Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №18.13**

Дисциплина: «Информатика»

Тема: “ Объектно-ориентированное программирование.

Стандартные обобщённые алгоритмы библиотеки STL.”

Вариант 15

Выполнил:

Студент группы ИВТ-20-2Б Чувашев Максим

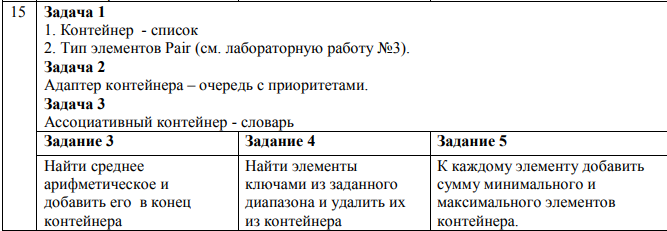
Проверила:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**



**Анализ задачи**

1. Для решения задачи необходимо:
   1. Использовать класс Pair.
   2. Использовать контейнеры библиотеки STL.
   3. Использовать алгоритмы библиотеки STL.
2. В ходе работы были использованы следующие типы данных:
   1. Pair
   2. Int
   3. Double
   4. multimap<int, Pair> m;
3. Для решения задачи данные были представлены в следующем виде:
   1. multimap<int, Pair> m;
   2. list<Pair> l;
   3. priority\_queue<Pair> q;
4. Для операций ввода и вывода использовались следующие операторы и функции:
   1. Операторы ввода и вывода cin cout
5. Для операций ввода и вывода использовались следующие операторы и функции
   1. Функция ввода ключа

void INTER(Pair \*key)

{

cout << "\n\n\n";

cout << "Введите ключ: ";

cin >> \*key;

cout << "\n\n\n";

}

void INTER(pair<int ,Pair>\* key)

{

cout << "\n\n\n";

cout << "Введите ключ: ";

cin >> key->first;

cout << "Введите значение: \n";

cin >> key->second;

cout << "\n\n\n";

}

* 1. Average

void average(list<Pair>\* l)

{

list<Pair> lcopy;

copy(l->begin(), l->end(), back\_inserter(lcopy)); // функция копирования списка

Pair AVG;

while (lcopy.empty() != true) // пока в новом списке есть элементы

{

AVG = AVG + lcopy.front();

lcopy.pop\_front();

}

AVG.set\_first(AVG.get\_first() / l->size());

AVG.set\_second(AVG.get\_second() / l->size());

l->push\_back(AVG);

}

void average(priority\_queue<Pair>\* q)

{

priority\_queue<Pair> qcopy = \*q;

Pair AVG;

while (qcopy.empty() != true) // пока в новом списке есть элементы

{

AVG = AVG + qcopy.top();

qcopy.pop();

}

AVG.set\_first(AVG.get\_first() / q->size());

AVG.set\_second(AVG.get\_second() / q->size());

cout << "\n\nСреднее арифметическое: " << AVG << "\n\n";

q->push(AVG);

}

void average(multimap<int, Pair>\* m)

{

Pair AVG;

for(const auto& elem : \*m) // проходимся по всем элементам

{

AVG = AVG + elem.second;

}

AVG.set\_first(AVG.get\_first() / m->size());

AVG.set\_second(AVG.get\_second() / m->size());

cout << "\n\nСреднее арифметическое: " << AVG << "\n\n\n";

m->emplace(rand() % 100, AVG);

}

* 1. Increase

list<Pair> increase(list<Pair>\* l)

{

list<Pair> lcopy;

copy(l->begin(), l->end(), back\_inserter(lcopy)); // функция копирования списка

Pair max;

max = -1;

Pair min;

min = 101;

for(const auto& elem : lcopy)

{

if (max < elem)

{

max = elem;

}

if (min > elem)

{

min = elem;

}

}

cout << endl << "min: " << min << endl;

cout << endl << "max: " << max << endl;

Pair res = min + max;

lcopy.clear();

for(const auto& elem : \*l)

{

Pair tmp = elem;

lcopy.push\_back(tmp + res);

}

return lcopy;

}

multimap<int, Pair> increase(multimap<int, Pair>\* m)

{

Pair max;

max = -1;

Pair min;

min = 101;

for(const auto& elem : \*m)

{

if (max < elem.second)

{

max = elem.second;

}

if (min > elem.second)

{

min = elem.second;

}

}

cout << endl << "min: " << min << endl;

cout << endl << "max: " << max << endl;

Pair res = min + max;

multimap<int, Pair> mcopy;

for(const auto& elem : \*m)

{

Pair tmp = elem.second;

mcopy.emplace(elem.first, tmp + res);

}

return mcopy;

}

priority\_queue<Pair> increase(priority\_queue<Pair>\* q)

{

priority\_queue<Pair> qcopy = \*q;

Pair max;

max = -1;

Pair min;

min = 101;

while (qcopy.empty() != true)

{

if (max < qcopy.top())

{

max = qcopy.top();

}

if (min > qcopy.top())

{

min = qcopy.top();

