Пермский национальный исследовательский политехнический университет

**Лабораторная работа №11**

**Дисциплина: Основы алгоритмизации и**

**программирования**

**Вариант № 9**

**"Последовательные контейнеры библиотеки STL"**

Выполнил: Зайченко Никита Геннадьевич

Проверила: Доцент кафедры ИТАС Полякова О. А.

Пермь 2022

**Содержание отчета**

1) Постановка задачи (общая и конкретного варианта).

2) Функции для решения задачи 1.

3) Основная программа для решения задачи 1

4) Объяснение результатов работы программы.

5) Описание пользовательского класса для решения задачи 2.

6) Определение перегруженных операций для пользовательского класса.

7) Функции для решения задачи 2.

8) Основная программа для решения задачи 2

9) Объяснение результатов работы программы.

10) Описание параметризированного класса для решения задачи 3.

11) Определение методов и операций для решения задачи 3.

12) Основная программа для решения задачи 3

13) Объяснение результатов работы программы.

14) Функции для решения задачи 4.

15) Основная программа для решения задачи 4

16) Объяснение результатов работы программы.

17) Описание параметризированного класса для решения задачи 5.

18) Определение методов и операций для решения задачи 5.

19) Основная программа для решения задачи 2

20) Объяснение результатов работы программы.

21) Ответы на контрольные вопросы

**Постановка задачи (общая и конкретного варианта)**

1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.

2) Использование последовательных контейнеров библиотеки STL в ОО

программе.

3) **Задача 1**

1. Контейнер - двунапрвленная очередь

2. Тип элементов - int

**Задача 2**

Тип элементов Money (см. лабораторную работу №3).

**Задача 3**

Параметризированный класс – Вектор (см. лабораторную работу №7)

**Задача 4**

Адаптер контейнера - стек.

**Задача 5**

Параметризированный класс – Вектор

Адаптер контейнера - стек.

|  |  |  |
| --- | --- | --- |
| **Задание 3**  Найти максимальный  элемент и добавить его в конец контейнера | **Задание 4**  Найти элемент с  заданным ключом и  удалить его из  контейнера | **Задание 5**  К каждому элементу добавить  Среднее арифметическое  элементов контейнера |

**Функции для решения задачи 1.**

// Функция для создания контейнера и заполнения его элементами

deque<int> makeDeque(int n) {

deque<int> myDeque(n);

for (int i = 0; i < n; ++i) {

myDeque[i] = rand() % 100;

}

return myDeque;

}

// Функция для вывода элементов контейнера на экран

void printDeque(const deque<int>& myDeque) {

for (auto element : myDeque) {

cout << element << " ";

}

cout << endl;

}

**Основная программа для решения задачи 1**

int main() {

deque<int> myDeque = makeDeque(5); // создание контейнера

printDeque(myDeque);

// Задание 3

// Найти максимальный элемент и добавить его в конец контейнера

if (!myDeque.empty()) {

auto maxElementIndex = distance(myDeque.begin(), max\_element(myDeque.begin(), myDeque.end()));

myDeque.push\_back(myDeque[maxElementIndex]);

cout << "Max element = " << myDeque[maxElementIndex] << endl;

}

else {

cout << "Deque is empty" << endl;

}

printDeque(myDeque);

// Задание 4

// Найти элемент с заданным ключом и удалить его из контейнера

int key;

cout << "Key = ";

cin >> key;

auto it = find(myDeque.begin(), myDeque.end(), key);

if (it != myDeque.end()) {

myDeque.erase(it);

}

else {

cout << "Key not found" << endl;

}

// Задание 5

// К каждому элементу добавить среднее арифметическое элементов контейнера

double srednee = accumulate(myDeque.begin(), myDeque.end(), 0.0) / myDeque.size();

cout << "Average = " << srednee << endl;

for (auto& element : myDeque) {

element += static\_cast<int>(srednee);

}

printDeque(myDeque);

return 0;

