Пермский национальный исследовательский политехнический университет

**Лабораторная работа №12**

**Дисциплина: Основы алгоритмизации и**

**программирования**

**Вариант № 9**

**"Ассоциативные 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) 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 ассоциативных контейнеров библиотеки STL в ОО программе.

3) **Задача 1**

1. Контейнер - set

2. Тип элементов - int

**Задача 2**

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

**Задача 3**

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

|  |  |  |
| --- | --- | --- |
| **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.**

TSet make\_set(int n)

{

TSet s;

int a;

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

{

cout << "Введите значение: ";

cin >> a;

s.insert(a);

}

return s;

}

void print\_set(TSet s)

{

for (auto i = s.begin(); i != s.end(); i++)

cout << \*i << " ";

cout << endl;

}

void add\_max\_element(TSet s)

{

if (s.empty()) return;

int max\_elem = \*max\_element(s.begin(), s.end());

cout << "Максимальный элемент = " << max\_elem << ". ";

s.insert(max\_elem);

}

TSet remove\_element(TSet s, int key)

{

auto it = s.find(key);

if (it != s.end()) s.erase(it);

return s;

}

TSet add\_average(TSet& s)

{

int average = accumulate(s.begin(), s.end(), 0) / s.size();

cout << "Средние значение = " << average << endl;

TSet e;

transform(s.begin(), s.end(), inserter(e, e.end()),

[average](int element) { return element + average; });

return e;

}

**Основная программа для решения задачи 1**

int main()

{

setlocale(LC\_ALL, "rus");

int n;

TSet s = make\_set(5);

cout << "Элементы: ";

print\_set(s);

add\_max\_element(s);

cout << "Максимальный элемент добавлен в конец: ";

print\_set(s);

cout << "Ключ = "; cin >> n;

s = remove\_element(s, n);

cout << "Элемент удален: ";

print\_set(s);

s = add\_average(s);

cout << "Добавлено среднее значение: ";

print\_set(s);

return 0;

