Пермский национальный исследовательский политехнический университет

**Лабораторная работа №13**

**Дисциплина: Основы алгоритмизации и**

**программирования**

**Вариант № 9**

**"Стандартные обобщенные алгоритмы 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) Описание параметризированного класса для решения задачи 2.

8) Основная программа для решения задачи 2.

9) Объяснение результатов работы программы.

10) Определение методов и операций для решения задачи 3.

11) Основная программа для решения задачи 3

12) Объяснение результатов работы программы.

**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 - двунаправленная очередь

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

**Задача 2**

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

**Задача 3**

Ассоциативный контейнер - множество

|  |  |  |
| --- | --- | --- |
| **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

MDeque makeDeque(int n) {

MDeque myDeque(n);

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

myDeque[i] = Money(rand() % 100, rand() % 100); // создание экземпляра класса Money с параметрами rub и cop

}

return myDeque;

}

// Функция для вывода элементов контейнера на экран

void printDeque(const MDeque& myDeque) {

for (auto element : myDeque) {

cout << element.get\_rub() << "." << element.get\_cop() << " ";

}

cout << endl;

}

// Функция для нахождения максимального элемента и добавления его в конец контейнера

void findMaxElementAndPushBack(MDeque& myDeque) {

auto maxElIter = max\_element(myDeque.begin(), myDeque.end()); // находим итератор на максимальный элемент

myDeque.push\_back(\*maxElIter); // добавляем максимальный элемент в конец контейнера

}

// Функция для нахождения элемента с заданным ключом и удаления его из контейнера

void findAndRemoveByKey(MDeque& myDeque, Money key) {

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

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

myDeque.erase(it);

}

else {

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

}

}

// Функция добавления к каждому элементу среднего арифметического элементов контейнера

MDeque findSredneeAndPush(MDeque& myDeque)

{

Money m = Money(0, 0);

if (myDeque.empty()) {

cout << "Контейнер пуст" << endl;

}

else

{

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

int n = myDeque.size();

Money r = m / n;

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

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

myDeque[i] = r + myDeque[i];

}

return myDeque;

}

}

**Основная программа для решения задачи 1**

int main()

{

setlocale(LC\_ALL, "rus");

int n;

MDeque myDeque = makeDeque(5); // создание контейнера

cout << "Контейнер: "; printDeque(myDeque); cout << endl;

findMaxElementAndPushBack(myDeque); // находим максимальный элемент и добавляем его в конец контейнера

cout << "Добавили максимальный элемент в конец: "; printDeque(myDeque); cout << endl; // выводим элементы контейнера на экран

Money key;

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

findAndRemoveByKey(myDeque, key); // находим элемент с заданным ключом и удаляем его

cout << "Удалили элемнт: "; printDeque(myDeque); // выводим элементы контейнера на экран

findSredneeAndPush(myDeque);

cout << "Добавили к каждому элементу: "; printDeque(myDeque);

