Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по лабораторной работе №12

Тема: «Ассоциативные контейнеры библиотеки STL.»

Выполнили:

Студенты группы РИС-22-2б

Худеньких В.Д.

Проверил доц. Кафедры ИТАС:

Полякова О.А.

Пермь 2023

# Постановка задачи

Задача 1.

* + 1. Создать ассоциативный контейнер.
    2. Заполнить его элементами стандартного типа (тип указан в варианте).
    3. Добавить элементы в соответствии с заданием
    4. Удалить элементы в соответствии с заданием.
    5. Выполнить задание варианта для полученного контейнера.
    6. Выполнение всех заданий оформить в виде глобальных функций.

Задача 2.

1. Создать ассоциативный контейнер.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде глобальных функций.

Задача 3

1. Создать параметризированный класс, используя в качестве контейнера ассоциативный контейнер.
2. Заполнить его элементами.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде методов параметризированного класса.

ВАРИАНТ 15:

|  |  |  |
| --- | --- | --- |
| **Задача 1**   1. Контейнер - multiset 2. Тип элементов - double   **Задача 2**  Тип элементов Pair (см. лабораторную работу №3).  **Задача 3**  Параметризированный класс – Список (см. лабораторную работу №7) | | |
| **Задание 3** | **Задание 4** | **Задание 5** |
| Найти среднее арифметическое и добавить его в конец  контейнера | Найти элементы ключами из заданного диапазона и удалить их  из контейнера | К каждому элементу добавить сумму минимального и максимального элементов  контейнера. |

**Контрольные вопросы**

1. Что представляет собой ассоциативный контейнер?

Ассоциативный контейнер - это контейнер, который содержит упорядоченный набор элементов, представленных в форме пар ключ-значение. Элементы располагаются в контейнере в соответствии с заданным отношением порядка, определенным в компараторе. Ключи уникальны в контейнере, т.е. в контейнере не может быть двух элементов с одинаковым ключом. Основным преимуществом ассоциативных контейнеров является возможность быстрого поиска элемента по ключу благодаря использованию бинарного дерева поиска (например, красно-черного дерева) для хранения элементов. Примерами ассоциативных контейнеров в STL являются map и set.

1. Перечислить ассоциативные контейнеры в библиотеке STL.

* set - контейнер, хранящий уникальные элементы в отсортированном порядке. Все элементы в set уникальны, то есть не могут дублироваться.
* map - контейнер, хранящий пары ключ-значение, отсортированные по ключу. Ключи в map должны быть уникальными.
* multiset - контейнер, хранящий уникальные элементы в отсортированном порядке, но допускающий дублирование элементов.
* multimap - контейнер, хранящий пары ключ-значение, отсортированные по

1. Каким образом можно получить доступ к элементам ассоциативного контейнера?

Для доступа к элементам ассоциативного контейнера в STL используются итераторы. В отличие от последовательных контейнеров, в ассоциативных контейнерах элементы хранятся не в порядке их добавления, а в отсортированном порядке на основе ключа. Поэтому для доступа к элементам по индексу, как в последовательных контейнерах, в ассоциативных контейнерах используют итераторы. С помощью итераторов можно получить доступ к ключу и соответствующему ему значению элемента контейнера.

1. Привести примеры методов, используемых в ассоциативных контейнерах.

* insert(): добавляет элемент в контейнер.
* erase(): удаляет элемент из контейнера.
* find(): ищет элемент по заданному ключу.
* count(): возвращает количество элементов с заданным ключом в контейнере.
* size(): возвращает количество элементов в контейнере.
* empty(): возвращает значение true, если контейнер пуст, и false в противном случае.
* begin(): возвращает итератор, указывающий на первый элемент в контейнере.
* end(): возвращает итератор, указывающий на элемент следующий за последним \* элементом контейнера.
* lower\_bound(): возвращает итератор на первый элемент в контейнере, не меньший \* заданного ключа.
* upper\_bound(): возвращает итератор на первый элемент в контейнере, больший заданного ключа.
* equal\_range(): возвращает диапазон элементов в контейнере, соответствующих заданному ключу.

1. Каким образом можно создать контейнер map? Привести примеры.

* С помощью конструктора по умолчанию:

map<string, int> myMap;

В данном случае создается пустой контейнер map с ключами типа string и значениями типа int.

* С помощью списка инициализации:

map<string, int> myMap = {{"apple", 1}, {"banana", 2}, {"cherry", 3}};

В данном случае создается контейнер map с начальными значениями ключей и значений, которые передаются в список инициализации.

* С помощью пары итераторов:

map<std::string, int> myMap(anotherMap.begin(), anotherMap.end());

В данном случае создается контейнер map, который инициализируется парами ключ-значение из другого контейнера, заданного итераторами begin() и end().