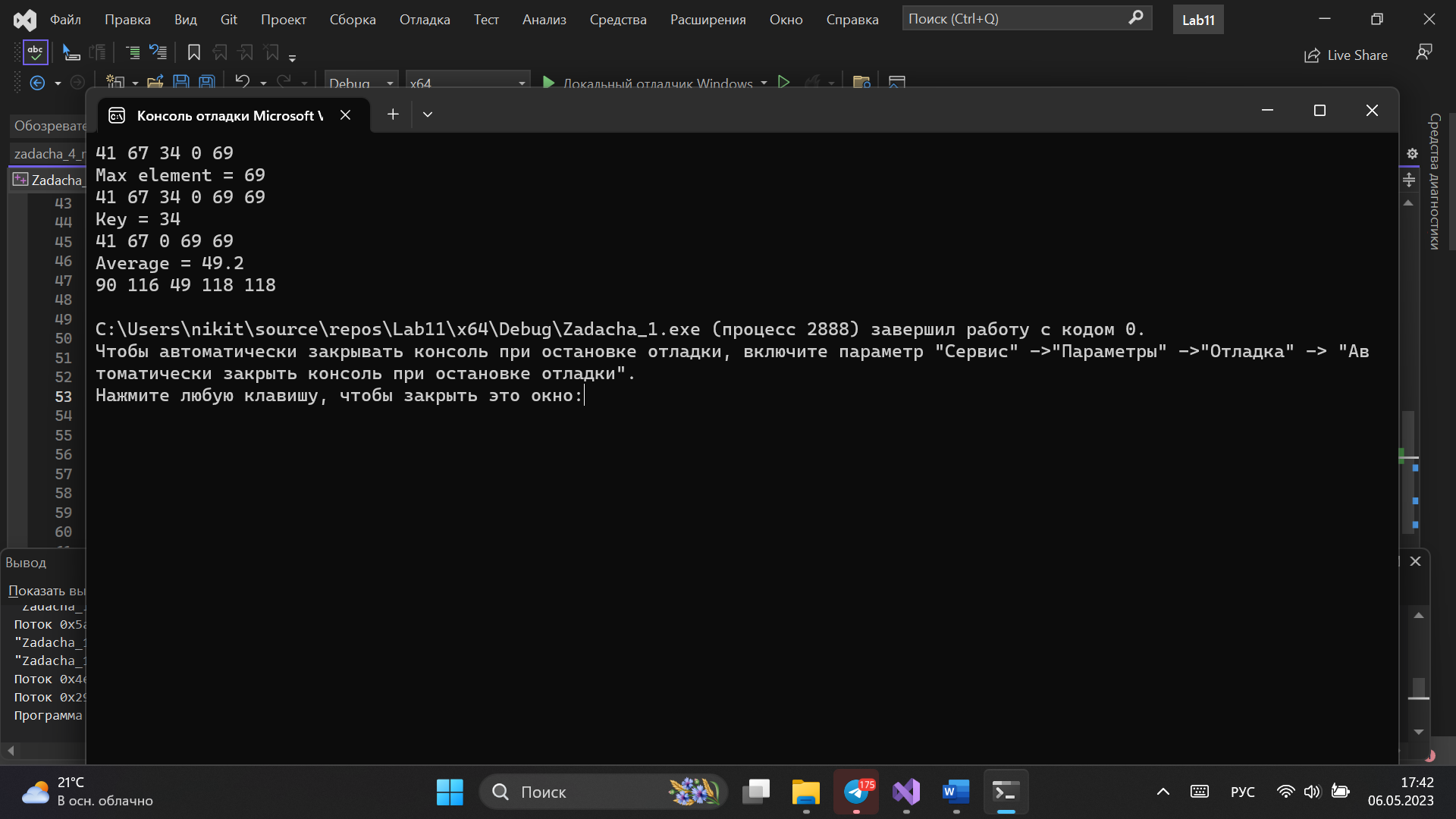
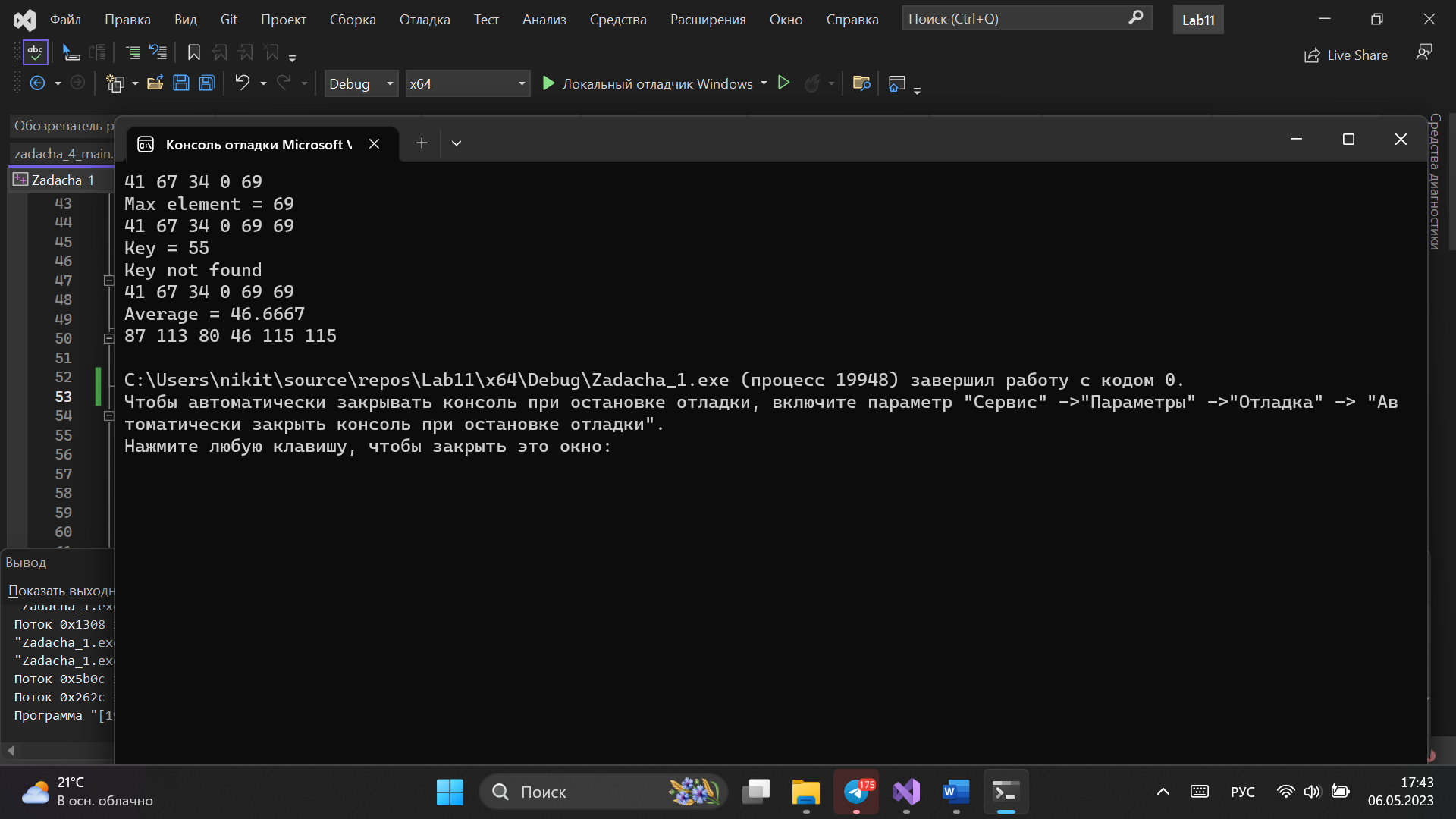
}

**Объяснение результатов работы программы.**

В данном коде представлено создание двунаправленной очереди и заполнение его случайными числами от 0 до 99, так же присутствует функция вывода на экран двунаправленной очереди. В основной функции выполняются поставленные задания: нахождение максимального

элемента и добавления его в конец контейнера, нахождение элемента с

заданным ключом и удаление его из контейнера, добавления к каждому элементу среднего арифметического элементов контейнера.

**Описание пользовательского класса для решения задачи 2.**

class Money

{

long rub; int cop;

public:

Money() { rub = 0; cop = 0; }; //конструктор без параметров

Money(long r, int c) { rub = r; cop = c; } //конструктор с параметрами

Money(const Money& m) { rub = m.rub; cop = m.cop; } //конструктор копирования

~Money() {}; //деструктор

int get\_rub() { return rub; } //селектор

int get\_cop() { return cop; } //селектор

void set\_rub(long r) { rub = r; } //модификатор

void set\_cop(int c) { cop = c; } //модификатор

//перегруженные операции

Money& operator=(const Money&); //перегрузка операции присваивания

Money& operator++(); //перегрузка префиксной операции инкремент

Money operator++(int); //постфиксная операция

Money operator+(const Money&) const; //перегрузка бинарной операции сложения

Money operator-(const Money&); //перегрузка бинарной операции вычитания

Money operator/(const int&);

bool operator==(const Money&); //перегрузка операции сравнения

bool operator!=(const Money&); //перегрузка операции сравнения

bool operator>(const Money&); //перегрузка операции сравнения

//глобальные функции ввода-вывода

friend istream& operator>>(istream& in, Money& m);

friend ostream& operator<<(ostream& out, const Money& m);

};

****

**Определение перегруженных операций для пользовательского класса.**

//перегрузка операции присваивания

Money& Money::operator=(const Money& m)

{

//проверка на самоприсваивание

if (&m == this) return \*this;

rub = m.rub;

cop = m.cop;

return \*this;

}

//перегрузка префиксной операции инкремент

Money& Money::operator++()

{

long e = rub + (cop / 100);

rub = e;

cop = cop % 100;

return \*this;

}

//перегрузка постфиксной операции инкремент

Money Money::operator ++(int)

{

long e = rub + (cop / 100);

Money t(rub, cop);

rub = e;

cop = cop % 100;

return t;

}

//перегрузка бинарной операции сложения

Money Money::operator+(const Money& m) const

{

long rubs = rub + m.rub;

int cops = cop + m.cop;

if (cops >= 100) { // обработка случая, когда cop >= 100

rubs += 1;

cops -= 100;

}

Money p(rubs, cops);

return p;

}

//перегрузка бинарной операции вычитания

Money Money::operator-(const Money& m)

{

double total = rub + cop / 100.0 - (m.rub + m.cop / 100.0);

long rub = static\_cast<long>(total); // static\_cast: оператор преобразования типов данных

int cop = static\_cast<int>((total - rub) \* 100);

if (cop < 0) { // обработка случая, когда cop < 0

rub -= 1;

cop += 100;

}

if (rub < 0) { // обработка случая, когда rub < 0

rub = 0;

cop = 0;

}

Money p(rub, cop);

return p;

}

Money Money::operator/(const int& m)

{

double total = (rub + cop / 100.0) / m;

long rub = static\_cast<long>(total); // static\_cast: оператор преобразования типов данных

int cop = static\_cast<int>((total - rub) \* 100);

if (cop < 0) { // обработка случая, когда cop < 0

rub -= 1;

cop += 100;