}

qcopy.pop();

}

cout << endl << "min: " << min << endl;

cout << endl << "max: " << max << endl;

Pair res = min + max;

while (q->empty() != true)

{

Pair tmp = q->top();

qcopy.push(tmp + res);

q->pop();

}

return qcopy;

}

* 1. Функция удаления элемента

void Del(priority\_queue<Pair> \*q)

{

cout << "Введите число которое вы хотите удалить: \t";

Pair tmp = 0;

cin >> tmp;

priority\_queue<Pair> qcopy;

int size = q->size();

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

{

if (q->top() != tmp)

{

qcopy.push(q->top());

}

q->pop();

}

size = qcopy.size();

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

{

q->push(qcopy.top());

qcopy.pop();

}

}

void Del(pair<int, Pair> key, multimap<int, Pair> \*m)

{

int choice = 0;

bool f = false;

while (f == false)

{

switch (choice)

{

case 1: m->erase(key.first); f = true; break;

case 2: for (const auto& elem : \*m)

{

if (elem.second == key.second)

{

m->erase(elem.first);

break;

}

}

f = true;

break;

default: cout << "\n\n1 : удаление по ключу\n2 : удаление по значению\n\n"; cin >> choice;

}

}

}

* 1. Функция для решения первого задания

void LIST\_TASK1()

{

cout << "Решение 1 задания:\n\n";

list<Pair> l;

double size = 0;

SIZE\_INIT(&size);

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

{

Pair tmp(rand() % 100, rand() % 10000 / 100.0);

l.push\_back(tmp);

}

Print(l);

average(&l);

Print(l);

Pair key(-1);

INTER(&key);

l.erase(remove(l.begin(), l.end(), key), l.end()); // перебрать

Print(l);

l = increase(&l);

Print(l);

l.sort(PP);

Print(l);

reverse(l.begin(), l.end());

Print(l);

Pair ptr;

cout << "\n\nКакую пару мы хотите найти: ";

cin >> ptr;

if (find(l.begin(), l.end(), ptr) != l.end())

{

cout << "\n\nЭлемент в списке найден!\n\n";

}

Print(l);

}

* 1. Функция для решения второго задания

void QUEUE\_TASK2()

{

cout << "\n\n+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-\n\n";

cout << "Решение 2 задания:\n\n";

priority\_queue<Pair> q;

double size = 0;

SIZE\_INIT(&size);

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

{

Pair tmp(rand() % 100, rand() % 10000 / 100.0);

q.push(tmp);

}

Print(q);

average(&q);

Print(q);

Del(&q);

Print(q);

q = increase(&q);

Print(q);

priority\_queue<Pair> qcopy = q;

list<Pair> listHelp;

while (qcopy.empty() != true)

{

listHelp.push\_back(qcopy.top());

qcopy.pop();

}

reverse(listHelp.begin(), listHelp.end());

Print(listHelp);

listHelp.clear();

}

* 1. Функция для решения третьего задания

void MAP\_TASK3()

{

cout << "\n\n+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-\n\n";

cout << "Решение 3 задания:\n\n";

multimap<int, Pair> m;

double size = 0;

SIZE\_INIT(&size);

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

{

Pair tmp(rand() % 100, rand() % 10000 / 100.0);

m.emplace(rand() % 100, tmp);

}

Print(m);

average(&m);

Print(m);

pair<int, Pair> key(0, 0);

INTER(&key);

Del(key, &m);

Print(m);

m = increase(&m);

Print(m);

list<pair<int,Pair>> listHelp;

for (const auto& elem : m)

{

listHelp.push\_front(elem);

}

Print(listHelp);

listHelp.clear();

cout << "\n\n\nВведите ключ для поиска: ";

int key1 = 0;

cin >> key1;

auto tmp = m.find(key1);

if (tmp != m.end())

{

cout << "\n\nЭлемент найден!\n\n";

pair<int, Pair> tmp1 = \*tmp;

cout << "\n\nKey: [" << tmp1.first << "] - ";

cout << tmp1.second << " \n\n\n";

}

else

{

cout << "\n\nЭлемента с данным ключом не существует!\n\n";

}

}

**Код**

#include <iostream>

#include <algorithm>

#include <map>

#include <list>

#include <queue>

#include <ctime>

#include "Pair.h"

using namespace std;

bool operator<(Pair a, Pair b)

{

return a.get\_first() + a.get\_second() < b.get\_first() + b.get\_second();

}

bool operator>(Pair a, Pair b)

{

return a.get\_first() + a.get\_second() > b.get\_first() + b.get\_second();

}

bool operator==(Pair a, Pair b)

{

return ((a.get\_first() == b.get\_first()) && (a.get\_second() == b.get\_second()));

}

bool operator!=(Pair a, Pair b)

{

return ((a.get\_first() != b.get\_first()) || (a.get\_second() != b.get\_second()));

}

bool PP(Pair a, Pair b)

{

return a < b;

}

void SIZE\_INIT(double\* size)