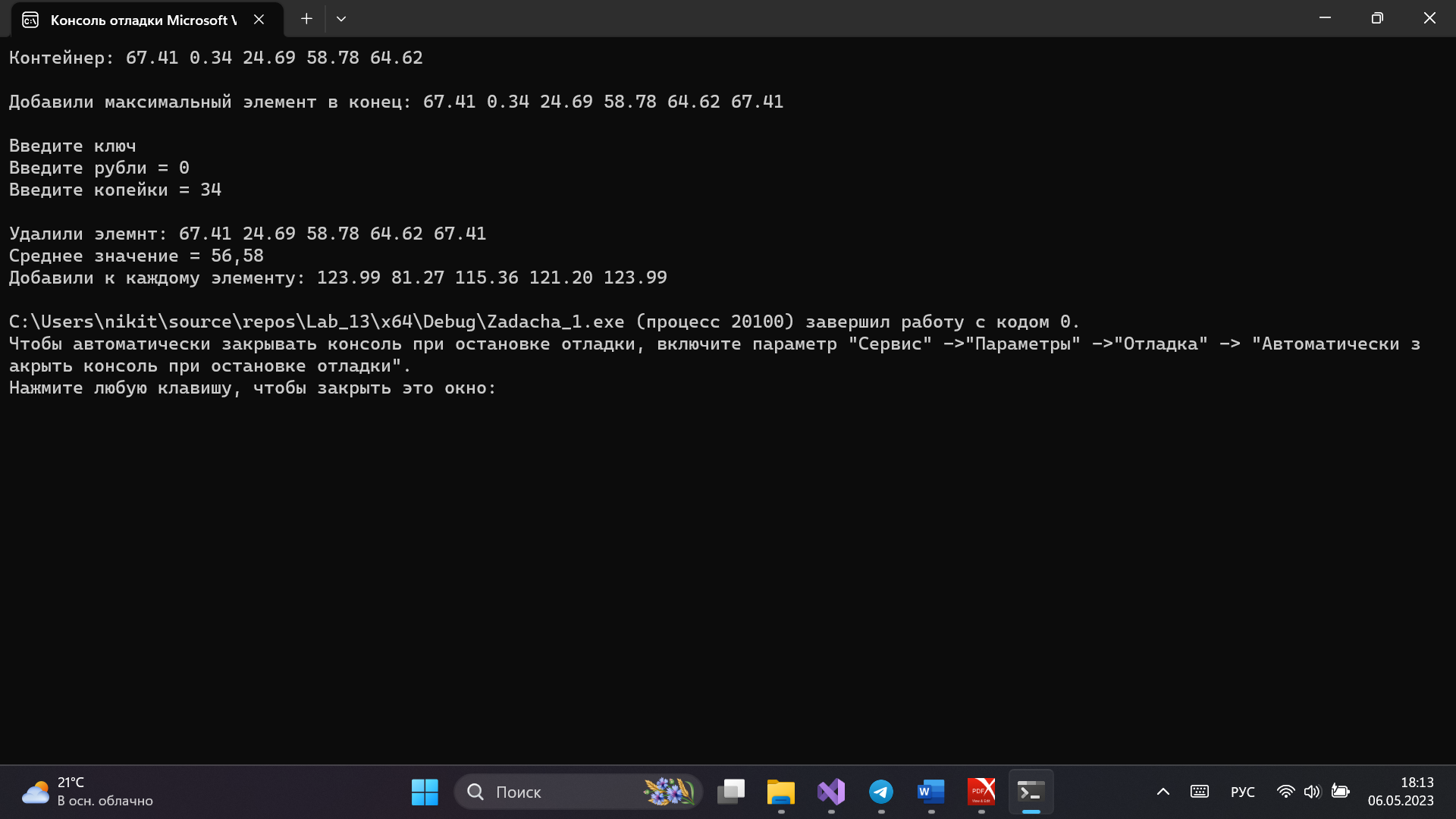
}

**Объяснение результатов работы программы.**

В данном коде создается контейнер - двунаправленная очередь с типом элементов 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 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);

}

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

//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

};

//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;

}

//Поиск среднего значения

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 2**

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

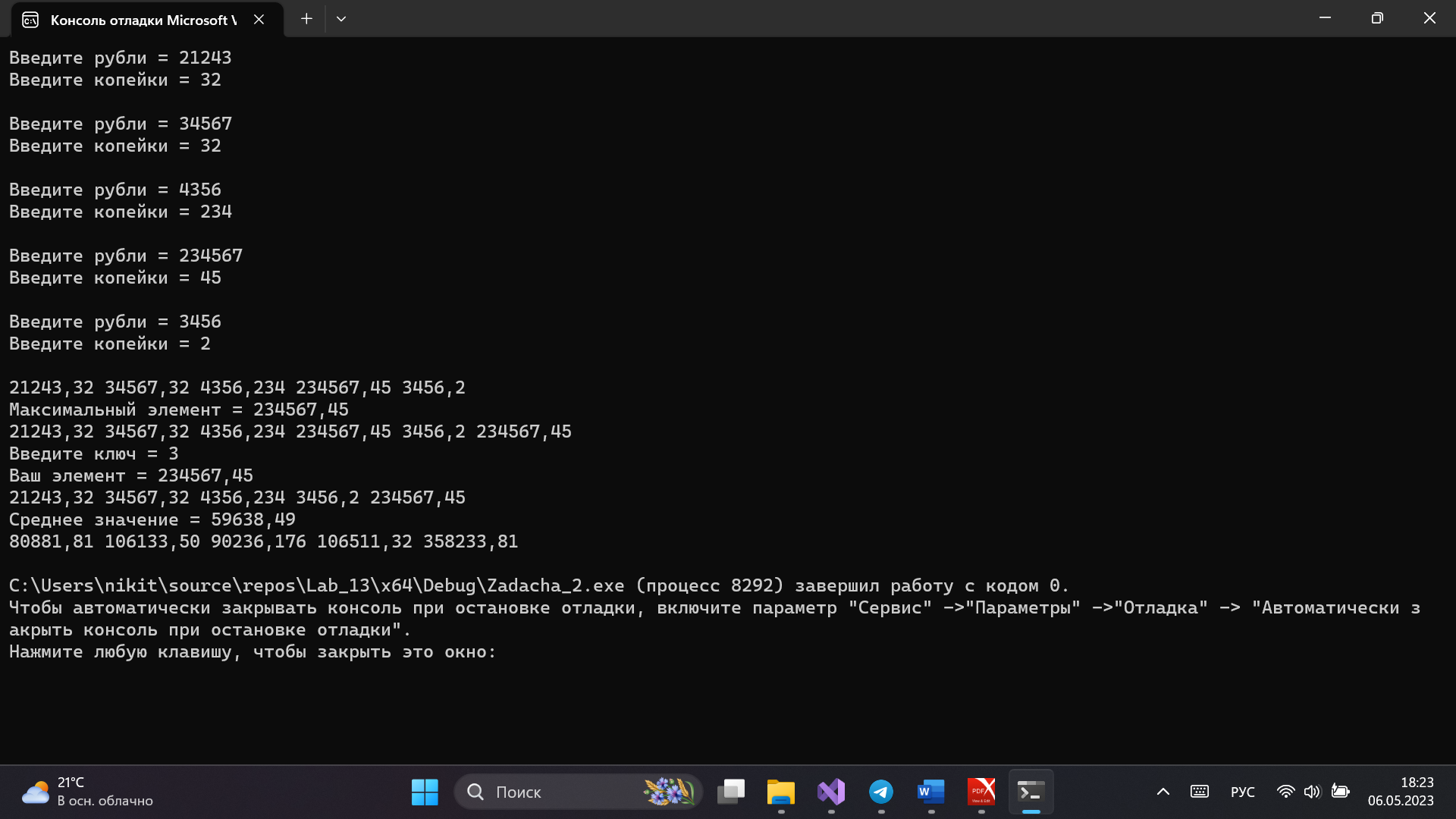
vec.Print(); //печать вектора

}

**Объяснение результатов работы программы.**

В данном коде создается параметризированный класс – Вектор, 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.

В главной функции создается экземпляр класса Vector, где в качестве параметра пользовательский класс Money, а так же вызов методов для выполнения заданий.



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

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 3.**

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 данной программе создается Ассоциативный контейнер – множество,

в качестве параметра используется пользовательский класс Money.

В программе представлены методы: печать, создание, 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.