}

if (rub < 0) { // обработка случая, когда rub < 0

rub = 0;

cop = 0;

}

Money p(rub, cop);

return p;

}

//перегрузка операции сравнения ==

bool Money::operator==(const Money& m)

{

if (rub == m.rub && cop == m.cop) { return true; }

else { return false; }

}

//перегрузка операции сравнения !=

bool Money::operator!=(const Money& m)

{

if (rub != m.rub || cop != m.cop) { return true; }

else { return false; }

}

bool Money::operator>(const Money& m)

{

return (rub > m.rub) || (rub == m.rub && cop > m.cop);

}

//перегрузка глобальной функции-операции ввода

istream& operator>>(istream& in, Money& m)

{

cout << "Введите рубли = "; in >> m.rub;

cout << "Введите копейки = "; in >> m.cop;

cout << endl;

return in;

}

//перегрузка глобальной функции-операции вывода

ostream& operator<<(ostream& out, const Money& m)

{

return (out << m.rub << "," << m.cop);

}

**Функции для решения задачи 2.**

Deq make\_deque(int n)

{

Deq d;

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

{

long r = rand() % 100000;

int c = rand() % 100;

Money m(r, c);

d.push\_back(m);

}

return d;

}

void print\_deque(Deq d)

{

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

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

cout << endl;

}

// Найти максимальный элемент и добавить его в конец контейнера

void Max(Deq& d)

{

if (d.empty()) {

cout << "Контейнер пуст" << endl;

return;

}

Money max\_element = d.front();

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

if (d[i] > max\_element) {

max\_element = d[i];

}

}

cout << "Максимальный элемент = " << max\_element << endl;

d.push\_back(max\_element);

}

// Найти элемент с заданным ключом и удалить его из контейнера

void remove(Deq& d, Money key)

{

int index = -1;

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

if (d[i] == key) {

index = i;

break;

}

}

if (index != -1) {

d.erase(d.begin() + index);

cout << "Элемент удален" << endl;

}

else {

cout << "Элемент не найден" << endl;

}

}

Deq average(Deq& d)

{

Money m = Money(0, 0);

if (d.empty()) {

cout << "Контейнер пуст" << endl;

}

else

{

for (int i = 0; i < d.size(); i++) { m = m + d[i]; }

int n = d.size();

Money r = m / n;

cout << "Среднее значение = " << r << endl;

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

d[i] = r + d[i];

}

return d;

}

}

**Основная программа для решения задачи 2**

int main()

{

long r; int c;

srand(time(0));

setlocale(LC\_ALL, "rus");

Deq d = make\_deque(10);

cout << "Исходный контейнер: ";

print\_deque(d);

Max(d);

print\_deque(d);

cout << "Введите рубли и копейки для ключа: " << endl;

cin >> r; cin >> c;

Money s(r, c);

remove(d, s);

print\_deque(d);

average(d);

print\_deque(d);