{

while (\*size < 1 || \*size > 10 || \*size != (int)\*size)

{

cout << "Введите размер: ";

cin >> \*size;

}

}

void INTER(Pair \*key)

{

cout << "\n\n\n";

cout << "Введите ключ: ";

cin >> \*key;

cout << "\n\n\n";

}

void INTER(pair<int ,Pair>\* key)

{

cout << "\n\n\n";

cout << "Введите ключ: ";

cin >> key->first;

cout << "Введите значение: \n";

cin >> key->second;

cout << "\n\n\n";

}

template<typename T>

void Print(list<T> l)

{

cout << endl << "=========================================================================================\n\n";

while (l.empty() != true)

{

cout << l.front() << " \t";

l.pop\_front();

}

cout << endl << "=========================================================================================\n\n\n";

}

void Print(list<pair<int, Pair>> l)

{

cout << endl << "=========================================================================================\n\n";

for (const auto& elem : l)

{

cout << "Key: [" << elem.first << "] - ";

cout << elem.second << " \t";

}

cout << endl << "=========================================================================================\n\n\n";

}

void Print(priority\_queue<Pair> q)

{

cout << endl << "=========================================================================================\n\n";

while (q.empty() != true)

{

cout << q.top() << " \t";

q.pop();

}

cout << endl << "=========================================================================================\n\n\n";

}

void Print(multimap<int, Pair> m)

{

cout << endl << "=========================================================================================\n\n";

for(const auto& elem : m)

{

cout << "Key: [" << elem.first << "] - ";

cout << elem.second << " \t";

}

cout << endl << "=========================================================================================\n\n\n";

}

void average(list<Pair>\* l)

{

list<Pair> lcopy;

copy(l->begin(), l->end(), back\_inserter(lcopy)); // функция копирования списка

Pair AVG;

while (lcopy.empty() != true) // пока в новом списке есть элементы

{

AVG = AVG + lcopy.front();

lcopy.pop\_front();

}

AVG.set\_first(AVG.get\_first() / l->size());

AVG.set\_second(AVG.get\_second() / l->size());

l->push\_back(AVG);

}

void average(priority\_queue<Pair>\* q)

{

priority\_queue<Pair> qcopy = \*q;

Pair AVG;

while (qcopy.empty() != true) // пока в новом списке есть элементы

{

AVG = AVG + qcopy.top();

qcopy.pop();

}

AVG.set\_first(AVG.get\_first() / q->size());

AVG.set\_second(AVG.get\_second() / q->size());

cout << "\n\nСреднее арифметическое: " << AVG << "\n\n";

q->push(AVG);

}

void average(multimap<int, Pair>\* m)

{

Pair AVG;

for(const auto& elem : \*m) // проходимся по всем элементам

{

AVG = AVG + elem.second;

}

AVG.set\_first(AVG.get\_first() / m->size());

AVG.set\_second(AVG.get\_second() / m->size());

cout << "\n\nСреднее арифметическое: " << AVG << "\n\n\n";

m->emplace(rand() % 100, AVG);

}

list<Pair> increase(list<Pair>\* l)

{

list<Pair> lcopy;

copy(l->begin(), l->end(), back\_inserter(lcopy)); // функция копирования списка

Pair max;

max = -1;

Pair min;

min = 101;

for(const auto& elem : lcopy)

{

if (max < elem)

{

max = elem;

}

if (min > elem)

{

min = elem;

}

}

cout << endl << "min: " << min << endl;

cout << endl << "max: " << max << endl;

Pair res = min + max;

lcopy.clear();

for(const auto& elem : \*l)

{

Pair tmp = elem;

lcopy.push\_back(tmp + res);

}

return lcopy;

}

multimap<int, Pair> increase(multimap<int, Pair>\* m)

{

Pair max;

max = -1;

Pair min;

min = 101;

for(const auto& elem : \*m)

{

if (max < elem.second)

{

max = elem.second;

}

if (min > elem.second)

{

min = elem.second;

}

}

cout << endl << "min: " << min << endl;

cout << endl << "max: " << max << endl;

Pair res = min + max;

multimap<int, Pair> mcopy;

for(const auto& elem : \*m)

{

Pair tmp = elem.second;

mcopy.emplace(elem.first, tmp + res);

}

return mcopy;

}

priority\_queue<Pair> increase(priority\_queue<Pair>\* q)

{

priority\_queue<Pair> qcopy = \*q;

Pair max;

max = -1;

Pair min;

min = 101;

while (qcopy.empty() != true)

{

if (max < qcopy.top())

{

max = qcopy.top();

}

if (min > qcopy.top())

{

min = qcopy.top();

}

qcopy.pop();

}

cout << endl << "min: " << min << endl;

cout << endl << "max: " << max << endl;

Pair res = min + max;

while (q->empty() != true)

{

Pair tmp = q->top();

qcopy.push(tmp + res);

q->pop();

}

return qcopy;

}

void Del(priority\_queue<Pair> \*q)

{

cout << "Введите число которое вы хотите удалить: \t";