* С помощью списка пар ключ-значение:

map<std::string, int> myMap = {make\_pair("apple", 1), make\_pair("banana", 2),make\_pair("cherry", 3)};

В данном случае создается контейнер map, который инициализируется парами ключ-значение из списка пар, созданных с помощью функции make\_pair().

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

Элементы отсортированы в порядке возрастания ключей.

Чтобы изменить порядок, можно определить пользовательскую функцию сравнения, которая будет сравнивать ключи в обратном порядке. Например:

bool compare(int a, int b) {

return a > b;

}

Затем, мы можем создать map следующим образом:

std::map<int, std::string, decltype(compare)\*> myMap(compare);

1. Какие операции определены для контейнера map?

Контейнер map поддерживает операции добавления и удаления элементов, поиска и доступа к элементам по ключу, а также проверки наличия элементов в контейнере. Кроме того, контейнер map поддерживает итераторы для обхода содержимого.

1. Написать функцию для добавления элеентов в контейнер map с помощью функции make\_pair().

template <class T1, class T2>

void addElements(std::map<T1, T2>& m, int n) {

T1 temp1 = 0;

T2 temp2;

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

{

cin >> temp2;

m.insert(make\_pair(temp1++, temp2));

}

}

1. Написать функцию для добавления элементов в контейнер map с помощь функции операци прямого доступпа [].

template <class T1, class T2>

void addElements(std::map<T1, T2>& m, T1 key, T2 value) {

m[key] = value;

}

1. Написать функцию для для печати контейнера map с помощью итератора.

template <class T1, class T2>

void printMap(map<T1, T2>& m) {

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

cout << it->first << " : " << it->second << endl;

}

}

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

template <class T1, class T2>

void printMap(map<T1, T2>& m) {

for (const auto& p : m)

cout << p.first << " : " << m[p.first] << endl;

}

1. Чем отличаются контейнеры map и multimap?

Отличие между map и multimap заключается в том, что map хранит только уникальные ключи и соответствующие значения, тогда как multimap может хранить несколько значений для одного и того же ключа. Другими словами, map — это контейнер с уникальными ключами, а multimap - контейнер с неуникальными ключами.

1. Что представляет собой контейнер set?

Контейнер set - упорядоченное множество уникальных элементов. Он реализован в виде бинарного дерева поиска и обеспечивает быстрый доступ, вставку и удаление элементов в отсортированном порядке.

1. Чем отличаются контейнеры map и set?

Контейнер map предназначен для хранения пары "ключ-значение", где каждый ключ уникален, а контейнер set используется для хранения уникальных элементов, без пары "ключ-значение".

Таким образом, map используется для хранения и доступа к значению по ключу, а set используется для хранения элементов в отсортированном порядке и быстрого поиска элементов по значению.

1. Каким ообразом можно создать контейнер set? Привести примеры.

* Создание пустого контейнера с помощью конструктора по умолчанию:

set<int> mySet;

* Создание с заданными начальными значениями с помощью списка инициализации:

set<int> mySet = {1, 2, 3, 4};

* Создание с помощью диапазона значений другого контейнера:

vector<int> myVec = {1, 2, 3, 4};

set<int> mySet(myVec.begin(), myVec.end());

* Создание пустого контейнера с заданным компаратором:

struct Compare {

bool operator()(int a, int b) const {

return a > b;

}

};

set<int, Compare> mySet(Compare());

* Создание с заданными начальными значениями и компаратором:

struct Compare {

bool operator()(int a, int b) const {

return a > b;

}

};

set<int, Compare> mySet = {1, 2, 3, 4};

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

Элементы в контейнере set упорядочены по возрастанию. Чтобы изменить порядок на убывание, можно задать компаратор при создании контейнера, который будет сравнивать элементы в обратном порядке. Например:

#include <functional> // для std::greater

int main() {

set<int,greater<int>> s {5, 2, 7, 1, 8};

// элементы будут упорядочены в порядке убывания

return 0;

}

Здесь std::greater<int> - это функциональный объект, который сравнивает элементы в порядке убывания. Он передается вторым параметром шаблона контейнера set.

1. Какие методы определены для контейнера set?

