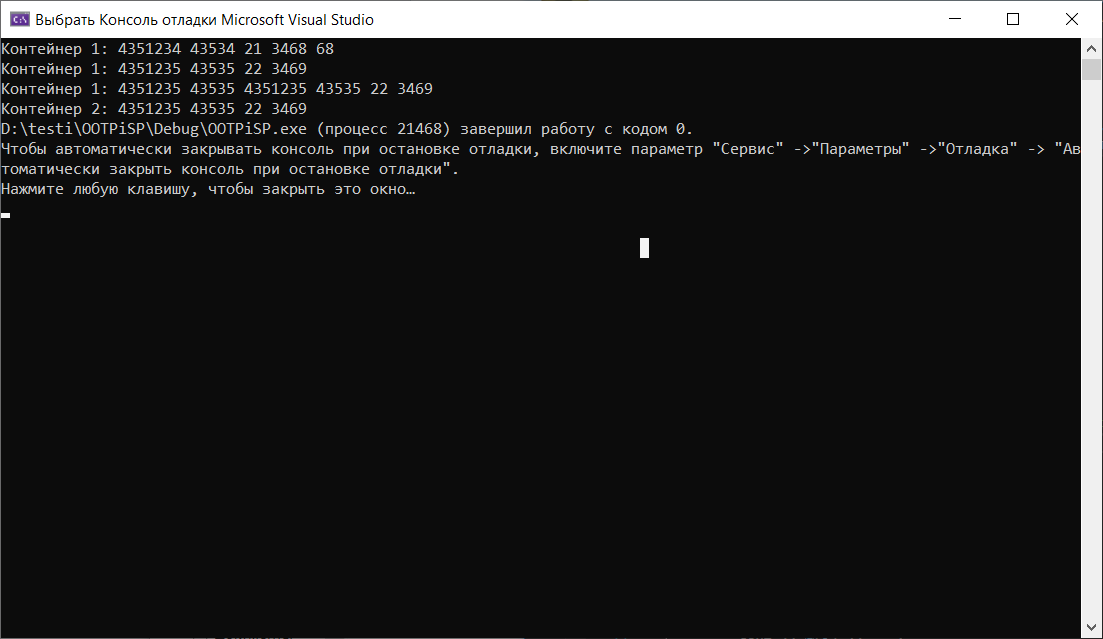
cout << e[i] << " ";

}

return 0;

}

**Результат работы:**



**Программа 2:**

В программе №2 выполнить то же самое, но для данных пользовательского типа.

**Текст программы:**

#include <iostream>

#include <deque>

#include <string>

using namespace std;

class Complex {

int real;

int imag;

friend ostream& operator<< (ostream& outstream, const Complex& cpl);

public:

Complex() {

this->real = 0;

this->imag = 0;

}

Complex(int re, int im) {

this->real = re;

this->imag = im;

}

void set\_complex(int re, int im) {

this->real = re;

this->imag = im;

}

string string\_complex() {

string str;

str += to\_string(this->real);

if (this->imag >= 0) {

str += '+';

str += to\_string(this->imag);

str += 'i';

}

else {

str += to\_string(this->imag);

str += 'i';

}

return str;

}

bool operator==(const Complex& cpl) {

return (this->real == cpl.real && this->imag == cpl.imag);

}

bool operator!=(const Complex& cpl) {

return !(\*this == cpl);

}

Complex& operator+(int i) {

this->imag += 1;

this->real += 1;

return \*this;

}

};

ostream& operator<< (ostream& outstream, Complex& cpl) {

return outstream << cpl.string\_complex();

}

int main()

{

setlocale(LC\_ALL, "rus");

Complex cm[5];

cm[0].set\_complex(2, 6);

cm[1].set\_complex(7, -6);

cm[2].set\_complex(5, 2);

cm[3].set\_complex(7, 5);

cm[4].set\_complex(12, -5);

/\*1. Создать объект-контейнер в соответствии с вариантом задания

и заполнить его данными, тип которых определяется вариантом задания.\*/

deque<Complex> d;

d.push\_back(cm[0]);

d.push\_front(cm[1]);

d.push\_back(cm[2]);

d.push\_front(cm[3]);

d.push\_back(cm[4]);

/\*2. Просмотреть контейнер.\*/

cout << "Контейнер 1: ";

for (int i = 0; i < d.size(); i++) {

cout << d[i] << " ";

}

cout << endl;

/\*3. Изменить контейнер, удалив из него одни элементы и заменив другие.\*/

d.pop\_back();

for (int i = 0; i < d.size(); i++) {

d[i] = d[i] + 1;

}

/\*4. Просмотреть контейнер, используя для доступа к его элементам итераторы.\*/

deque<Complex>::iterator p = d.begin();

cout << "Контейнер 1: ";

while (p != d.end()) {

cout << \*p++ << " ";

}

cout << endl;