Pair tmp = 0;

cin >> tmp;

priority\_queue<Pair> qcopy;

int size = q->size();

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

{

if (q->top() != tmp)

{

qcopy.push(q->top());

}

q->pop();

}

size = qcopy.size();

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

{

q->push(qcopy.top());

qcopy.pop();

}

}

void Del(pair<int, Pair> key, multimap<int, Pair> \*m)

{

int choice = 0;

bool f = false;

while (f == false)

{

switch (choice)

{

case 1: m->erase(key.first); f = true; break;

case 2: for (const auto& elem : \*m)

{

if (elem.second == key.second)

{

m->erase(elem.first);

break;

}

}

f = true;

break;

default: cout << "\n\n1 : удаление по ключу\n2 : удаление по значению\n\n"; cin >> choice;

}

}

}

void LIST\_TASK1()

{

cout << "Решение 1 задания:\n\n";

list<Pair> l;

double size = 0;

SIZE\_INIT(&size);

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

{

Pair tmp(rand() % 100, rand() % 10000 / 100.0);

l.push\_back(tmp);

}

Print(l);

average(&l);

Print(l);

Pair key(-1);

INTER(&key);

l.erase(remove(l.begin(), l.end(), key), l.end()); // перебрать

Print(l);

l = increase(&l);

Print(l);

l.sort(PP);

Print(l);

reverse(l.begin(), l.end());

Print(l);

Pair ptr;

cout << "\n\nКакую пару мы хотите найти: ";

cin >> ptr;

if (find(l.begin(), l.end(), ptr) != l.end())

{

cout << "\n\nЭлемент в списке найден!\n\n";

}

Print(l);

}

void QUEUE\_TASK2()

{

cout << "\n\n+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-\n\n";

cout << "Решение 2 задания:\n\n";

priority\_queue<Pair> q;

double size = 0;

SIZE\_INIT(&size);

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

{

Pair tmp(rand() % 100, rand() % 10000 / 100.0);

q.push(tmp);

}

Print(q);

average(&q);

Print(q);

Del(&q);

Print(q);

q = increase(&q);

Print(q);

priority\_queue<Pair> qcopy = q;

list<Pair> listHelp;

while (qcopy.empty() != true)

{

listHelp.push\_back(qcopy.top());

qcopy.pop();

}

reverse(listHelp.begin(), listHelp.end());

Print(listHelp);

listHelp.clear();

}

void MAP\_TASK3()

{

cout << "\n\n+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-\n\n";

cout << "Решение 3 задания:\n\n";

multimap<int, Pair> m;

double size = 0;

SIZE\_INIT(&size);

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

{

Pair tmp(rand() % 100, rand() % 10000 / 100.0);

m.emplace(rand() % 100, tmp);

}

Print(m);

average(&m);

Print(m);

pair<int, Pair> key(0, 0);

INTER(&key);

Del(key, &m);

Print(m);

m = increase(&m);

Print(m);

list<pair<int,Pair>> listHelp;

for (const auto& elem : m)

{

listHelp.push\_front(elem);

}

Print(listHelp);

listHelp.clear();

cout << "\n\n\nВведите ключ для поиска: ";

int key1 = 0;

cin >> key1;

auto tmp = m.find(key1);

if (tmp != m.end())

{

cout << "\n\nЭлемент найден!\n\n";

pair<int, Pair> tmp1 = \*tmp;

cout << "\n\nKey: [" << tmp1.first << "] - ";

cout << tmp1.second << " \n\n\n";

}

else

{

cout << "\n\nЭлемента с данным ключом не существует!\n\n";

}

}

int main()

{

srand(time(NULL));

setlocale(LC\_ALL, "Ru");

LIST\_TASK1();

QUEUE\_TASK2();

MAP\_TASK3();

}

#include "Pair.h"

#include <iostream>

using namespace std;

//перегрузка операции присваивания

Pair& Pair::operator=(const Pair& other)

{

if (&other == this) return \*this; //проверка на самоприсваивание

this->first = other.first;

this->second = other.second;

return \*this;

}

Pair Pair::operator=(int tmp)

{

this->first = tmp;

this->second = tmp;

return \*this;

}

Pair Pair::operator-(const Pair& other)

{

Pair tmp;

tmp.first = this->first - other.first;

tmp.second = this->second - other.second;

return tmp;

}

Pair Pair::operator+(double constant)

{

if (constant == int(constant))

{

this->first = this->first + constant;

}

else

{

this->second = this->second + constant;

}

return \*this;

}

Pair Pair::operator+(const Pair& other)

{

this->first += other.first;

this->second += other.second;

return \*this;

}

Pair Pair::operator /(int size)

{

this->first = this->first / size;

this->second = this->second / size;

return \*this;

}

//перегрузка глобальной функции-операции ввода

istream& operator>>(istream& in, Pair& other)

{

cout << "Первое число: ";

in >> other.first;

cout << "Второе число: ";

in >> other.second;

return in;

}

//перегрузка глобальной функции-операции вывода

ostream& operator<<(ostream& out, const Pair& other)