* insert() - добавляет элемент в контейнер
* erase() - удаляет элемент из контейнера по значению или по итератору
* find() - ищет элемент в контейнере и возвращает итератор на найденный элемент, либо итератор на конец контейнера, если элемент не найден
* size() - возвращает количество элементов в контейнере
* empty() - возвращает true, если контейнер пуст, иначе – false
* clear() - удаляет все элементы из контейнера

1. Написать функцию для добавления элементов в контейнер set.

template <class T1>

void addElements(set<T1>& st, int n) {

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

st.insert(T1(rand()));

}

1. Написать функцию для печати контейнера set.

template <class T1>

void printSet(set<T1>& st) {

for (const auto& i : st)

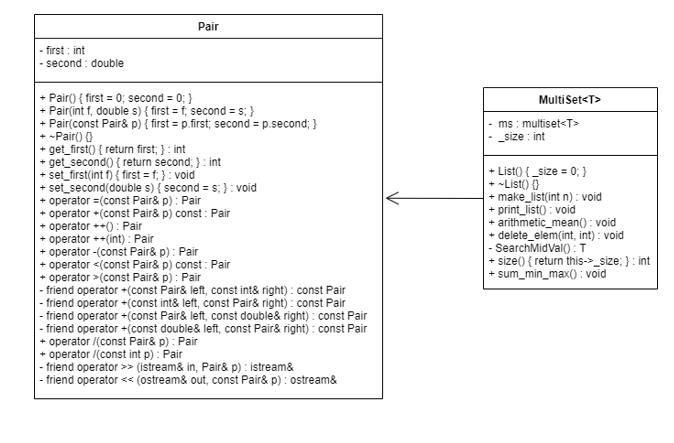
cout << i << endl;

}

1. Чем отличается контейнер set и multiset?

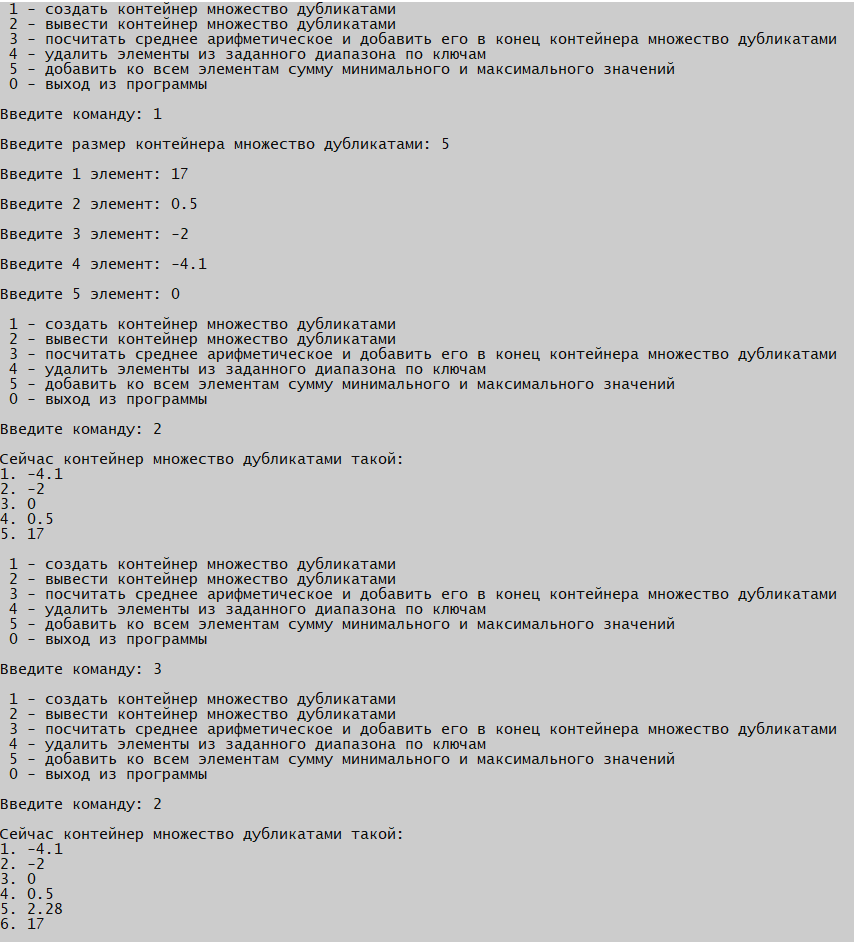
Отличие между ними заключается в том, что set может хранить только уникальные элементы, а multiset может хранить несколько одинаковых элементов.

