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 №18.12**

Дисциплина: «Информатика»

Тема: “ Объектно-ориентированное программирование.

Ассоциативные контейнеры библиотеки STL.”

Вариант 15

Выполнил:

Студент группы ИВТ-20-2Б Чувашев Максим

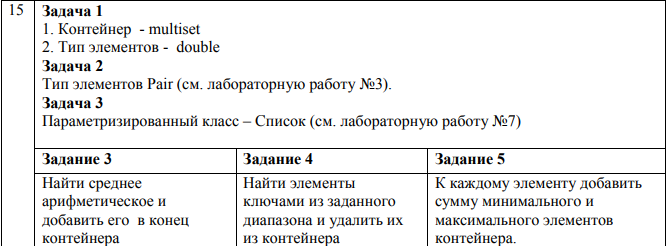
Проверила:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**



**Анализ задачи**

1. Для решения задачи необходимо:
   1. Использовать класс Pair.
   2. Использовать ассоциативные контейнеры библиотеки STL.
   3. Реализовать класс LIST
2. В ходе работы были использованы следующие типы данных:
   1. Pair
   2. Double
   3. Int
3. Для решения задачи данные были представлены в следующем виде:
   1. multiset<Pair> m;
   2. multiset<double> m;
4. Для операций ввода и вывода использовались следующие операторы и функции:
   1. Cout и cin
   2. Перегрузка оператора << для класса LIST
5. Поставленные задачи будут решены следующими действиям:
   1. Функция для подсчета среднего арифметического

template<typename T>

void AVG(multiset<T> \*m)

{

T avg(0);

for (const auto& elem : \*m)

{

avg = avg + elem;

}

avg = avg / m->size();

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

m->insert(avg);

}

* 1. Функция удаления элемента из стека

template<typename T>

void Del(multiset<T>\* m)

{

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

T tmp = 0;

cin >> tmp;

m->erase(tmp);

}

* 1. Функция добавления суммы максимального и минимального элемента

template<typename T>

void ADD\_SUM(multiset<T>\* m)

{

T min = 101;

T max = -1;

for (const auto& elem : \*m)

{

if (elem > max)

{

max = elem;

}

if (elem < min)

{

min = elem;

}

}

cout << "Минимальный элемент: " << min << "\n\n";

cout << "Максимальный элемент: " << max << "\n\n";

multiset<T> mcopy = \*m;

m->clear();

for (const auto& elem : mcopy)

{

T tmp = elem;

m->insert(tmp + max + min);

}

mcopy.clear();

}

* 1. Функция для выполнения первого задания

void MULTISET\_INIT()

{

cout << "Первое задание " << endl;

multiset<double> m;

double size = 0;

SIZE\_INIT(&size);

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

{

m.insert(rand() % 10000 / 100.0);

}

Print(m);

AVG(&m);

Print(m);

Del(&m);

Print(m);

ADD\_SUM(&m);

Print(m);

}

* 1. Функция для решения второго задания

void MULTISET\_PAIR()

{

cout << "Второе задание " << endl;

multiset<Pair> m;

double size = 0;

SIZE\_INIT(&size);

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

{

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

m.insert(tmp);

}

Print(m);

AVG(&m);

Print(m);

Del(&m);

Print(m);

ADD\_SUM(&m);

Print(m);

}

* 1. Функция для решения третьего задания

void LIST\_INIT()

{

cout << "\n\nТретье задание\n\n";

double size = 0;

SIZE\_INIT(&size);

LIST<double> l(size);

cout << l;

l.AVG();

cout << l;

l.Del();

cout << l;

l.PLUS\_MIN\_MAX();

cout << l;

}

**Код**

#include <algorithm>

#include <iostream>

#include <iterator>

#include <set>

#include "Pair.h"

#include <time.h>

#include "LIST.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();

}

void SIZE\_INIT(double\* size)

{

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

{

cout << "Введите размер: ";

cin >> \*size;

}

}

template<typename T>

void Print(multiset<T> m)

{

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

for (const auto &element : m)

{

cout << element << " \t";

}

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

}

template<typename T>

void AVG(multiset<T> \*m)

{

T avg(0);

for (const auto& elem : \*m)