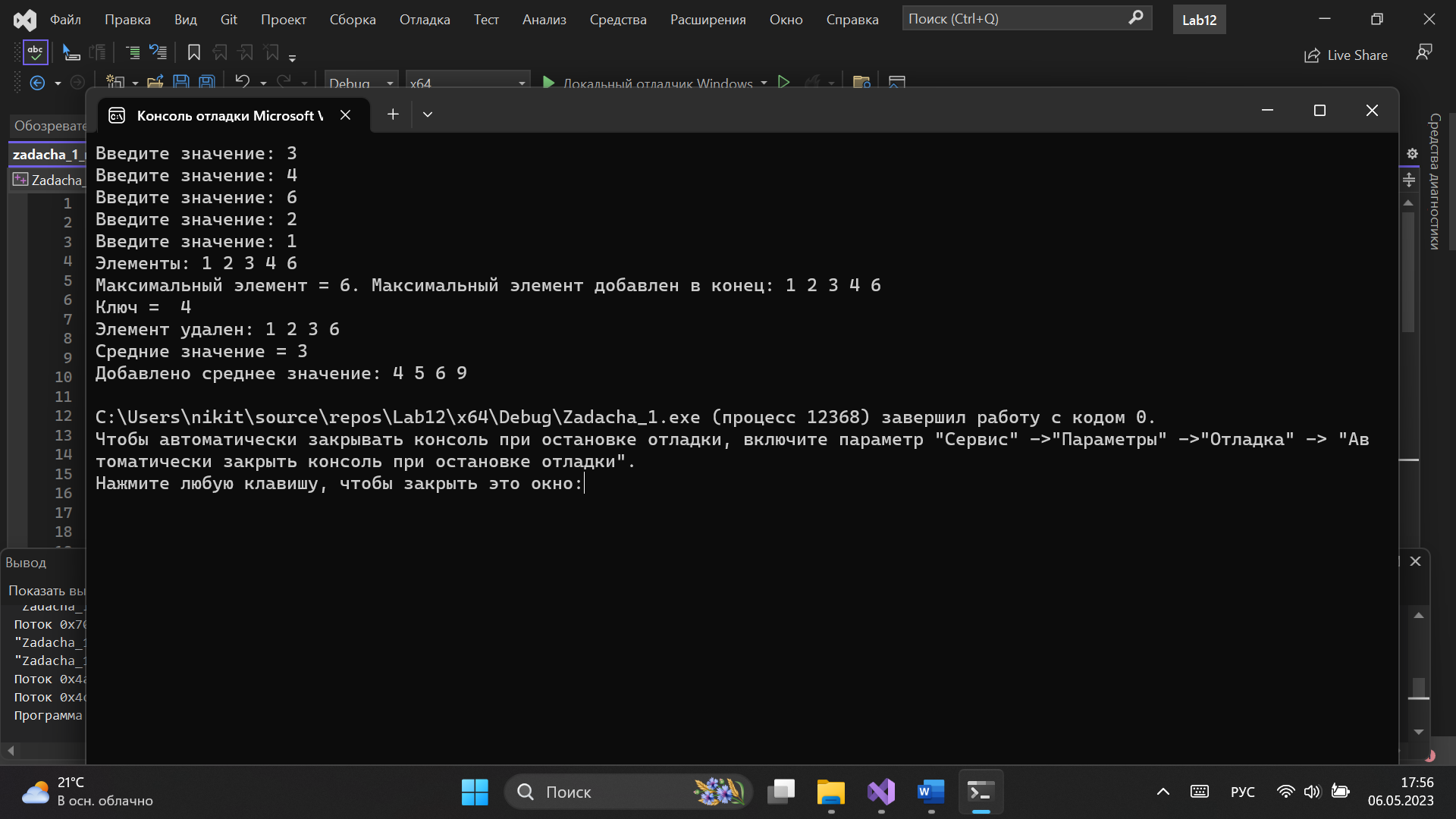
}

**Объяснение результатов работы программы.**

В данном коде создается контейнер Set с типом элементов int, представлены необходимые функции для создания контейнера, печати и выполнения всех 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 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

Money& operator=(const Money& m);

//перегруженные операции

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

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

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

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

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

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

bool operator<(const Money& m) const;

Money operator/(int divisor) const {

long total\_cop = this->rub \* 100 + this->cop;

total\_cop /= divisor;

long new\_rub = total\_cop / 100;

int new\_cop = total\_cop % 100;

return Money(new\_rub, new\_cop);

}

Money& operator+=(Money& other) {

rub += other.rub;

cop += other.cop;

if (cop >= 100) {

rub += cop / 100;

cop %= 100;

}

return \*this;

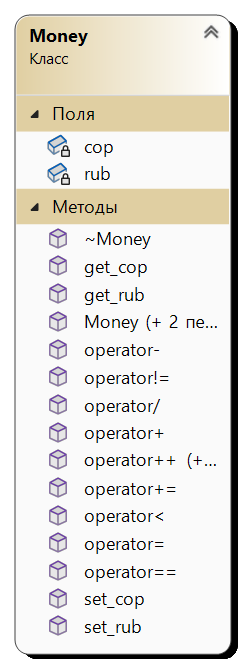
}

//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)

{

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;

}

//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) const

{

if (rub < m.rub) { return true; }

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

return false;

}

//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.**

TSet make\_set(int n)

{

TSet s;

Money a;

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

{

cin >> a;

s.insert(a);

}

return s;

}

void print\_set(const TSet& s)

{

for (auto i = s.begin(); i != s.end(); i++)

cout << \*i << " ";

cout << endl;

}

void add\_max\_element(TSet& s)

{

if (s.empty()) return;

Money max\_m = \*max\_element(s.begin(), s.end());

cout << "Максимальный элемент = " << max\_m << ". ";

s.insert(max\_m);

}

TSet remove\_element(TSet& s, const Money& key)

{

auto it = s.find(key);

if (it != s.end()) s.erase(it);

return s;

}

TSet add\_average(TSet& s)

{

Money sum = accumulate(s.begin(), s.end(), Money(0,0));

Money average = sum / s.size();

cout << "Средние значение = " << average << endl;

TSet e;

transform(s.begin(), s.end(), inserter(e, e.end()),

[average](Money element) { return element + Money(average); });

return e;

}

**Основная программа для решения задачи 2**

int main()

{

setlocale(LC\_ALL, "rus");

int n = 5, c;

long r;