/\*5. Создать второй контейнер этого же класса

и заполнить его данными того же типа, что и первый контейнер.\*/

deque<Complex> e(d);

//6. Изменить первый контейнер, удалив из него n элементов после заданного

// и добавив затем в него все элементы из второго контейнера.

int n = 2;

int delAfter = 2;

p = d.begin();

p = p + delAfter;

d.erase(p, p + n);

for (int i = 0; i < e.size(); i++) {

d.push\_back(e[i]);

}

/\*7. Просмотреть первый и второй контейнеры.\*/

cout << "Контейнер 1: ";

for (int i = 0; i < d.size(); i++) {

cout << d[i] << " ";

}

cout << endl;

cout << "Контейнер 2: ";

for (int i = 0; i < e.size(); i++) {

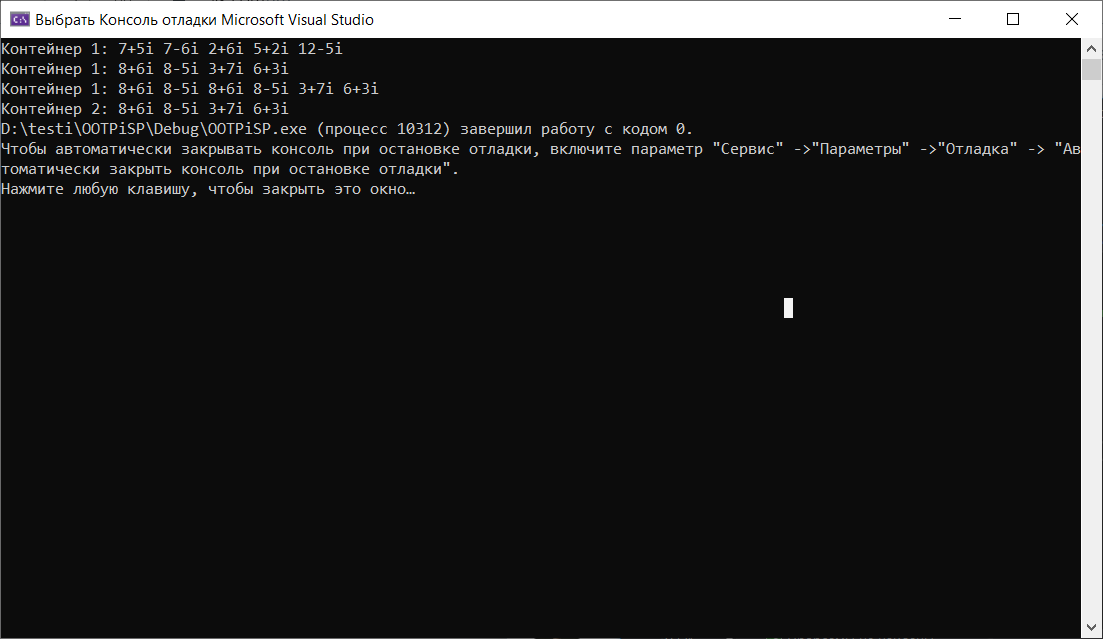
cout << e[i] << " ";

}

return 0;

}

**Результат работы:**



**Программа 3:**

В программе № 3 выполнить следующее:

1. Создать контейнер, содержащий объекты пользовательского типа. Тип контейнера выбирается в соответствии с вариантом задания.

2. Отсортировать его по убыванию элементов.

3. Просмотреть контейнер.

4. Используя подходящий алгоритм, найти в контейнере элемент, удовлетворяющий заданному условию.

5. Переместить элементы, удовлетворяющие заданному условию в другой (предварительно пустой) контейнер. Тип второго контейнера определяется вариантом задания.

6. Просмотреть второй контейнер.

7. Отсортировать первый и второй контейнеры по возрастанию элементов.

8. Просмотреть их.

9. Получить третий контейнер путем слияния первых двух.

10. Просмотреть третий контейнер.

11 .Подсчитать, сколько элементов, удовлетворяющих заданному условию, содержит третий контейнер.

12.Определить, есть ли в третьем контейнере элемент, удовлетворяющий заданному условию.

**Текст программы:**

#include <iostream>

#include <deque>

#include <algorithm>

#include <string>

using namespace std;

class Complex {

int real;

int imag;

friend ostream& operator<< (ostream& outstream, const Complex& cpl);

public:

Complex() {

this->real = 0;

this->imag = 0;

}

Complex(int re, int im) {

this->real = re;

this->imag = im;

}

int get\_real() {

return this->real;

}

int get\_imag() {

return this->imag;

}

void set\_complex(int re, int im) {

this->real = re;

this->imag = im;

}

string string\_complex() {

string str;

str += to\_string(this->real);

if (this->imag >= 0) {

str += '+';

str += to\_string(this->imag);

str += 'i';

}

else {

str += to\_string(this->imag);

str += 'i';

}

return str;