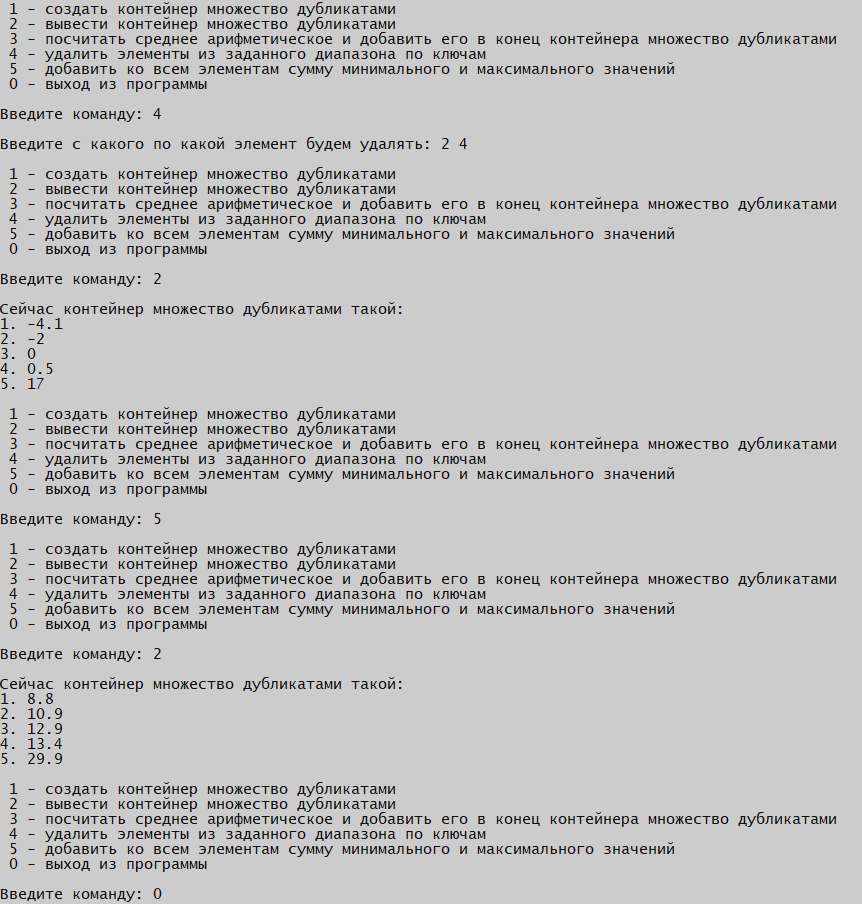
**UML таблица**

****

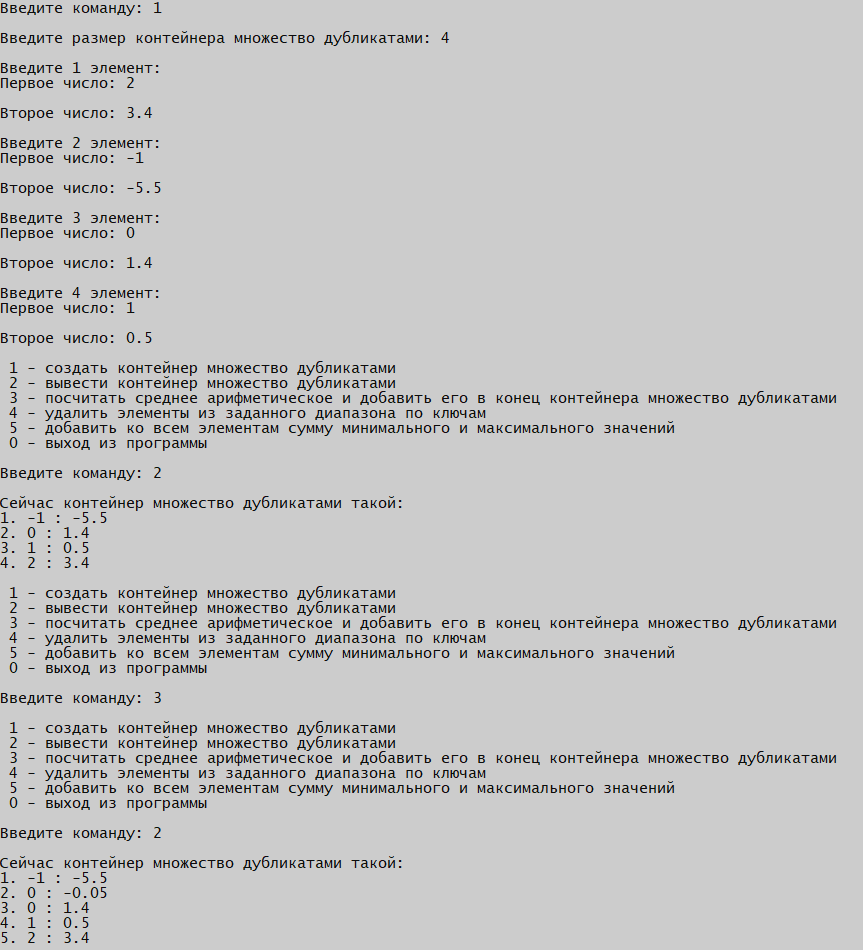
**Скриншоты работы программы**

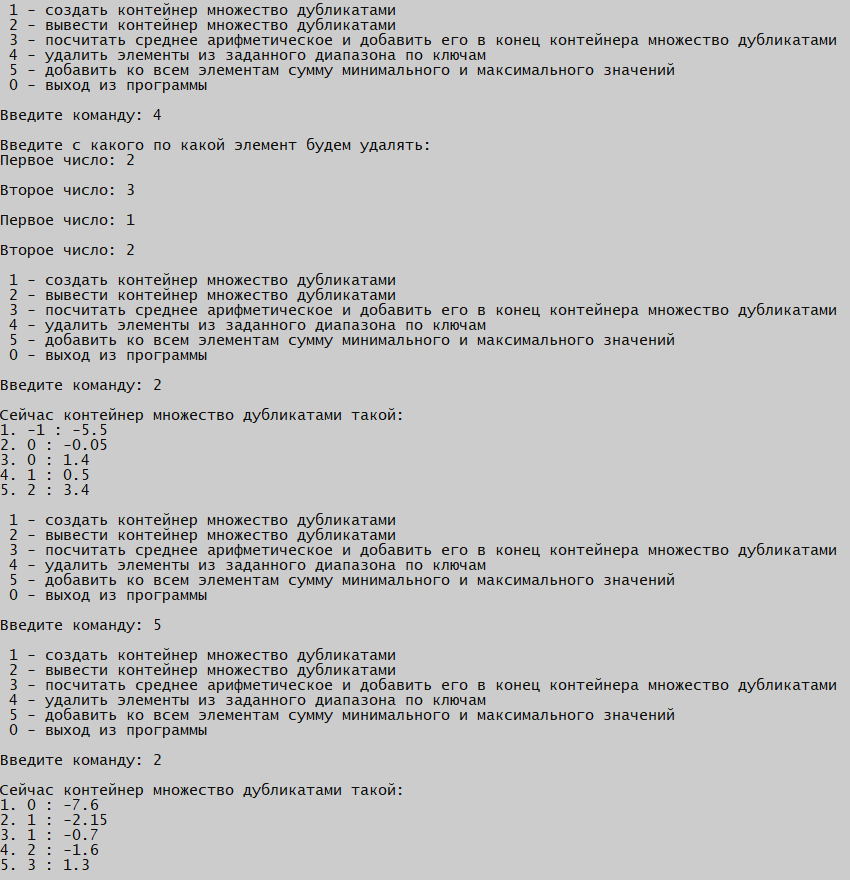
**Задача 1.**

****

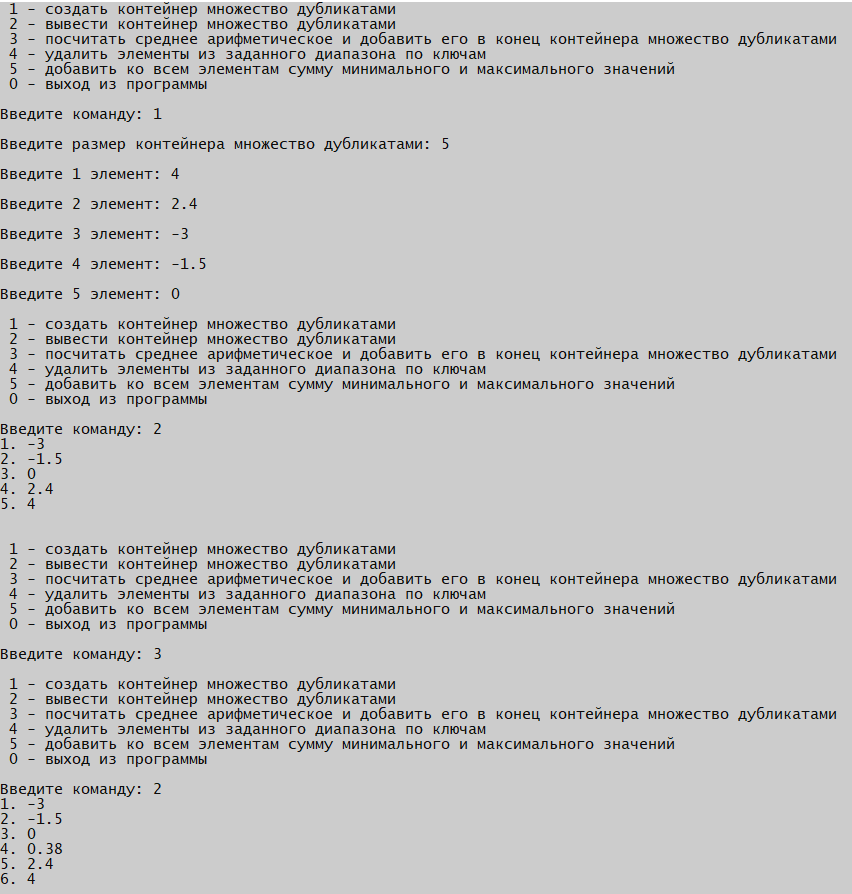
****

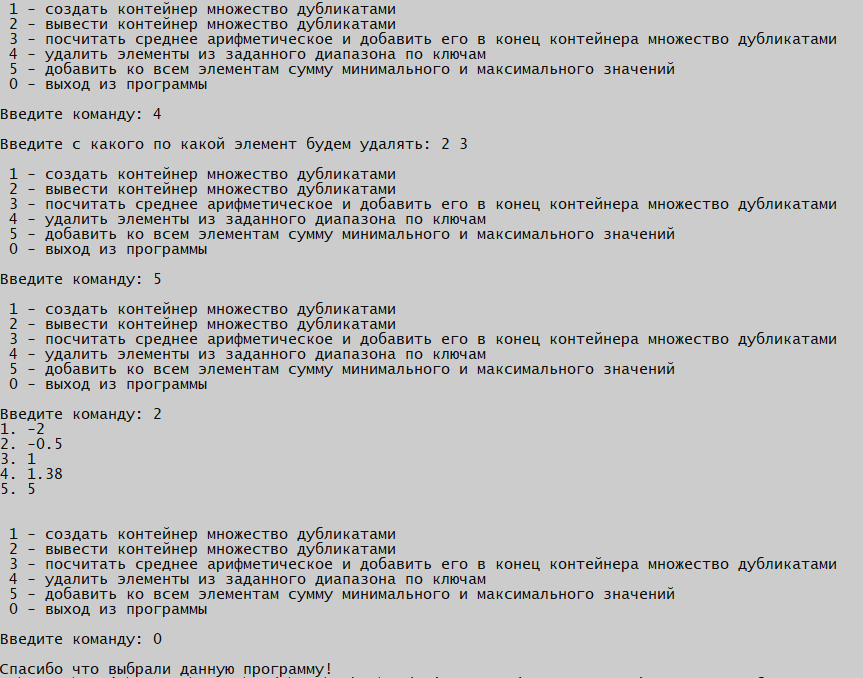
**Задача 2.**

****

****

**Задача 3.**

****

****

**Код программы**

**Задача 1.**

**main.cpp:**

#include <iostream>

#include <set>

#include "Pair.h"

using namespace std;

bool operator<(const Pair& pr1, const Pair& pr2)

{

return ((pr1.a < pr2.a) && (pr1.b < pr2.b));

}

void ex\_1()

{

setlocale(0, "rus");

multiset <double> mst;

multiset <double> tmp;

mst.insert(1.5);

mst.insert(-1.5);

mst.insert(2.3);

mst.insert(2.3);

double sr = 0;

cout << "Изначальное множество" << endl;