TSet s = make\_set(n);

print\_set(s);

add\_max\_element(s);

cout << "Максимальный элемент добавлен в конец: ";

print\_set(s);

cout << "Ведите рубли и копейки для ключа " << endl;

cin >> r; cin >> c;

cout << "Ключ = " << r << "," << c << endl;

Money q(r, c);

s = remove\_element(s, q);

cout << "Элемент удален: ";

print\_set(s);

s = add\_average(s);

cout << "Среднее добавлено к элементам: ";

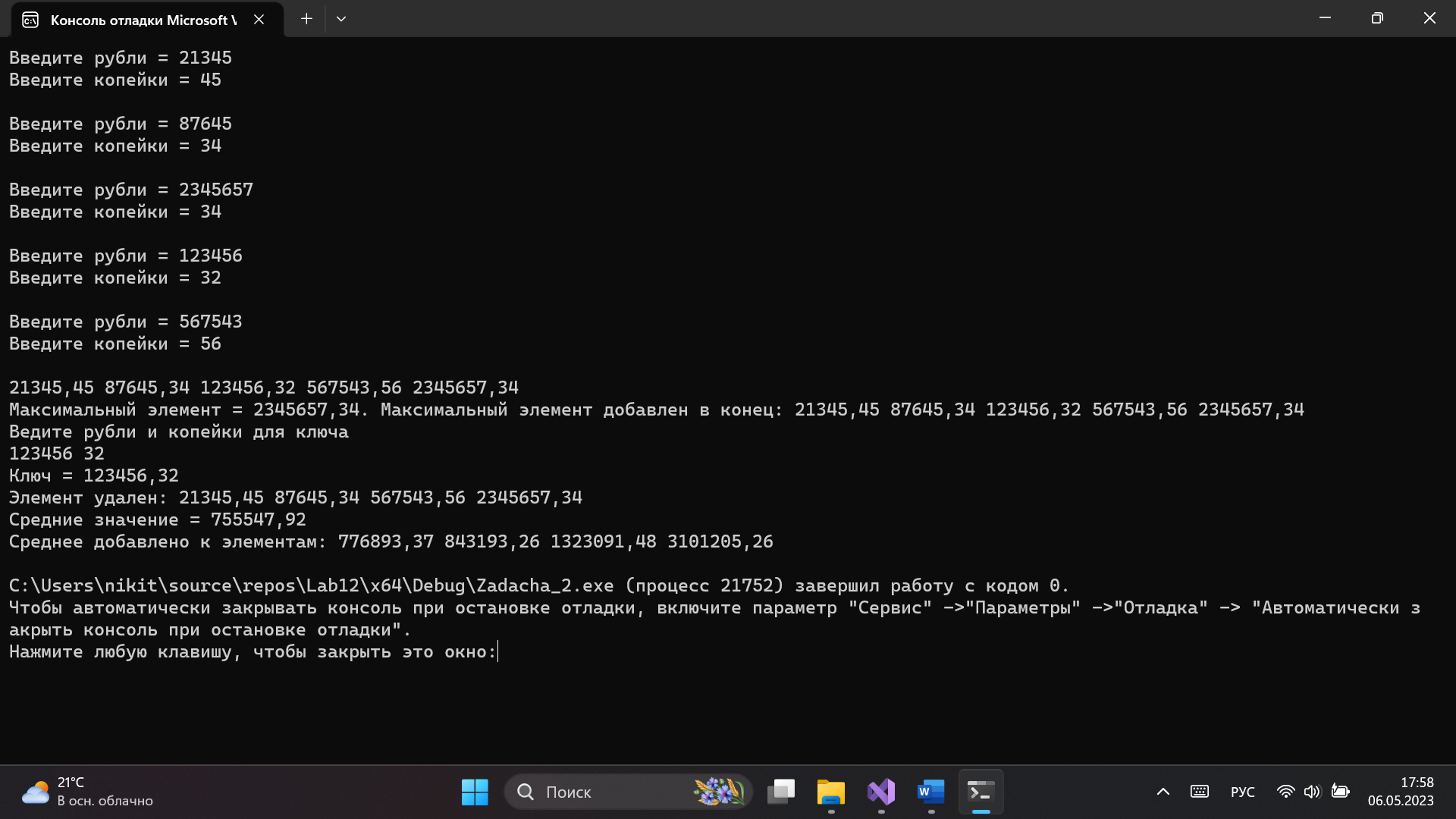
print\_set(s);

return 0;