{

avg = avg + elem;

}

avg = avg / m->size();

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

m->insert(avg);

}

template<typename T>

void Del(multiset<T>\* m)

{

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

T tmp = 0;

cin >> tmp;

m->erase(tmp);

}

template<typename T>

void ADD\_SUM(multiset<T>\* m)

{

T min = 101;

T max = -1;

for (const auto& elem : \*m)

{

if (elem > max)

{

max = elem;

}

if (elem < min)

{

min = elem;

}

}

cout << "Минимальный элемент: " << min << "\n\n";

cout << "Максимальный элемент: " << max << "\n\n";

multiset<T> mcopy = \*m;

m->clear();

for (const auto& elem : mcopy)

{

T tmp = elem;

m->insert(tmp + max + min);

}

mcopy.clear();

}

void MULTISET\_INIT()

{

cout << "Первое задание " << endl;

multiset<double> m;

double size = 0;

SIZE\_INIT(&size);

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

{

m.insert(rand() % 10000 / 100.0);

}

Print(m);

AVG(&m);

Print(m);

Del(&m);

Print(m);

ADD\_SUM(&m);

Print(m);

}

void MULTISET\_PAIR()

{

cout << "Второе задание " << endl;

multiset<Pair> m;

double size = 0;

SIZE\_INIT(&size);

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

{

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

m.insert(tmp);

}

Print(m);

AVG(&m);

Print(m);

Del(&m);

Print(m);

ADD\_SUM(&m);

Print(m);

}

void LIST\_INIT()

{

cout << "\n\nТретье задание\n\n";

double size = 0;

SIZE\_INIT(&size);

LIST<double> l(size);

cout << l;

l.AVG();

cout << l;

l.Del();

cout << l;

l.PLUS\_MIN\_MAX();

cout << l;

}

int main()

{

setlocale(LC\_ALL, "Ru");

MULTISET\_INIT();

MULTISET\_PAIR();

LIST\_INIT();

}

#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;

}

bool Pair::operator>(const Pair& other)

{

return this->first + this->second > other.first + other.second;

}

bool Pair::operator <(const Pair& other)

{

return this->first + this->second<other.first + other.second;

}

//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 <string>

#include <iostream>

#include "Pair.h"

using namespace std;

template <class T>

class LIST

{

public:

LIST()

{

Size = 0;

data = nullptr;

}

LIST(int Size) // Конструктор с параметрами

{

this->Size = Size;

data = new T[Size];

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

{

data[i] = rand() % 10000 / 100.0;

}

}

LIST(const LIST& other) // Конструктор с параметрами

{

Size = other.Size;

if (data != nullptr)

{

delete[] data;

}

data = new T[Size];

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

{

data[i] = other.data[i];

}

}

~LIST() // деструктор

{

if (this->data != nullptr)

{

delete[] this->data;

}

data = nullptr;

}

LIST& operator = (const LIST& a)// оператор присваивания

{

if (this == &a) // Првоверка на самоприсваивание

{

return \*this;

}

Size = a.Size;

if (data != 0) // Если ячейка не пуста, то очищаем ячейку

{

delete[] data;

}

data = new int[Size];

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

{

data[i] = a.Size;

}

return \*this;

}

T& operator [] (int index) // операция доступа к индексу

{

if (index < Size)

{

return data[index];

}

else

{

cout << "\nError! Index>Size";

}

}

LIST<double>& AVG()

{

double\* res = new double[Size];

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

{

res[i] = data[i];

}

delete[] data;

data = new double[Size + 1];

double avg = 0;

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

{

data[i] = res[i];

avg += res[i];

}

avg = avg / this->Size;

cout << "\nСреднее арифметическое: \t" << avg << "\n\n";

data[Size] = avg;

Size++;

return \*this;

}

LIST<double>& Del()

{

double\* tmp = new double[Size];

int count = 0;

int countf = 0;

double N;

cout << "\n\nВведите число которое вы хотите удалить: ";

cin >> N;

cout << "\n\n";

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

{

if (data[i] != N)

{

tmp[count] = data[i];

count++;

}

else

{

countf++;

}

}

delete[] data;

data = new double[Size - countf];

Size = Size - countf;

count = 0;