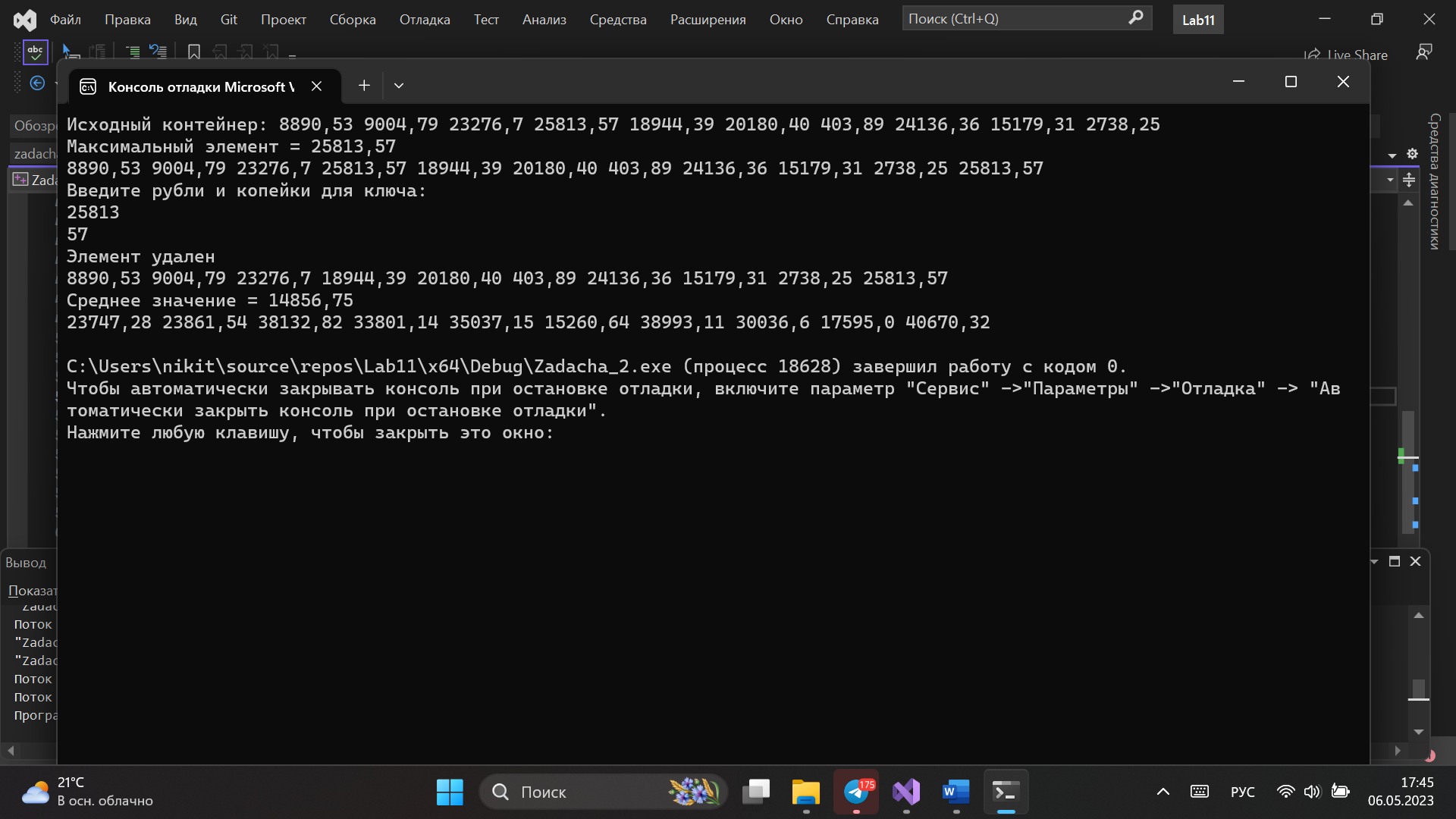
}

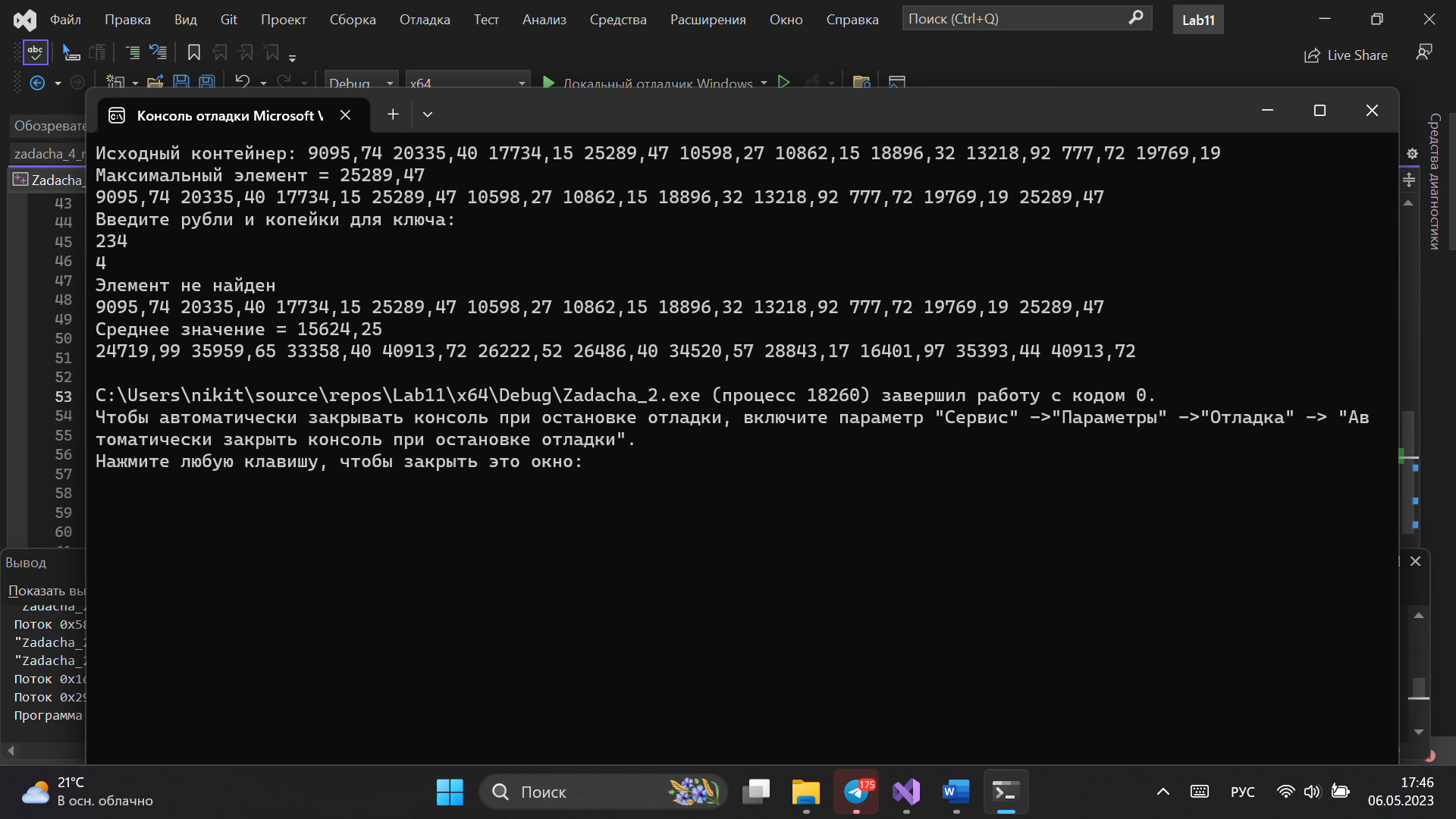
**Объяснение результатов работы программы.**

В данной программе представлена двунаправленная очередь, которая в качестве параметра принимает пользовательский класс Money и выполняются поставленные задания: нахождение максимального

элемента и добавления его в конец контейнера, нахождение элемента с

заданным ключом и удаление его из контейнера, добавления к каждому элементу среднего арифметического элементов контейнера.





**Описание параметризированного класса для решения задачи 3.**

//шаблон класса

template<class T>

class Vector

{

vector <T> v;//последовательный контейнер для хранения элементов вектора

int len;

public:

Vector(void);//конструктор без параметра

Vector(int n);

T& operator[](int index);

//конструктор с параметром

void Print();//печать

~Vector(void);//деструктор

T Srednee();//вычисление среднего арифметического

void Add(T el);//добавление элемента el в конец

int Max();//найти номер максимальнго элемента

void Del(int pos);//удалить элемент из позиции pos

void Add\_srednee();//добавить среднее арифметическое

};



**Определение методов и операций для решения задачи 3.**

//конструктор без параметра

template <class T>

Vector<T>::Vector()

{

len = 0;

}

//деструктор

template <class T>

Vector<T>::~Vector(void)

{

}

//конструктор с параметром

template <class T>

Vector<T>::Vector(int n)

{

T a;

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

{

cin >> a;

v.push\_back(a);

}

len = v.size();

}

//операция доступа по индексу

template <class T>

T& Vector<T>::operator[](int index)

{

if (index < v.size()) return v[index];

else cout << "\nError! Index>size";

}

//печать

template <class T>

void Vector<T>::Print()

{

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

cout << v[i] << " ";

cout << endl;

}

template<class T>

T Vector<T>::Srednee()

{

T s = v[0];

for (int i = 1; i < v.size(); i++)

s = s + v[i];

int n = v.size();

return s / n;

}

//добавление элемента

template<class T>

void Vector<T>::Add(T el)

{

v.insert(v.begin() + v.size(), el);

}

//поиск максимального элемента

template <class T>

int Vector<T>::Max()

{

T m = v[0];

int n = 0;

for (int i = 1; i < v.size(); i++)

if (v[i] > m)

{

m = v[i];

n = i;

}

return n;

}

//удаление элемента из позиции pos

template<class T>

void Vector<T>::Del(int pos)

{

v.erase(v.begin() + pos);

}

// Функция добавления к каждому элементу среднее арифметическое

template<class T>

void Vector<T>::Add\_srednee()

{

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

v[i] = v[i] + Srednee();

}

**Основная программа для решения задачи 3.**

void main()

{

int p, s;

setlocale(LC\_ALL, "rus");

Vector<Money>vec(5);//создать вектор из 5 элементов

vec.Print();//печать вектора

p = vec.Max();//найти максимальный элемент

cout << "Максимальный элемент = " << vec[p] << endl;

vec.Add(vec[p]); //добавление элемента

vec.Print(); //печать вектора

cout << "Введите ключ = "; cin >> s;

cout << "Ваш элемент = " << vec[s] << endl;

vec.Del(s);//удаление

vec.Print(); //печать вектора

Money m = vec.Srednee();//среднее ариметическое

cout << "Среднее значение = " << m << endl;