}

**Объяснение результатов работы программы.**

В данном коде создается контейнер Set, где в качестве параметра используется пользовательский класс Money, представлены необходимые функции для создания контейнера, печати и выполнения всех 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 3.**

template <class T> //T - параметр шаблона

class Vector

{

public:

//конструктор с параметрами: выделяет память под s элементов и заполняет их

//значением k

Vector(int s, T k);

//конструктор с параметрами

Vector(const Vector<T>& a);

//деструктор

~Vector();

//оператор присваивания

Vector& operator=(const Vector<T>& a);

//операция доступа по индексу

T& operator[](int index);

//операция для добавление константы

Vector operator+(const T k);

//операция для нахождения максимального значения и добавления в конец

Vector& addMaxToEnd();

//операция удаления по ключу

void removeByKey(const T k);

//операция добавления среднего значения к каждому элементу

void addToEach();

//операция, возвращающая длину вектора

int operator()();

//перегруженные операции ввода-вывода

// <> - указывают на то, что функция является шаблоном

friend ostream& operator<< <>(ostream& out, const Vector<T>& a);

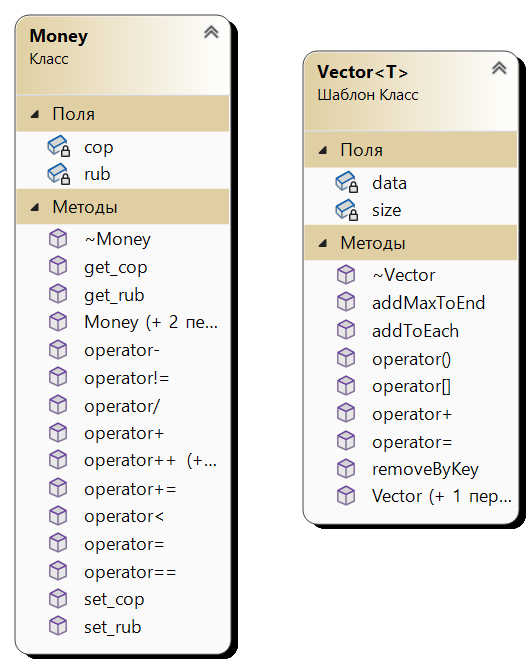
friend istream& operator>> <>(istream& in, Vector<T>& a);

private:

int size;//размер вектора

T\* data;//укзатель на динамический массив значений вектора

};



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

//конструктор с параметрами

template <class T>

Vector<T>::Vector(int s, T k)

{

size = s;

data = new T[size];

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

data[i] = k;

}

//конструктор копирования

template <class T>

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

{

size = a.size;

data = new T[size];

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

data[i] = a.data[i];

}

//деструктор

template <class T>

Vector<T>::~Vector()

{

delete[]data;

data = 0;

}

//операция присваивания

template <class T>

Vector<T>& Vector<T>::operator=(const Vector<T>& a)

{

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

size = a.size;

if (data != 0) delete[]data;

data = new T[size];

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

data[i] = a.data[i];

return \*this;

}

//операция доступа по индексу

template <class T>

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

{

if (index < size) return data[index];

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

}

//операция для добавления константы

template <class T>

Vector<T> Vector<T>::operator+(const T k)//+k

{

Vector<T> temp(size, k);//инициализируем временный вектор любым значением

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

temp.data[i] = data[i] + k;

return temp;

}

//операция для нахождения максимального значения и добавления в конец

template<class T>

Vector<T>& Vector<T>::addMaxToEnd()

{

auto maxVal = max\_element(data, data + size);

T maxValue = \*maxVal;

size++;

T\* temp = new T[size];

for (int i = 0; i < size - 1; i++)

temp[i] = data[i];

temp[size - 1] = maxValue;

delete[]data;

data = temp;

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

return \*this;

}

//операция удаления по ключу

template <class T>

void Vector<T>::removeByKey(const T k)

{

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

{

if (data[i] == k)

{

for (int j = i; j < size - 1; j++)

{

data[j] = data[j + 1];

}

size--;

break;