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

{

data[i] = tmp[i];

}

delete[] tmp;

return \*this;

}

LIST<double>& PLUS\_MIN\_MAX()

{

double min = 101;

double max = -1;

double\* tmp = new double[Size];

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

{

if (min > data[i])

{

min = data[i];

}

if (max < data[i])

{

max = data[i];

}

tmp[i] = data[i];

}

delete[] data;

data = new double[Size];

cout << "\n\nМинимальное число: \t" << min << "\n\n";

cout << "Максимальное число: \t" << max << "\n\n";

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

{

data[i] = tmp[i] + min + max;

}

delete[] tmp;

return \*this;

}

LIST<double>& LR()

{

cout << "Введите диапазон\n";

int left = -1;

while (left < 0 || left > 15)

{

cout << "Левая граница: ";

cin >> left;

}

int right = -1;

while (right < left || right > this->Size)

{

cout << "Правая граница граница: ";

cin >> right;

}

double\* tmp = new double[Size - (right - left + 1)];

int count = 0;

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

{

if (i < left || i > right)

{

tmp[count] = data[i];

count++;

}

}

Size = Size - (right - left + 1);

delete[] data;

data = new double[Size];

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

{

data[i] = tmp[i];

}

delete[] tmp;

return \*this;

}

LIST<T>& operator \* (LIST& other)

{

LIST\* res = new LIST;

if (this->Size > other.Size)

{

int Size1 = this->Size;

res->Size = Size1;

res->data = new T[Size1];

Size1 = other.Size;

int j = 0;

for (int i = 0; i < Size1; i++)

{

res->data[i] = this->data[i] \* other.data[i];

j = i;

}

Size1 = this->Size;

j++;

for (j; j < Size1; j++)

{

res->data[j] = 0; // Заполняем разничу 1 и 2 листа нулями, так как число умножить не на что

}

return \*res;

}

else

{

int Size1 = other.Size;

res->Size = Size1;

res->data = new T[Size1];

Size1 = this->Size;

int j = 0;

for (int i = 0; i < Size1; i++)

{

res->data[i] = this->data[i] \* other.data[i];

j = i;

}

Size1 = other.Size;

j++;

for (j; j < Size1; j++)

{

res->data[j] = 0;

}

return \*res;

}

}

int operator ()()

{

return this->Size;

}

bool operator <(const LIST& other)

{

double res1 = 0;

double res2 = 0;

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

{

res1 += this->data[i];

}

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

{

res2 += other.data[i];

}

if (res1 < res2)

{

return true;

}

else

{

return false;

}

}

void SetSize(int size) // Сеттер размера вектора

{

this->Size = size;

}

int GetData() // Геттер данных ячейчки

{

return \*this->data;

}

int GetSize() //Геттер размера вектора

{

this->Size = size;

}

int size()

{

return Size;

}

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

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

//friend ostream& operator<< (ostream& out, const LIST<Pair>& other);

private:

int Size = 0;

T\* data;

};

template <typename T>

ostream& operator<< <>(ostream& out, const LIST<T>& other)

{

out << endl << "\n\n=========================================================================================\n\n";

for (int i = 0; i < other.Size; i++)

{

out << other.data[i] << " \t";

}

out << endl << "\n=========================================================================================\n\n\n";

return out;

}

template<typename T>

istream& operator>> <>(istream& in, LIST<T>& other)

{

for (int i = 0; i < other.Size; i++)

{

cout << "Введите " << i << " элемент списка: ";

in >> other.data[i];

}

return in;

}

#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

};

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);

bool operator >(const Pair& other);

bool operator <(const Pair& other);

//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

};