vec.Add\_srednee();//добавить среднее арифметическое

vec.Print(); //печать вектора

}

**Объяснение результатов работы программы.**

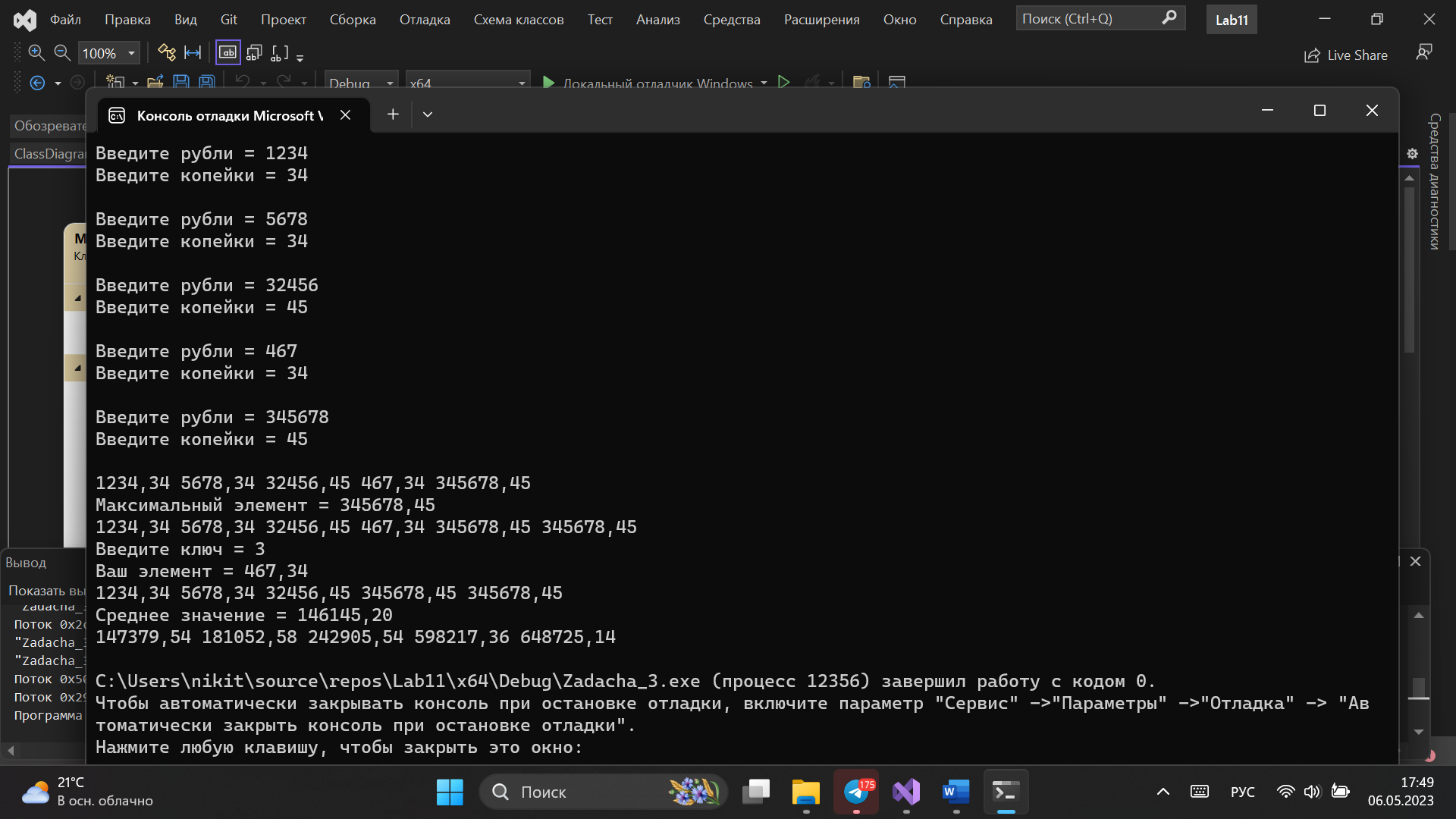
В данной программе создается параметризированный класс – вектор,

В котором представлены методы: нахождение максимального

элемента и добавления его в конец контейнера, нахождение элемента с

заданным ключом и удаление его из контейнера, добавления к каждому элементу среднего арифметического элементов контейнера.

В основной функции создает обьект класса Вектор, который в качестве параметра принимает пользовательский класс Money и вызываются функции для решения поставленных заданий.



**Функции для решения задачи 4.**

St make\_stack(int n)

{

St s;

Money m;

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

{

cin >> m;//ввод переменной

s.push(m);//добавление ее в стек

}

return s;//вернуть стек как результат функции

}

//копирует стек в вектор

Vec copy\_stack\_to\_vector(St s)

{

Vec v;

while (!s.empty())//пока стек не пустой

{

//добавить в вектор элемент из вершиы стека

v.push\_back(s.top());

s.pop();

}

return v; //вернуть вектор как результат функции

}

//копирует вектор в стек

St copy\_vector\_to\_stack(Vec v)

{

St s;

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

{

s.push(v[i]);//добавить в стек элемент вектора

}

return s; //вернуть стек как результат функции

}

//Вывод стека

void print\_stack(St s)

{

while (!s.empty()) // пока стек не пустой

{

cout << s.top() << " "; // выводим верхний элемент стека

s.pop(); // удаляем верхний элемент из стека

}

cout << endl; // переходим на новую строку

}

//вычисление среднего значения

Money Srednee(St s)

{

Vec v = copy\_stack\_to\_vector(s);//копируем стек в вектор

int n = 1;

Money sum = s.top();//начальное значениедля суммы

s.pop();//удалить первый элемент из вектора

while (!s.empty())//пока стек не пустой

{

sum = sum + s.top();//добавить в сумму элемент из вершины стека

s.pop();//удалить элемент

n++;

}

s = copy\_vector\_to\_stack(v);//скопировать вектор в стек

return sum / n; //вернуть среднее арифметическое

}

//поиск максимального элемента в стеке

Money Max(St s)

{

Money m = s.top();//переменной m присваиваем значение из вершины стека

Vec v = copy\_stack\_to\_vector(s);//копируем стек в вектор

while (!s.empty())//пока стк не пустой

{

if (s.top() > m)m = s.top();//срвниваем m и элемнт в вершине стека

s.pop();//удаляем элемент из стека

}

s = copy\_vector\_to\_stack(v);//копиуем вектор в стек

return m; //возвращаем m

}

//добавление элемента в стек

void Add\_to\_stack(St& s, Money el, int pos)

{

Vec v;

Money t;

int i = 1;//номер элемента в стеке

while (!s.empty())//пока стек не пустой

{

t = s.top();//получить элемент из вершины

//если номер равен номеру позиции, на котоую добвляем элемент

if (i == pos)v.push\_back(el);//добавить новый элемент в вектор

v.push\_back(t);//добавить элемент из стека в вектор

s.pop();//удалить элемент из стека

i++;

}

s = copy\_vector\_to\_stack(v);//копируем вектор в стек

}

//Нахождение элемента с заданным ключом и удаление его из контейнера

bool removeByKey(St& stack, const Money& k) {

St tempStack; // временный стек для хранения элементов

bool found = false; // флаг, указывающий на то, найден ли элемент

while (!stack.empty()) {

Money m = stack.top();

stack.pop();

if (m == k) {

found = true;

break;

}

else {

tempStack.push(m);

}

}

while (!tempStack.empty()) {

stack.push(tempStack.top());

tempStack.pop();

}

return found;

}

//добавление среднего арифметического контейнера к каждому элементу

void addAverageToStack(St& stack, const Money& t) {

Vec vec;

while (!stack.empty()) {

Money m = stack.top();

stack.pop();

vec.push\_back(m);

}

for (const auto& m : vec) {

Money M = t + m;

stack.push(M);

}

}

**Основная программа для решения задачи 4**

void main()

{

setlocale(LC\_ALL, "rus");

Money m, t;

St s;

int n, a; long r; int c;

cout << "Введите n = "; cin >> n;

s = make\_stack(n);

print\_stack(s);

m = Max(s);

cout << "Максимальный элемент = " << m << endl;

s.push(m); //добавление в конец очереди

print\_stack(s);

cout << "Введите рубли и копейки для ключа: " << endl;

cin >> r; cin >> c;

Money q(r, c);

if (removeByKey(s, q)) cout << "Элемент " << q << " найден и удален" << endl;

else cout << "Элемент не найден" << endl;

print\_stack(s);

t = Srednee(s);

cout << "Среднее значение " << t << endl << "Добавили к каждому элементу" << endl;

addAverageToStack(s, t);

print\_stack(s);