}

}

}

//операция добавления среднего значения к каждому элементу

template <class T>

void Vector<T>::addToEach()

{

T sum;

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

{

sum += data[i];

}

T average = sum / size;

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

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

{

data[i] += average;

}

}

//операция для получения длины вектора

template <class T>

int Vector<T>::operator ()()

{

return size;

}

//операции для ввода-вывода

template <class T>

ostream& operator<< (ostream& out, const Vector<T>& a)

{

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

out << a.data[i] << " ";

return out;

}

template <class T>

istream& operator>> (istream& in, Vector<T>& a)

{

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

in >> a.data[i];

return in;

}

**Основная программа для решения задачи 3.**

void main()

{

setlocale(LC\_ALL, "rus");

int n;

Money m;

cout << "N?"; cin >> n;

Vector<Money> v(n, m);

cin >> v;

cout << "Контейнер: " << v << endl;

v.addMaxToEnd();

cout << "Добавили в конец максимальное значение: " << v << endl;

Money k;

cout << "Введите ключ " << endl; cin >> k;

v.removeByKey(k);

cout << "Удалили элемент " << v << endl;

v.addToEach();

cout << "Добавили среднее к каждому элементу " << v;

}

**Объяснение результатов работы программы.**

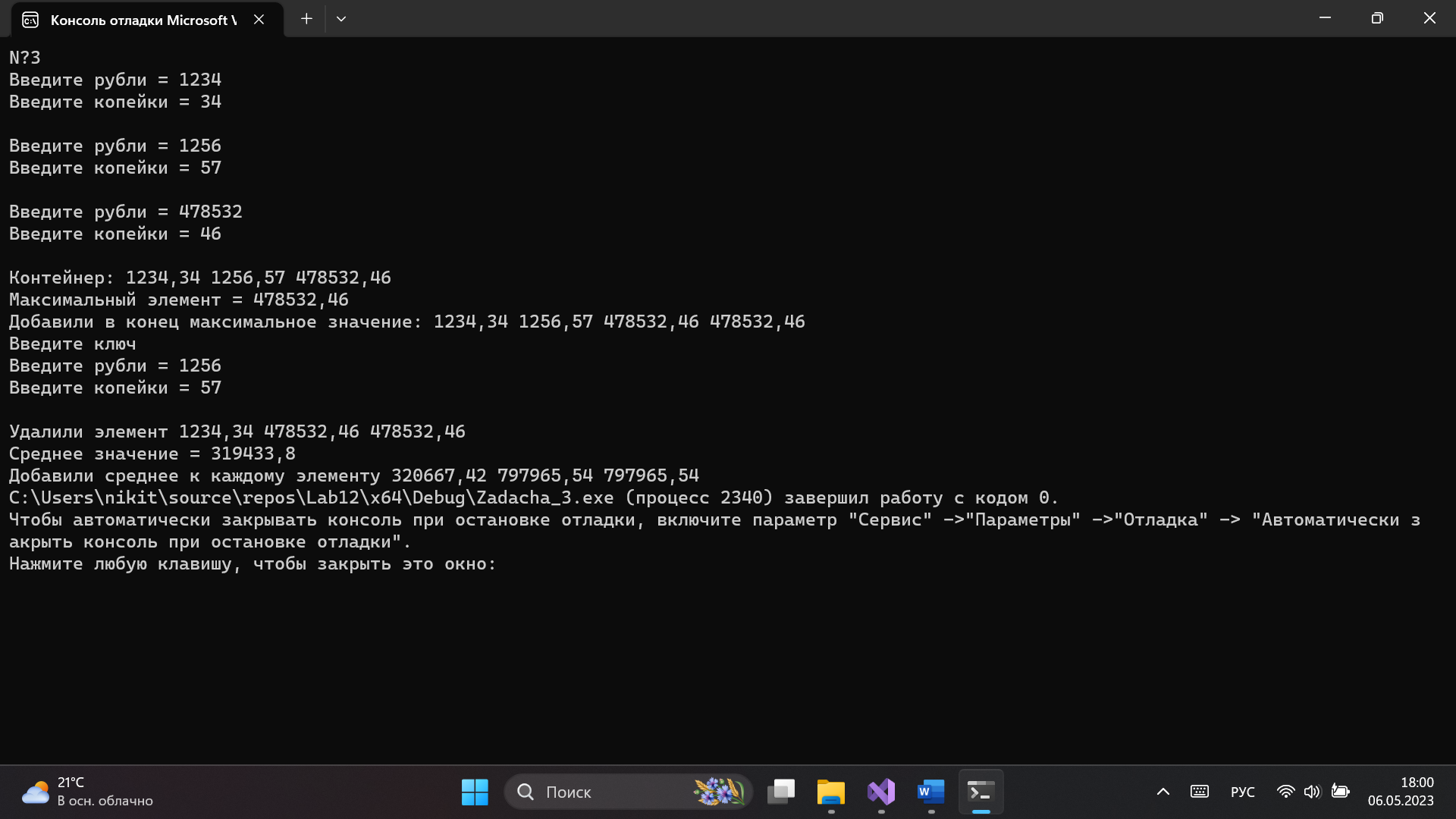
В данной программе создается параметризированный класс – вектор,