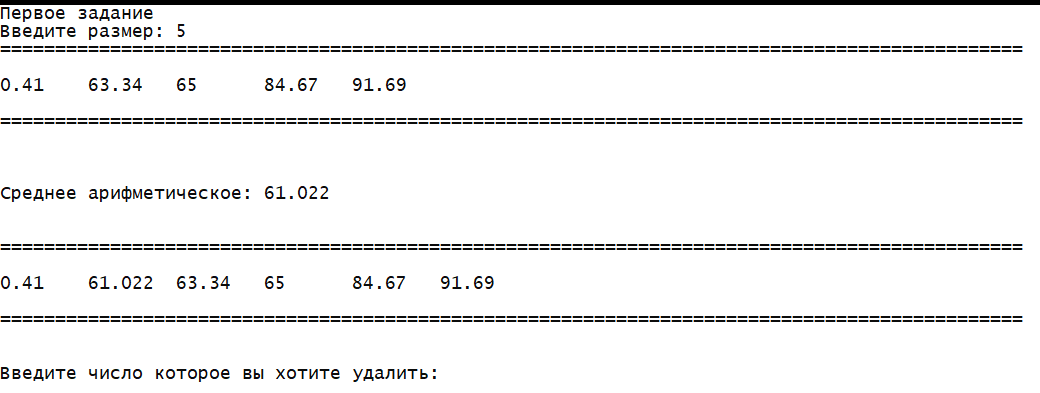
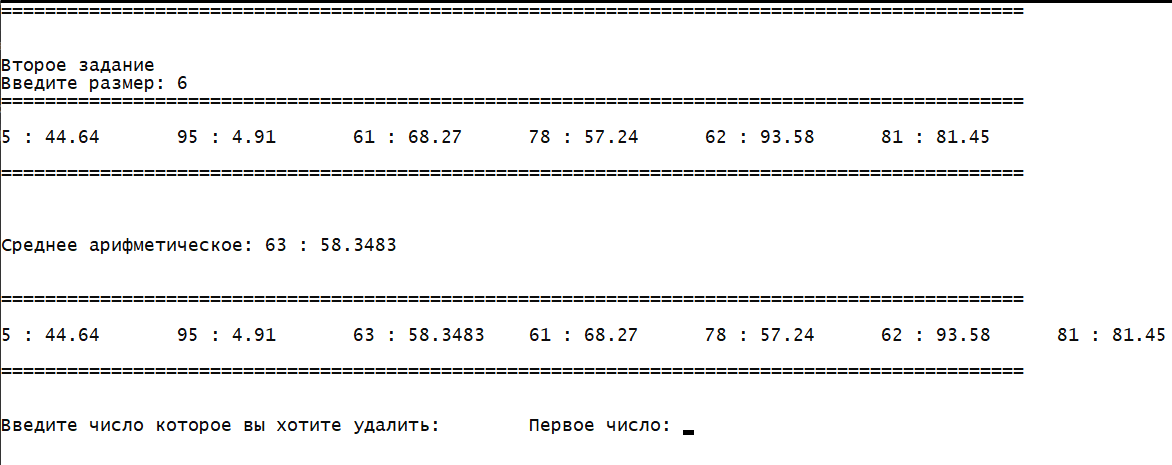
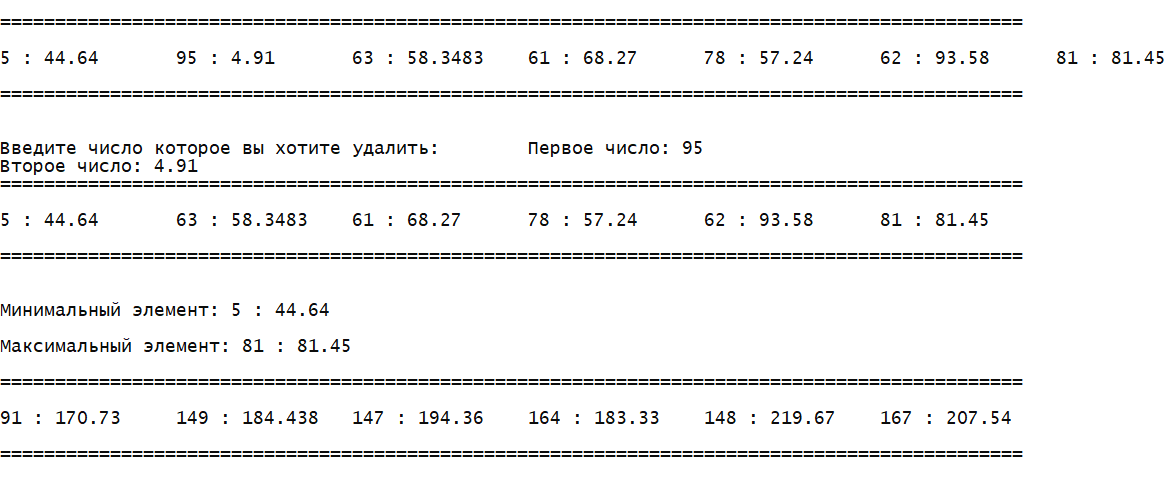
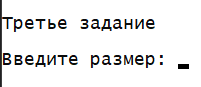