{

return (out << other.first << " : " << other.second);

}

#pragma once

#include <iostream>

using namespace std;

class Pair

{

private:

int first;

double second;

public:

// конструктор без параметров

Pair()

{

first = 0; // При создании объекта поле заполняется 0

second = 0; // При создании объекта поле заполняется 0

};

bool Func(Pair other)

{

return ((this->first == other.first) && (this->second == other.second));

}

Pair(int fs)

{

first = fs;

second = fs;

}

// конструктор с параметрами

Pair(int f, double s)

{

first = f; // При создании объекта поле заполняется переданным в качестве параметра числом

second = s; // При создании объекта поле заполняется переданынм в качестве параметра числом

}

// конструктор копирования

Pair(const Pair& other)

{

this->first = other.first;

this->second = other.second;

}

// деструктор

~Pair()

{

};

int get\_first()

{

return first;

}

int get\_second()

{

return second;

}

void set\_first(int m)

{

first = m;

}

void set\_second(int s)

{

second = s;

}

//перегруженные операции

Pair& operator=(const Pair&);

Pair operator=(int);

Pair operator -(const Pair&); // перегрузка оператора "-", вычитание полей объектов

Pair operator / (int size);

Pair operator +(double constant); // перегрузка оператора "+" прибавление константы дабл к полям объекта класса

Pair operator +(const Pair& constant);