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.



**Ответы на контрольные вопросы**

**1. Что представляет собой ассоциативный контейнер?**

**2. Перечислить ассоциативные контейнеры библиотеки STL.**

**3. Каким образом можно получить доступ к элементам ассоциативного контейнера?**

**4. Привести примеры методов, используемых в ассоциативных контейнерах.**

**5. Каким образом можно создать контейнер map? Привести примеры.**

**6. Каким образом упорядочены элементы в контейнере map по умолчанию? Как изменить порядок на обратный?**

**7. Какие операции определены для контейнера map?**

**8. Написать функцию для добавления элементов в контейнер map с помощью функции make\_pair().**

**9. Написать функцию для добавления элементов в контейнер map с помощью функции операции прямого доступа [].**

**10. Написать функцию для печати контейнера map с помощью итератора.**

**11. Написать функцию для печати контейнера map с помощью функции операции прямого доступа [].**

**12. Чем отличаются контейнеры map и multimap?**

**13. Что представляет собой контейнер set?**

**14. Чем отличаются контейнеры map и set?**

**15. Каким образом можно создать контейнер set? Привести примеры.**

**16. Каким образом упорядочены элементы в контейнере set по умолчанию? Как изменить порядок на обратный?**

**17. Какие операции определены для контейнера set?**

**18. Написать функцию для добавления элементов в контейнер set.**

**19. Написать функцию для печати контейнера set.**

**20. Чем отличаются контейнеры set и multiset?**

1.Ассоциативный контейнер - это контейнер из библиотеки STL, который позволяет хранить элементы в отсортированном порядке и осуществлять доступ к элементам по ключу. Он представляет собой структуру данных, которая хранит пары ключ-значение и использует свойства ключей, такие как порядок и уникальность, для обеспечения быстрого доступа к значениям. Некоторые примеры ассоциативных контейнеров включают map, multimap, set и multiset.

2. Ассоциативные контейнеры библиотеки STL: map, multimap, set, multiset.

3. Доступ к элементам ассоциативного контейнера осуществляется с помощью ключа. Используется операция доступа к элементам или метод at(). Операция создает новый элемент, если ключ не найден, а метод at() генерирует исключение, если ключ не найден. Также доступ к элементам можно получить с помощью итераторов.

4. Некоторые методы, используемые в ассоциативных контейнерах:

- insert() - вставляет элемент в контейнер. Если элемент с таким ключом уже существует, то вставка не производится. Метод возвращает пару, состоящую из итератора на уже существующий элемент с данным ключом и флага, указывающего, была ли произведена вставка нового элемента.

- clear() - удаляет все элементы из контейнера.

- erase() - удаляет элемент из контейнера по заданному итератору или ключу.

- find() - осуществляет поиск элемента по заданному ключу. Если ключ не найден, то возвращается итератор, указывающий на конец контейнера.

- count() - возвращает количество элементов с заданным ключом.

- size() - возвращает количество элементов в контейнере.

- empty() - возвращает true, если контейнер пуст, и false в противном случае.