}

bool operator==(const Complex& cpl) {

return (this->real == cpl.real && this->imag == cpl.imag);

}

bool operator!=(const Complex& cpl) {

return !(\*this == cpl);

}

Complex& operator+(int i) {

this->imag += 1;

this->real += 1;

return \*this;

}

};

ostream& operator<< (ostream& outstream, Complex& cpl) {

return outstream << cpl.string\_complex();

}

int main()

{

setlocale(LC\_ALL, "rus");

Complex cm[5];

cm[0].set\_complex(2, 6);

cm[1].set\_complex(7, -6);

cm[2].set\_complex(5, 2);

cm[3].set\_complex(7, 5);

cm[4].set\_complex(17, -5);

/\*1. Создать контейнер, содержащий объекты пользовательского типа.

Тип контейнера выбирается в соответствии с вариантом задания.\*/

deque<Complex> d;

d.push\_back(cm[0]);

d.push\_front(cm[1]);

d.push\_back(cm[2]);

d.push\_front(cm[3]);

d.push\_back(cm[4]);

cout << "Контейнер 1: ";

for (int i = 0; i < d.size(); i++) {

cout << d[i] << " ";

}

cout << endl;

/\*2. Отсортировать его по убыванию элементов.\*/

sort(d.begin(), d.end(),

[](Complex a, Complex b) {

double modA = sqrt(pow(a.get\_real(), 2) + pow(a.get\_imag(), 2));

double modB = sqrt(pow(b.get\_real(), 2) + pow(b.get\_imag(), 2));

return modA > modB;

});

/\*3. Просмотреть контейнер.\*/

cout << "Контейнер 1, отсортированный по убыванию: ";

for (int i = 0; i < d.size(); i++) {

cout << d[i] << " ";

}

cout << endl;

/\*4. Используя подходящий алгоритм, найти в контейнере элемент,

удовлетворяющий заданному условию.\*/

deque<Complex>::iterator p = find\_if(d.begin(), d.end(),

[](Complex a) {

return a.get\_imag() == 2;

});

cout << "Найденный элемент: " << p->string\_complex();

cout << endl;

/\*5. Переместить элементы, удовлетворяющие заданному условию в другой(предварительно пустой) контейнер.

Тип второго контейнера определяется вариантом задания.\*/

deque<Complex> c(3);

remove\_copy\_if(d.begin(), d.end(), c.begin(),

[](Complex a) {

return a.get\_real() == 7;

});

/\*6. Просмотреть второй контейнер.\*/

cout << "Контейнер 2: ";

for (int i = 0; i < c.size(); i++) {

cout << c[i] << " ";

}

cout << endl;

/\*7. Отсортировать первый и второй контейнеры по возрастанию элементов.\*/

sort(d.begin(), d.end(),

[](Complex a, Complex b) {

double modA = sqrt(pow(a.get\_real(), 2) + pow(a.get\_imag(), 2));

double modB = sqrt(pow(b.get\_real(), 2) + pow(b.get\_imag(), 2));

return modA < modB;

});

sort(c.begin(), c.end(),

[](Complex a, Complex b) {

double modA = sqrt(pow(a.get\_real(), 2) + pow(a.get\_imag(), 2));

double modB = sqrt(pow(b.get\_real(), 2) + pow(b.get\_imag(), 2));

return modA < modB;

});

deque<Complex> e; //третий контейнер

/\*8. Просмотреть их.\*/

/\*9. Получить третий контейнер путем слияния первых двух.\*/

cout << "Контейнер 1, отсортированный по возрастанию: ";

for (int i = 0; i < d.size(); i++) {

cout << d[i] << " ";

e.push\_back(d[i]);

}

cout << endl;

cout << "Контейнер 2, отсортированный по возрастанию: ";

for (int i = 0; i < c.size(); i++) {

cout << c[i] << " ";

e.push\_back(c[i]);

}

cout << endl;

/\*10. Просмотреть третий контейнер.\*/

cout << "Контейнер 3: ";

for (int i = 0; i < e.size(); i++) {

cout << e[i] << " ";

}

cout << endl;

/\*11.Подсчитать, сколько элементов, удовлетворяющих заданному условию, содержит третий контейнер.\*/

int count = count\_if(d.begin(), d.end(),

[](Complex a) {

return a.get\_real() == 7;

});

cout << "Элементов, удовлетворяющих условию \"действительная часть = 7\" - " << count << endl;

/\*12.Определить, есть ли в третьем контейнере элемент, удовлетворяющий заданному условию.\*/

count = count\_if(d.begin(), d.end(),

[](Complex a) {

return a.get\_imag() == 2;

});

if (count == 0) {

cout << "Элемента, удовлетворяющего условию \"мнимая часть = 2\" нет" << endl;

}

else {

cout << "Элемент(ы), удовлетворяющий(ие) условию \"мнимая часть = 2\" есть" << endl;

}

return 0;

}

**Результат работы:**