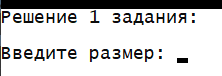
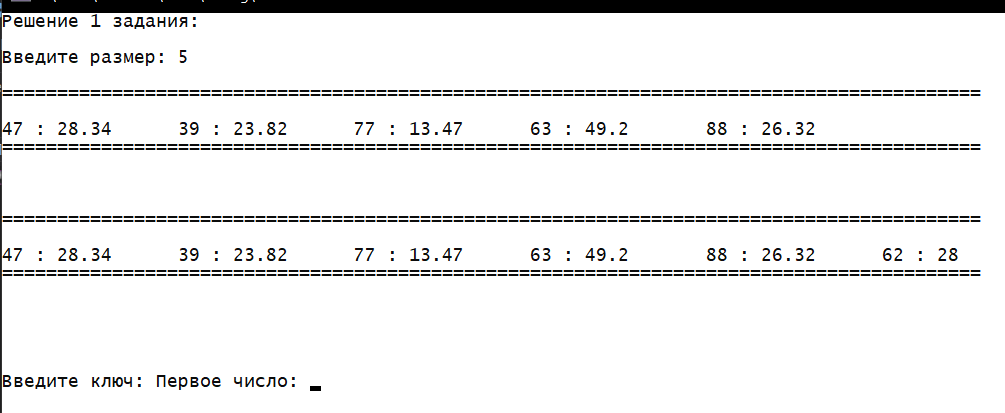
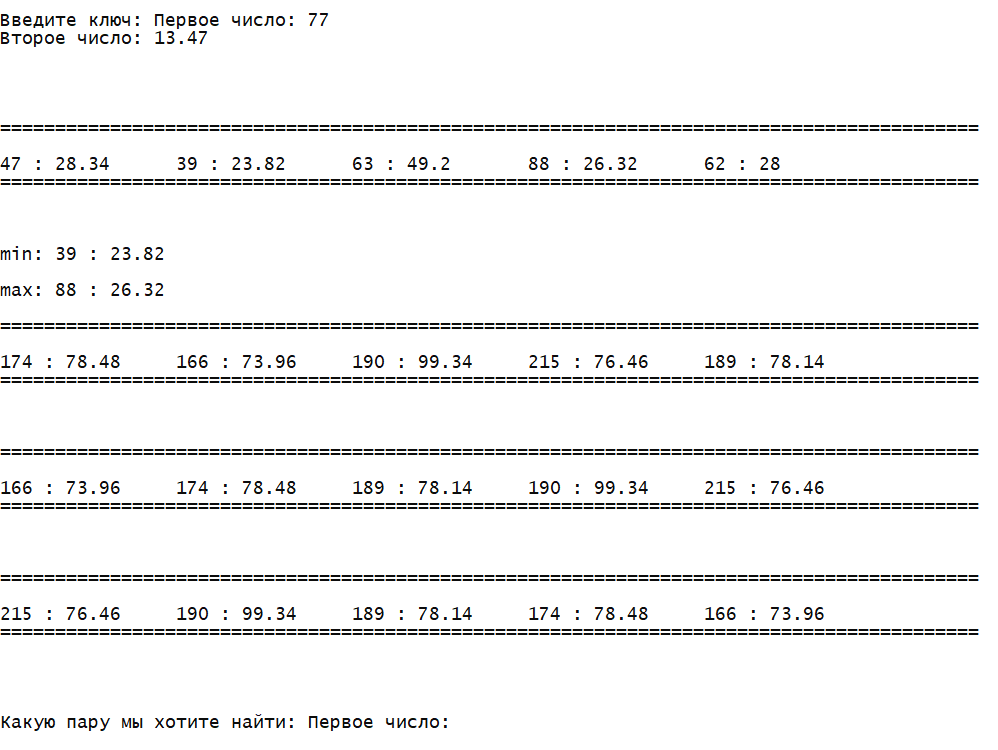
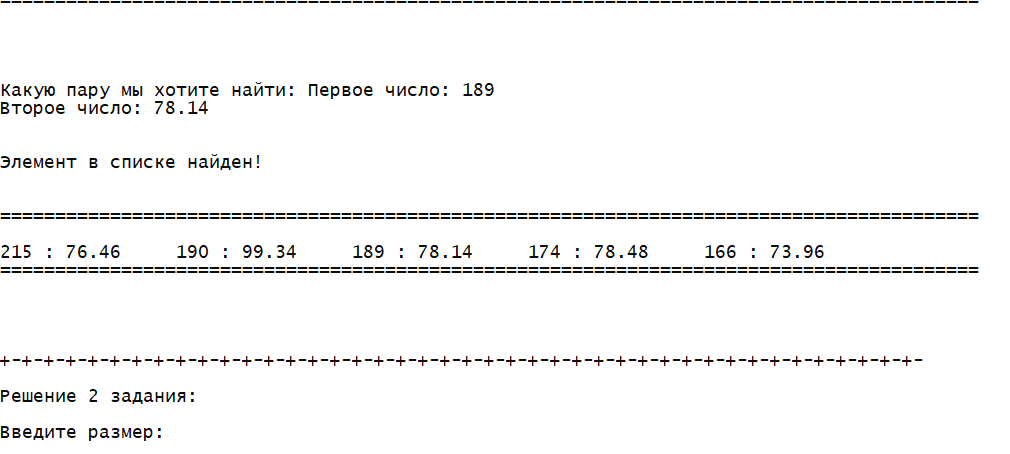
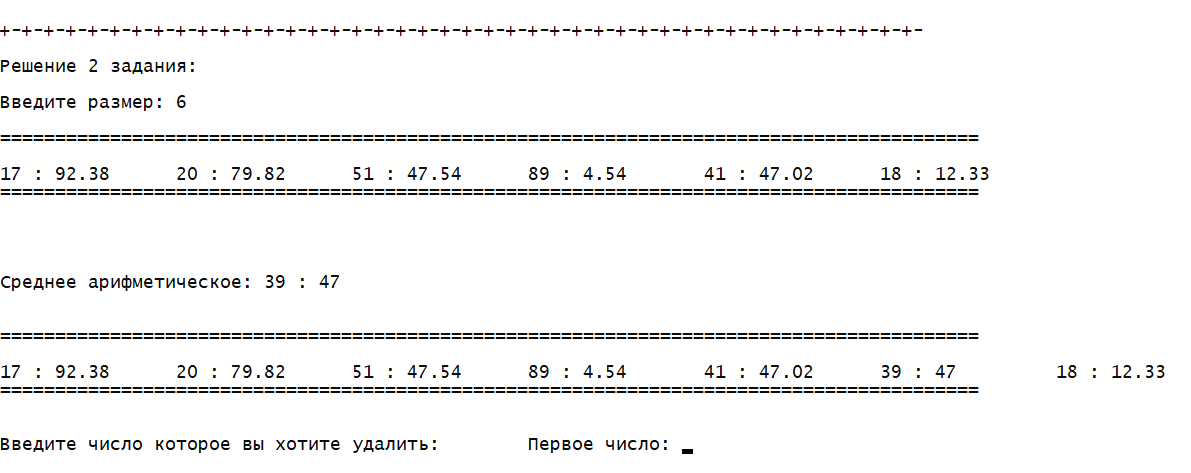