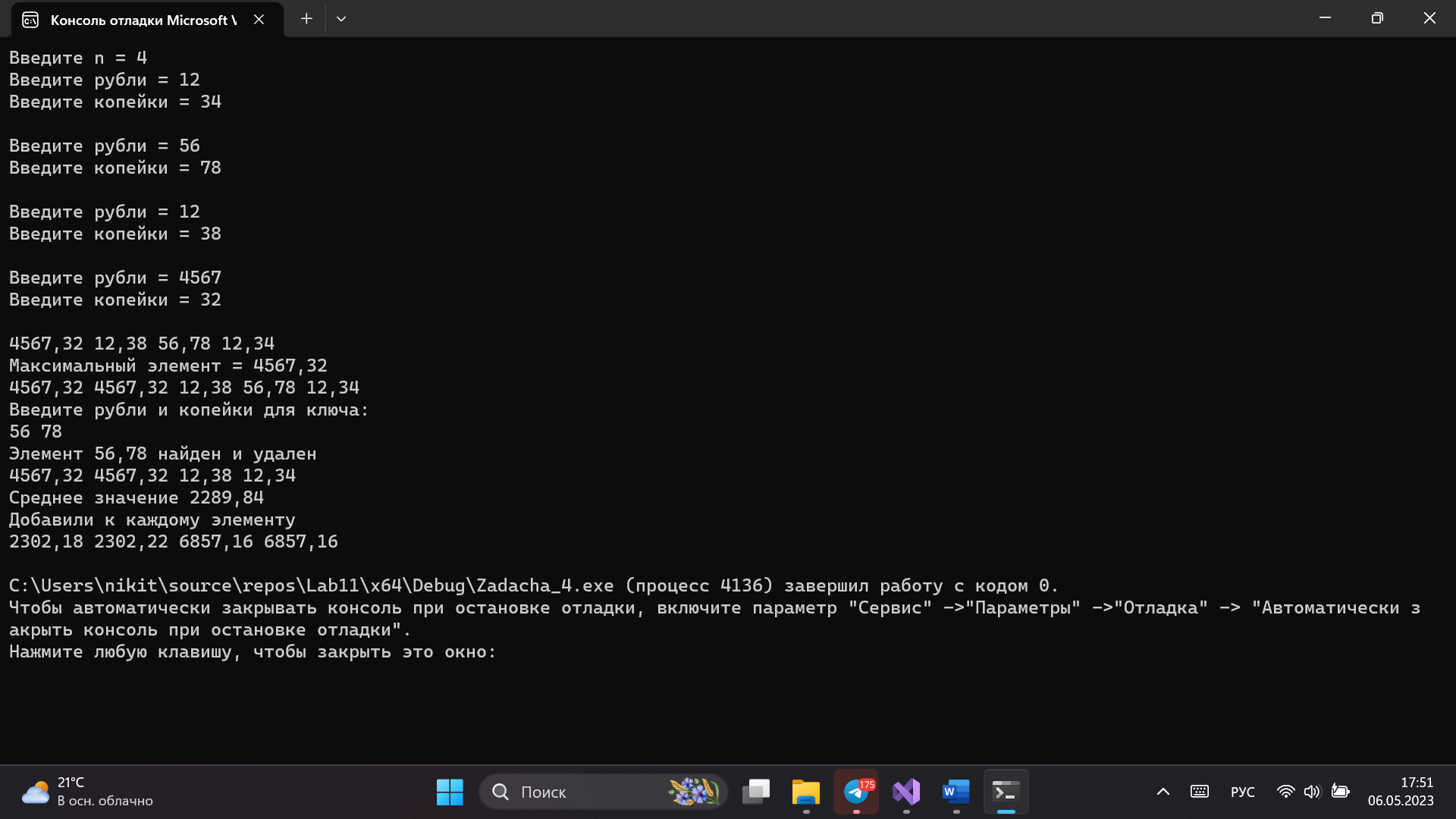
}

**Объяснение результатов работы программы.**

В данной программе представлен, адаптер контейнера – стек, функции необходимые для создания, где в качестве параметра принимается пользовательский класс Money, так же функция печати стека и необходимые функции для выполнения заданий: нахождение максимального

элемента и добавления его в конец контейнера, нахождение элемента с

заданным ключом и удаление его из контейнера, добавления к каждому элементу среднего арифметического элементов контейнера.



**Описание параметризированного класса для решения задачи 5.**

template<class T>

class Vector

{

stack <T> s;//контейнер

int len;//размер контейнера

public:

Vector();//конструктор без параметров

Vector(int n);//конструктор с параметрами

Vector(const Vector<T>&);//конструктор копирования

void Print();

T Srednee();

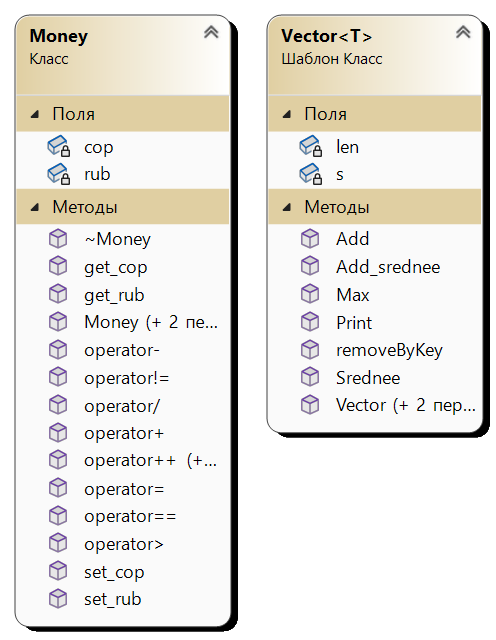
void Add(T el);

T Max();

T removeByKey(T el);

void Add\_srednee(T el);

};



**Определение методов и операций для решения задачи 5.**

//копирование стека в вектор

template <class T>

vector<T> copy\_stack\_to\_vector(stack<T> s)

{

vector<T> v;

while (!s.empty())

{

v.push\_back(s.top());

s.pop();

}

return v;

}

//копирование вектора в стек

template <class T>

stack<T> copy\_vector\_to\_stack(vector<T> v)

{

stack<T> s;

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

{

s.push(v[i]);

}

return s;

}

//конструктор без параметров

template <class T>

Vector<T>::Vector() { len = 0; }

//конструктор с параметром

template <class T>

Vector<T>::Vector(int n)

{

T a;

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

{

cin >> a;

s.push(a);//добавить в стек элемент а

}

len = s.size();

}

//конструктор копирования

template <class T>

Vector<T>::Vector(const Vector<T>& v)

{

len = v.len;

// копируем значения стека v.s в вектор v

std::vector<T> vec = copy\_stack\_to\_vector(v.s);

// копируем вектор v в стек s

s = copy\_vector\_to\_stack(vec);

}

//печать

template <class T>

void Vector<T>::Print()

{

//копируем стек в вектор

vector<T> v = copy\_stack\_to\_vector(s);

while (!s.empty())//пока стек не пустой

{

cout << s.top() << endl;//вывод элемента в вершине стека

s.pop();//удаляем элемент из вершины

}

//копируем вектор в стек

s = copy\_vector\_to\_stack(v);

}

//вычисление среднего значения

template <class T>

T Vector<T>::Srednee()