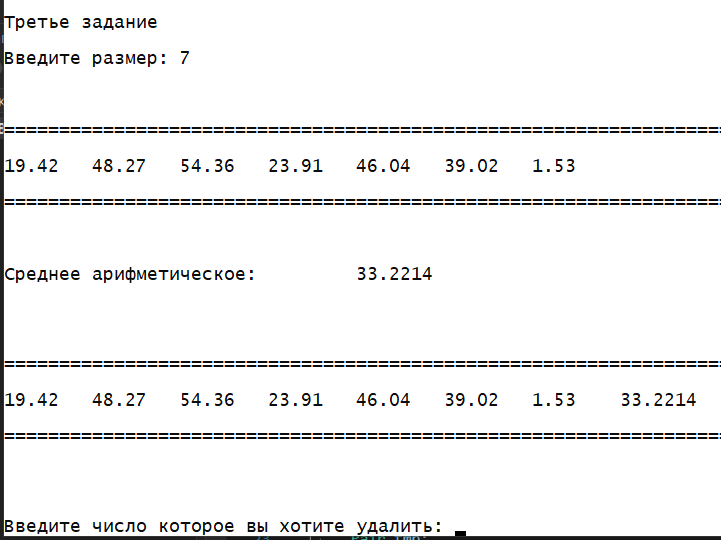
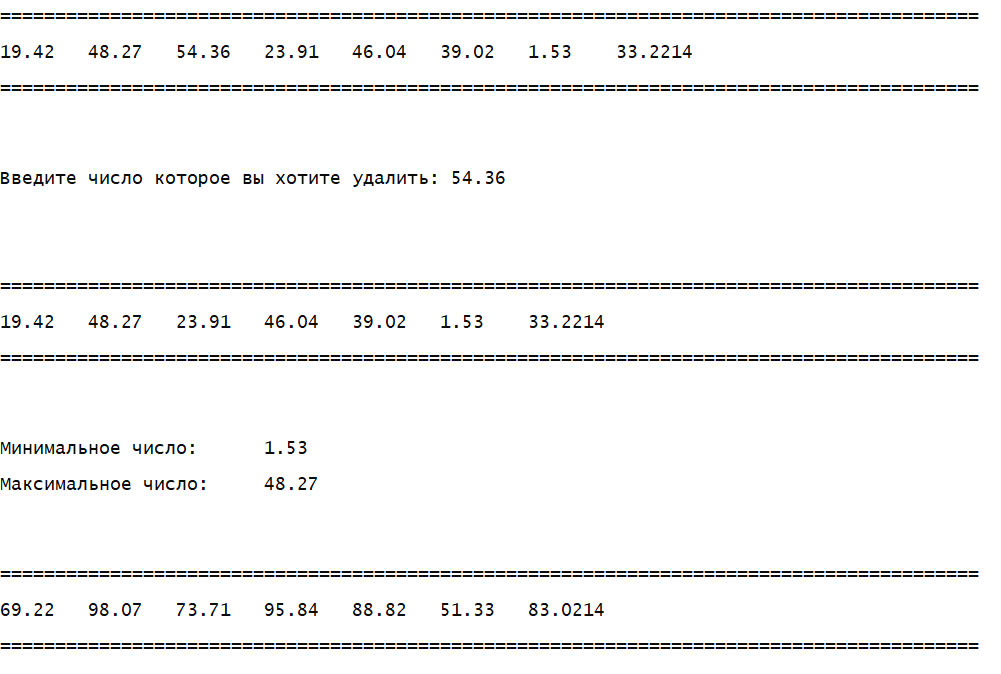
**Блок-схема**







**Скриншоты**

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

**Ответы на вопросы**