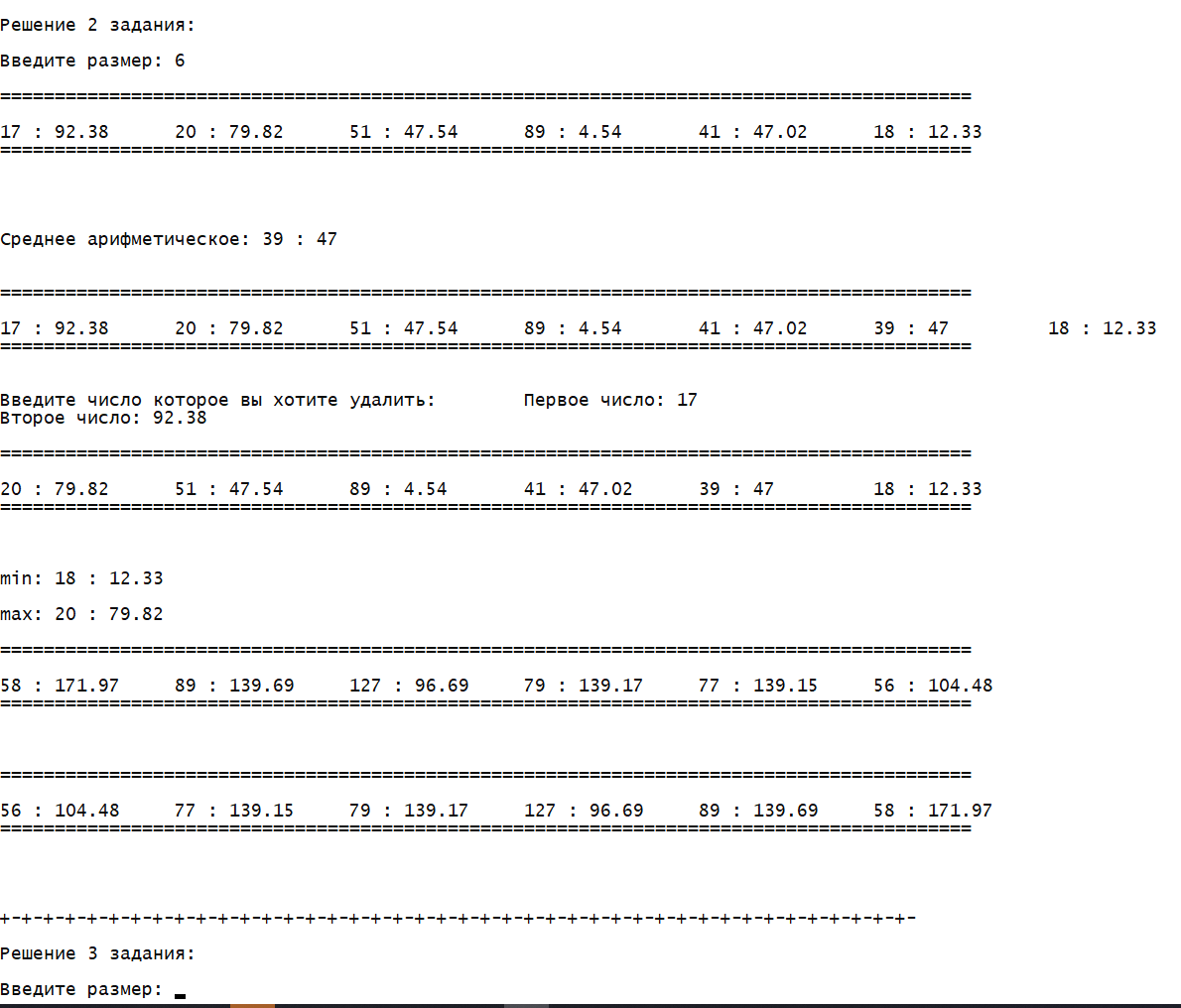
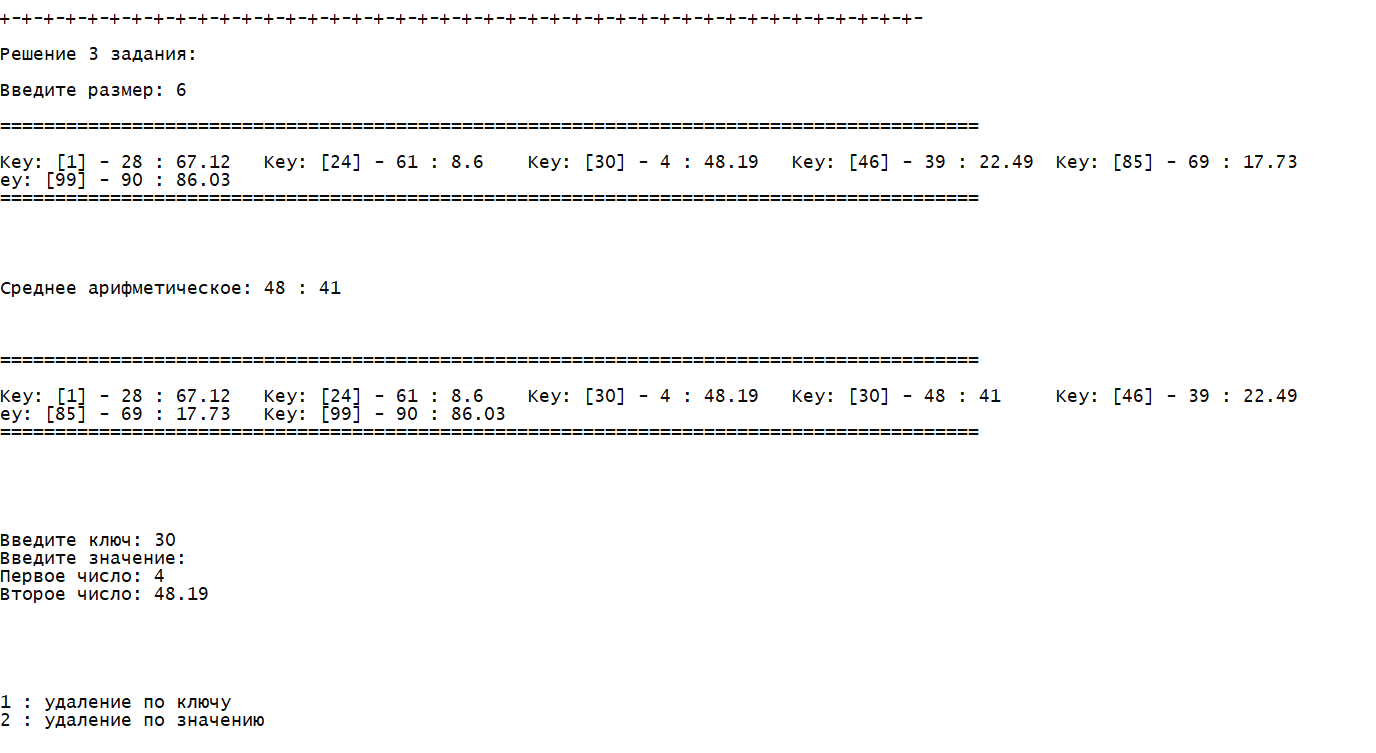
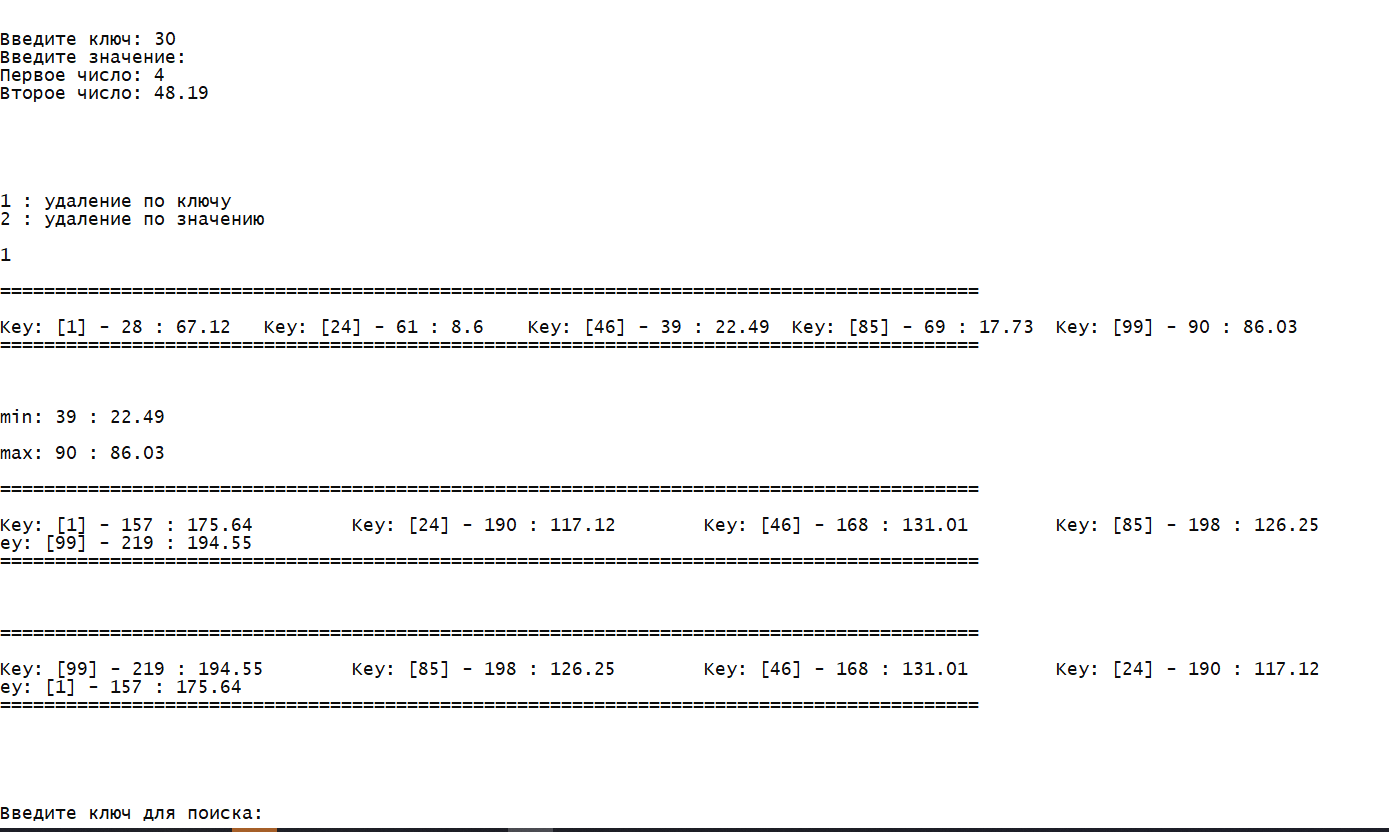
//глобальные функции ввода-вывода

friend istream& operator>>(istream& in, Pair& other); // объявление дружественного оператора ввода

friend ostream& operator<<(ostream& out, const Pair& other); // объявление дружественного оператора вывода

};

**Скриншоты**

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 