{

//копируем стек в вектор

vector<T> v = copy\_stack\_to\_vector(s);

int n = 1;//количество элементов в стеке

T sum = s.top();//начальное значение для суммы

s.pop();//удаляем элемент из вершины стека

while (!s.empty())//пока стек не пустой

{

sum = sum + s.top();//добавляем в сумму элемент из вершины стека

s.pop();//удаляем элемент из вершины стека

n++;//увеличиваем количество элементов

}

//копируем вектор в стек

s = copy\_vector\_to\_stack(v);

return sum / n;

}

//добавление элемента el в стек в конец

template <class T>

void Vector<T>::Add(T el) { s.push(el); }

//поиск максимального элемента

template <class T>

T Vector<T>::Max()

{

T m = s.top();//m присвоить значение из вершины стека

//в вектор скопировать элементы стека

vector<T> v = copy\_stack\_to\_vector(s);

while (!s.empty())//пока стек не пустой

{

//сравниваем m и элемент в вершине стека

if (s.top() > m)m = s.top();

s.pop();//удаляем элемент из вершины стека

}

s = copy\_vector\_to\_stack(v);//копируем вектор в стек

return m;

}

//Нахождение элемента с заданным ключом и удаление его из контейнера

template <class T>

T Vector<T>::removeByKey(T el)

{

T m = el;

vector<T> v;

T t;

while (!s.empty())

{

t = s.top();

if (t != m)v.push\_back(t);

s.pop();

}

s = copy\_vector\_to\_stack(v);

return el;

}

//Добавление среднего арифметического элементов контейнера

template <class T>

void Vector<T>::Add\_srednee(T el)

{

T m = el;

vector<T> v;

T t;

while (!s.empty())

{

t = s.top();

v.push\_back(t + m);

s.pop();

}

s = copy\_vector\_to\_stack(v);

}

**Основная программа для решения задачи 2**

void main()

{

setlocale(LC\_ALL, "rus");

Money s, t;

long r; int c;

Vector<Money>v(3);

v.Print();

Money m = v.Max();

cout << "Максимальный элемент = " << m << endl;

v.Add(m);

v.Print();

cout << "Введите рубли и копейки для ключа: " << endl;

cin >> r; cin >> c;

Money q(r, c);

s = v.removeByKey(q);

cout << "Элемент " << s << " удален" << endl;

v.Print();

t = v.Srednee();

cout << "Среднее значение = " << t << endl;

v.Add\_srednee(t);

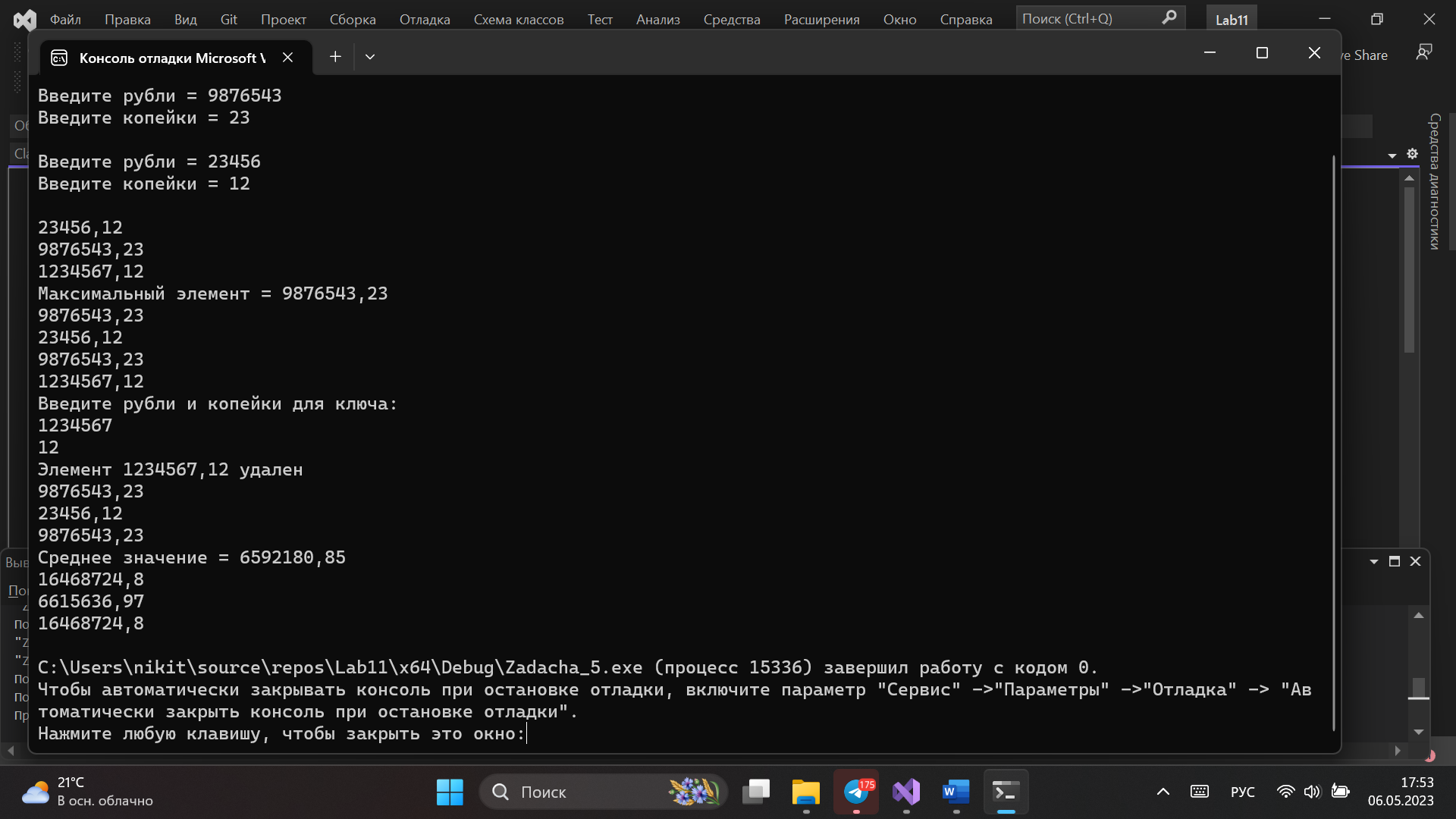
v.Print();

}

**Объяснение результатов работы программы.**

В данной программе представлен, параметризированный класс – Вектор с адаптером контейнера – стек, функции необходимые для создания, так же функция печати вектора и необходимые функции для выполнения заданий: нахождение максимального элемента и добавления его в конец контейнера, нахождение элемента с заданным ключом и удаление его из контейнера, добавления к каждому элементу среднего арифметического элементов контейнера.

В главной функции создается объект класса Вектор, где в качестве параметра используется пользовательский класс Money и вызовы функций для решения поставленных заданий.