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

{

sr += \*it;

cout << \*it << endl;

}

//1

cout << "Среднее, добавленное в конец: " << sr / mst.size() << endl;

mst.insert(sr / mst.size());

//2

double max = \*mst.begin();

double min = \*mst.begin();

tmp = mst;

mst.clear();

int c = 0;

int i, j;

double res;

cout << "Задайте диапозон: "; cin >> i >> j; cout << "Задайте ключ: "; cin >> res;

for (auto it = tmp.begin(); it != tmp.end(); it++, c++)

{

if (!(\*it == res && (c > i && c < j)))

{

if (max < \*it) max = \*it;

if (min > \*it) min = \*it;

mst.insert(\*it);

cout << \*it << endl;

}

}

cout << "Добавление суммы мин и макс: " << endl;

tmp = mst;

mst.clear();

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

{

mst.insert(max + min + \*it);

cout << \*it + max + min << endl;

}

}

void ex\_2()

{

setlocale(0, "rus");

multiset <Pair> mst;

multiset <Pair> tmp;

mst.insert(Pair(1, 1.5));

mst.insert(Pair(1, -1.5));

mst.insert(Pair(2.3, 2.3));

mst.insert(Pair(2.3, 2.3));

Pair sr(0, 0);

cout << "Изначальное множество" << endl;

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

{

sr = sr + \*it;

cout << \*it << endl;

}

//1

sr.a /= mst.size();

sr.b /= mst.size();

cout << "Среднее, добавленное в конец: " << sr << endl;

mst.insert(sr);

//2

Pair max = \*mst.begin();

Pair min = \*mst.begin();

tmp = mst;

mst.clear();

int c = 0;

int i, j;

Pair res;

cout << "Задайте диапозон: "; cin >> i >> j; cout << "Задайте ключ: "; cin >> res;

for (auto it = tmp.begin(); it != tmp.end(); it++, c++)