5. Контейнер map можно создать с помощью следующей конструкции:

#include <map>

map<тип\_ключа, тип\_значения> имя\_контейнера;

Например:

#include <map>

map<int, string> myMap;

6. Элементы в контейнере map упорядочены по ключу в порядке возрастания. Чтобы изменить порядок на обратный, можно указать третий аргумент типа компаратора в шаблоне контейнера map. Например, чтобы создать упорядоченный по убыванию контейнер map, можно использовать следующую конструкцию:

#include <map>

#include <functional>

map<int, string, std::greater<int>> myMap;

Здесь greater<int> - это компаратор, который сравнивает ключи в порядке убывания.

7. Для контейнера map определены следующие операции:

- operator[] - получение или изменение значения по ключу

- at - получение значения по ключу с проверкой наличия ключа в контейнере

- insert - вставка пары ключ-значение

- erase - удаление элемента по ключу

- clear - удаление всех элементов контейнера

- size - количество элементов в контейнере

- empty - проверка наличия элементов в контейнере

- begin, end - итераторы на начало и конец контейнера

- find - поиск элемента по ключу

- count - подсчет количества элементов с заданным ключом в контейнере.

8.

void addElement(std::map<int, std::string>& myMap, int key, std::string value) {

myMap.insert(std::make\_pair(key, value));

}

9.

void addElement(std::map<int, std::string>& myMap, int key, std::string value) {

myMap[key] = value;

}

10.

void printMap(const std::map<int, std::string>& myMap) {

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

std::cout << "Key: " << it->first << ", Value: " << it->second << '\n';

}

}

11.

void printMap(const std::map<int, std::string>& myMap) {

for (const auto& [key, value] : myMap) {

std::cout << "Key: " << key << ", Value: " << value << '\n';

}

}

12. Контейнеры map и multimap являются ассоциативными контейнерами, где элементы хранятся в отсортированном порядке по ключу. Разница между ними заключается в том, что map позволяет хранить уникальные ключи, в то время как multimap может хранить неуникальные ключи. Это означает, что multimap может хранить несколько элементов с одинаковым ключом.

13. Контейнер set является ассоциативным контейнером, который хранит уникальные элементы в отсортированном порядке. Он использует красно-черное дерево для быстрого поиска и вставки элементов. Операции поиска, вставки и удаления в set выполняются в среднем за O(log n) времени.

14. Основное различие между контейнерами map и set заключается в том, что map содержит ключ-значение, в то время как set содержит только уникальные элементы. Элементы в map сортируются по ключу, а элементы в set сортируются сами по себе. Кроме того, map позволяет быстро получить значение по ключу, а в set элементы можно только поисково получить.

15. Контейнер set можно создать с помощью шаблона класса set из стандартной библиотеки C++. Примеры:

int main() {

// Создание пустого контейнера set с типом данных int

set<int> mySet;

// Добавление элементов в контейнер set

mySet.insert(10);

mySet.insert(20);

mySet.insert(30);

// Вывод элементов контейнера set в поток вывода

for (const auto& elem : mySet) {

cout << elem << " ";

}

// Создание контейнера set с начальными значениями

set<std::string> myStringSet {"foo", "bar", "baz"};

return 0;

}

16. Элементы контейнера set по умолчанию упорядочены в возрастающем порядке. Для изменения порядка на обратный можно использовать конструктор контейнера set с указанием функции сравнения, которая будет определять порядок сортировки.

Например, чтобы отсортировать элементы в set по убыванию, можно использовать следующий код:

#include <iostream>

#include <set>

#include <functional>

int main() {

set<int, std::greater<int>> mySet {5, 2, 8, 1, 9};

for (const auto& elem : mySet) {

cout << elem << " ";

}

return 0;

}

17. Операции, определенные для контейнера set, включают в себя вставку элементов, удаление элементов, проверку наличия элемента, получение количества элементов и проход по элементам контейнера.

18. Функция для добавления элементов в контейнер set может выглядеть следующим образом:

void addElement(std::set<int>& mySet, int value) {

mySet.insert(value);

}

19. Функция для печати контейнера set может быть написана следующим образом:

void printSet(const std::set<int>& mySet) {

for (const auto& elem : mySet) {

std::cout << elem << " ";

}

}

20. Контейнеры set и multiset очень похожи, оба являются упорядоченными и не содержат дубликатов. Разница между ними заключается в том, что в set каждое значение может быть уникальным, тогда как в multiset допустимо наличие повторяющихся элементов.