**Ответы на контрольные вопросы**

**1. Из каких частей состоит библиотека STL?**

**2. Какие типы контейнеров существуют в STL?**

**3. Что нужно сделать для использования контейнера STL в своей программе?**

**4. Что представляет собой итератор?**

**5. Какие операции можно выполнять над итераторами?**

**6. Каким образом можно организовать цикл для перебора контейнера с**

**использованием итератора?**

**7. Какие типы итераторов существуют?**

**8. Перечислить операции и методы общие для всех контейнеров.**

**9. Какие операции являются эффективными для контейнера vector? Почему?**

**10. Какие операции являются эффективными для контейнера list? Почему?**

**11. Какие операции являются эффективными для контейнера deque? Почему?**

**12. Перечислить методы, которые поддерживает последовательный контейнер**

**vector.**

**13. Перечислить методы, которые поддерживает последовательный контейнер list.**

**14. Перечислить методы, которые поддерживает последовательный контейнер**

**deque.**

**15. Задан контейнер vector. Как удалить из него элементы со 2 по 5?**

**16. Задан контейнер vector. Как удалить из него последний элемент?**

**17. Задан контейнер list. Как удалить из него элементы со 2 по 5?**

**18. Задан контейнер list. Как удалить из него последний элемент?**

**19. Задан контейнер deque. Как удалить из него элементы со 2 по 5?**

**20. Задан контейнер deque. Как удалить из него последний элемент?**

**21. Написать функцию для печати последовательного контейнера с использованием**

**итератора.**

**22. Что представляют собой адаптеры контейнеров?**

**23. Чем отличаются друг от друга объявления stack<int> s и stack<int, list<int> > s?**

**24. Перечислить методы, которые поддерживает контейнер stack.**

**25. Перечислить методы, которые поддерживает контейнер queue.**

**26. Чем отличаются друг от друга контейнеры queue и priority\_queue?**

**27. Задан контейнер stack. Как удалить из него элемент с заданным номером?**

**28. Задан контейнер queue. Как удалить из него элемент с заданным номером?**

**29. Написать функцию для печати контейнера stack с использованием итератора.**

**30. Написать функцию для печати контейнера queue с использованием итератора.**

1. Библиотека STL состоит из трех частей: контейнеры, алгоритмы и итераторы.

2. В STL существует несколько типов контейнеров: vector, deque, list, set, map и другие.

3. Для использования контейнера STL в своей программе необходимо включить заголовочный файл, соответствующий данному контейнеру. Например, для использования контейнера vector нужно добавить

#include <vector>.

4. Итератор - это объект, который позволяет осуществить доступ к элементам контейнера и перебрать его содержимое.

5. С помощью итераторов можно выполнять операции: перебор значений, получение доступа к начальному и конечному элементам контейнера, перемещение указателя на следующий или предыдущий элемент и многое другое.

6. Для перебора контейнера с использованием итератора можно использовать цикл for с оператором auto. Например, для перебора контейнера vector это будет выглядеть следующим образом:

for(auto it = vec.begin(); it != vec.end(); ++it) {

// операции с элементами контейнера

}

7. В STL существуют три типа итераторов: внешний, внутренний и обратный.

8. Операции и методы, общие для всех контейнеров: size(), empty(), insert(), erase(), clear(), begin(), end(), rbegin(), rend().

9. Эффективными операциями для контейнера vector являются: добавление элемента в конец вектора, доступ к элементам по индексу и перебор элементов с помощью итератора. Это связано с тем, что вектор - это контейнер с произвольным доступом и быстрым добавлением элемента в конец.

10. Эффективными операциями для контейнера list являются: вставка элемента в список в любое место, удаление элемента из списка и перебор элементов с помощью итератора. Это связано с тем, что список - это контейнер с последовательным доступом и быстрой вставкой/удалением элементов в любую точку списка.

11. Эффективными операциями для контейнера deque являются: добавление/удаление элемента в начале и конце дека, доступ и изменение элементов по индексу. Это связано с тем, что deque - это контейнер со случайным доступом и возможностью быстрой вставки/удаления элементов в начале и конце дека.

12. Некоторые методы, поддерживаемые контейнером vector: pushback(), popback(), size(), resize(), capacity(), clear(), insert(), erase(), begin(), end(), rbegin(), rend().

13. Некоторые методы, поддерживаемые контейнером list: pushback(), pushfront(), popback(), popfront(), size(), resize(), clear(), insert(), erase(), begin(), end(), rbegin(), rend().

14. Некоторые методы, поддерживаемые контейнером deque: pushback(), pushfront(), popback(), popfront(), size(), resize(), clear(), insert(), erase(), begin(), end(), rbegin(), rend().

15. Чтобы удалить из контейнера vector элементы со 2 по 5, можно использовать метод erase() и передать ему итераторы на эти элементы:

vec.erase(vec.begin() + 1, vec.begin() + 5);

16. Чтобы удалить из контейнера vector последний элемент, можно использовать метод popback():

vec.popback();

17. Чтобы удалить из контейнера list элементы со 2 по 5, можно использовать метод erase() и передать ему итераторы на эти элементы:

auto itstart = std::next(lst.begin(), 1); // итератор на второй элемент

auto itend = std::next(lst.begin(), 5); // итератор на шестой элемент

lst.erase(itstart, itend);

18. Чтобы удалить из контейнера list последний элемент, можно использовать метод popback():

lst.popback();

19. Чтобы удалить из контейнера deque элементы со 2 по 5, можно использовать метод erase() и передать ему итераторы на эти элементы:

deque.erase(deque.begin() + 1, deque.begin() + 5);

20. Чтобы удалить из контейнера deque последний элемент, можно использовать метод popback():

deque.popback();

21. Пример функции для печати последовательного контейнера с использованием итератора:

template <typename T>

void print\_container(const T& container) {

for (auto it = container.begin(); it != container.end(); ++it) {

cout << \*it << " ";

}

cout << endl;

}

22. Адаптеры контейнеров - это классы, которые обеспечивают интерфейс специфического контейнера на основе другого контейнера. Они представляют общий интерфейс для работы с различными контейнерами, предоставляя заданный набор методов и функций. Примеры адаптеров: stack, queue, priorityqueue.

23. `stack<int> s` объявляет стек на основе контейнера std::deque<int>, а `stack<int, list<int> > s` объявляет стек на основе контейнера std::list<int>.

24. Методы, которые поддерживает контейнер stack:

- push() - добавляет элемент в стек

- pop() - удаляет элемент из вершины стека

- top() - возвращает элемент, находящийся на вершине стека

- empty() - возвращает true, если стек пустой, false - в противном случае

- size() - возвращает количество элементов в стеке

25. Методы, которые поддерживает контейнер queue:

- push() - добавляет элемент в конец очереди

- pop() - удаляет элемент из начала очереди

- front() - возвращает элемент, находящийся в начале очереди

- back() - возвращает элемент, находящийся в конце очереди

- empty() - возвращает true, если очередь пустая, false - в противном случае

- size() - возвращает количество элементов в очереди

26. Контейнер queue представляет обычную очередь, где элементы заносятся в конец и вынимаются из начала очереди последовательно. Контейнер priorityqueue представляет приоритетную очередь, где элементы заносятся в определенном порядке, а получение элементов осуществляется согласно заданному приоритету.

27. Для удаления элемента с заданным номером из контейнера stack необходимо сделать несколько шагов:

1. Создать временный стек и перенести в него нужное количество элементов из исходного стека, оставив нужный элемент в качестве вершины временного стека.

2. Удалить выбранный элемент.

3. Перенести все элементы, находящиеся в временном стеке, обратно в исходный стек.

28. Для удаления элемента с заданным номером из контейнера queue необходимо сделать несколько шагов:

1. Создать временную очередь и перенести в нее нужное количество элементов из исходной очереди, оставив удаленный элемент за концом временной очереди.

2. Удалить выбранный элемент.

3. Перенести все элементы, находящиеся во временной очереди, обратно в исходную очередь.

29. Пример функции для печати контейнера stack с использованием итератора:

template <typename T>

void print\_stack(const std::stack<T>& s) {

stack<T> temp = s;

while (!temp.empty()) {

scout << temp.top() << " ";

temp.pop();

}

cout << endl;

}

30. Пример функции для печати контейнера queue

template <typename T>

void print\_queue(const std::queue<T>& q) {

queue<T> temp = q;

while (!temp.empty()) {

cout << temp.front() << " ";

temp.pop();

}

cout << endl;

}