{

if ((c < i) || (c > j) || (res.a != (\*it).a || res.b != (\*it).b))

{

if (max < \*it) max = \*it;

if (min > \*it) min = \*it;

mst.insert(\*it);

cout << \*it << endl;

}

}

cout << "Добавление суммы мин и макс: " << endl;

tmp = mst;

mst.clear();

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

{

mst.insert(max + min + \*it);

cout << max + min + \*it << endl;

}

}

void ex\_3()

{

setlocale(0, "rus");

multiset <Pair> mst;

multiset <Pair> tmp;

mst.insert(Pair(1, 1.5));

mst.insert(Pair(1, -1.5));

mst.insert(Pair(2.3, 2.3));

mst.insert(Pair(2.3, 2.3));

Pair sr(0, 0);

cout << "Изначальное множество" << endl;

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

{

sr = sr + \*it;

cout << \*it << endl;

}

//1

sr.a /= mst.size();

sr.b /= mst.size();

cout << "Среднее, добавленное в конец: " << sr << endl;

mst.insert(sr);

//2

Pair max = \*mst.begin();

Pair min = \*mst.begin();

tmp = mst;

mst.clear();

int c = 0;

int i, j;

Pair res;

cout << "Задайте диапозон: "; cin >> i >> j; cout << "Задайте ключ: "; cin >> res;

for (auto it = tmp.begin(); it != tmp.end(); it++, c++)

{

if ((c < i) || (c > j) || (res.a != (\*it).a || res.b != (\*it).b))

{

if (max < \*it) max = \*it;

if (min > \*it) min = \*it;

mst.insert(\*it);

cout << \*it << endl;

}

}

cout << "Добавление суммы мин и макс: " << endl;

tmp = mst;

mst.clear();

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

{

mst.insert(max + min + \*it);

cout << max + min + \*it << endl;

}

}

int main()

{

ex\_1();

ex\_2();

ex\_3();

}

**Pair.h:**

#pragma once

#include <iostream>

using namespace std;

class **Pair**

{

public:

int a;

double b;

friend istream& operator>>(istream& in, Pair& p);

friend ostream& operator<<(ostream& out, const Pair& p);

**Pair**(int a, double b);

**Pair**();

**Pair**(const Pair& p);

~**Pair**();

Pair& operator=(const Pair& p);

Pair operator-(const Pair& p);

Pair& operator+(int a);

Pair& operator+(double b);

Pair& operator++();

Pair operator++(int);

Pair operator+(const Pair& p);

bool operator!=(const Pair& p) { return !((this->a == p.a) && (this->b == p.b)); }

bool operator==(const Pair& p) { return ((this->a == p.a) && (this->b == p.b)); }

bool operator>(const Pair& p) { return (this->a > p.a) || (this->b > p.b); }

bool operator>=(const Pair& p) { return (this->a >= p.a) || (this->b >= p.b); }

bool operator<=(const Pair& p) { return (this->a <= p.a) || (this->b <= p.b); }

bool operator<(const Pair& p) { return (this->a < p.a) || (this->b < p.b); }

Pair operator \* (Pair& pair)

{

Pair new\_pair;

new\_pair.a = this->a \* pair.a;

new\_pair.b = this->b \* pair.b;

return new\_pair;

}

};

**Pair.cpp:**

#include "Pair.h"

Pair::**Pair**(int a, double b)

{

this->a = a;

this->b = b;

}

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

{

cout << "(int)a : ";

in >> p.a;

cout << "(double)b :";

in >> p.b;

return in;

}

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

{

return (out << p.a << " : " << p.b);

}

Pair::**Pair**()

{

a = 0;

b = 0;

}

Pair::~**Pair**()

{

}

Pair::**Pair**(const Pair& p)

{

a = p.a;

b = p.b;

}

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

{

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

a = p.a;

b = p.b;

return \*this;

}

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

{

Pair res(a - p.a, b - p.b);

return res;

}

Pair& Pair::operator+(int a)

{

this->a += a;

return \*this;

}

Pair& Pair::operator+(double b)

{

this->b += b;

return \*this;

}

Pair& Pair::operator++()

{

++a; ++b;

return \*this;

}

Pair Pair::operator ++(int)

{

Pair temp = \*this;

this->a++; this->b++;

return temp;

}

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

{

this->a += p.a;

this->b += p.b;

return \*this;

}

**LinkedList.h:**

#pragma once

#pragma once

#include <iostream>

#include <algorithm>

#include "D:\?++\Lab\_OOP\_9\Error.h"

using namespace std;

template<typename T>

struct **Point**

{

T num;

Point\* next = nullptr;

};

template <typename T>

class **Iterator**

{

private:

Point<T>\* elem = nullptr;

public:

**Iterator**() { elem = nullptr; }

**Iterator**(Point<T>\* point) { elem = point; }

**Iterator**(const Iterator& it) { elem = it.elem; };

bool operator == (const Iterator& it) { return elem == it.elem; }

bool operator != (const Iterator& it) { return elem != it.elem; }

Iterator& operator ++ ();

Iterator operator ++ (int);

Iterator& operator + (int n);

T operator \* () const;

};

template <typename T>

T Iterator<T>::operator \* () const

{

if (elem)

return elem->num;

else

throw IndexError2();

}

template <typename T>

Iterator<T>& Iterator<T>::operator ++ ()

{

if (elem == nullptr) throw IndexError2();

elem = elem->next;

return \*this;

}

template <typename T>

Iterator<T> Iterator<T>::operator ++ (int)

{

if (elem == nullptr) throw IndexError2();

Iterator<T> temp(elem);

++(\*this);

return temp;

}

template <typename T>

Iterator<T>& Iterator<T>::operator + (int n)

{

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

{

if (elem == nullptr) throw IndexError2();

elem = elem->next;

}

return \*this;

}

template <typename T>

class **LinkedList**

{

private:

size\_t size = 0;

Point<T>\* head = nullptr;

Point<T>\* top = nullptr;

void **output**(Point<T>\* obj) const;

void **show**() const;

public:

friend ostream& operator<<(ostream& out, const LinkedList<T>& list)

{

list.show();

return out;

}

friend istream& operator>>(istream& in, LinkedList<T>& list)

{

T k;

in >> k;

list.push(k);

return in;

}

**LinkedList**();

**LinkedList**(size\_t s, T k);

~**LinkedList**();

**LinkedList**(const LinkedList<T>& list);

T& operator [](int index) const;

LinkedList& operator =(const LinkedList<T>& list);

LinkedList operator \* (LinkedList<T>& list);

int operator ()() { return size; };

void **push**(T k);

T **pop**();

Iterator<T> **begin**() { return Iterator<T>(head); }

Iterator<T> **end**() { return Iterator<T>(top->next); }

};

template <typename T>

LinkedList<T>::**LinkedList**()

{

head = nullptr;

top = nullptr;

size = 0;

}

template <typename T>

LinkedList<T>::**LinkedList**(size\_t s, T k)

{

size = s;

if (size != 0) {

Point<T>\* obj = new Point<T>;

obj->num = k;

head = obj;

Point<T>\* temp = head;

for (int i = size - 1; i != 0; i--)

{

obj = new Point<T>;

temp->next = obj;

obj->num = k;

temp = obj;

}

top = obj;

}

}

template <typename T>

void LinkedList<T>::**push**(T k)

{

if (size == 0)

{

head = new Point<T>;

head->num = k;

top = head;

}

else {

Point<T>\* obj = new Point<T>;

top->next = obj;

obj->num = k;

top = obj;

}

size++;

}

template <typename T>

T LinkedList<T>::**pop**()

{

if (size == 0) throw EmptySizeError();

T k = head->num;

Point<T>\* temp = head;

head = head->next;

size--;

delete temp;

return k;

}

template <typename T>

LinkedList<T>::~**LinkedList**()

{

cout << endl << "?????? ??????????!" << endl << endl;

while (head != nullptr)

{

pop();

}

}

template <typename T>

void LinkedList<T>::**output**(Point<T>\* obj) const

{

cout << obj->num << " ";

if (obj == top) return;

output(*obj->*next);

}

template <typename T>

void LinkedList<T>::**show**() const

{

if (size == 0) cout << "?????? ????" << endl;

else output(*head*);

}

template <typename T>

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

{

if (index == 0 && size == 0) throw IndexError();

if (index < 0) throw IndexError1();

if (index >= size) throw IndexError2();

Point<T>\* temp = head;

int count = 0;

while (temp != nullptr)

{

if (count == index)

return temp->num;

temp = temp->next;

++count;

}

}

template <typename T>

LinkedList<T>::**LinkedList**(const LinkedList<T>& list)

{

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

{

this->push(list[i]);

}

}

template <typename T>

LinkedList<T>& LinkedList<T>::operator =(const LinkedList<T>& list)

{

int size\_temp = size;

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

{

this->pop();

}

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

{

this->push(list[i]);

}

return \*this;

}

template <typename T>

LinkedList<T> LinkedList<T>::operator \* (LinkedList<T>& list)

{

T nulik;

nulik = 0;

LinkedList<T> new\_list;

for (int i = 0; i < min((int)size, list()); i++)

{

new\_list.push((\*this)[i] \* list[i]);

}

for (int i = 0; i < max((int)size, list()) - min((int)size, list()); i++)

{

new\_list.push(nulik);

}

return new\_list;

}