Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**Тема:** Лабораторная работа №12 по ООП

Семестр: 2

Выполнил студент ИВТ-22-2б:

Дияров Тимур Артурович

(дата, подпись)

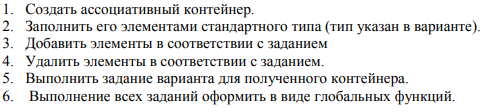
Проверила:

Полякова Ольга Андреевна

(дата, подпись)

Пермь 2023

**Задача 1:**



**Программное решение**

#include <iostream>

#include <set>

using namespace std;

void PushArithmeticMiddle(multiset<double>& m\_multiset) {

if (m\_multiset.size() > 0) {

double Sum = 0;

for (auto i = m\_multiset.begin(); i != m\_multiset.end(); i++) {

Sum += \*i;

}

m\_multiset.insert(Sum / m\_multiset.size());

}

else {

cout << "Множество пусто!\n";

}

}

void EraseIfInRange(multiset<double>& m\_multiset, const double BeginValue, const double EndValue) {

auto i = m\_multiset.begin();

while (i != m\_multiset.end()) {

if (\*i >= BeginValue && \*i <= EndValue) {

i = m\_multiset.erase(i);

}

else {

i++;

}

}

}

void PlusMinAndMax(multiset<double>& m\_multiset) {

if (m\_multiset.size() > 0) {

multiset<double> tmp\_multiset = m\_multiset;

double MaxValue, MinValue;

MaxValue = MinValue = \*m\_multiset.begin();

for (auto i = m\_multiset.begin(); i != m\_multiset.end(); i++) {

if (\*i > MaxValue) {

MaxValue = \*i;

}

else if (\*i < MinValue) {

MinValue = \*i;

}

}

m\_multiset = {};

for (auto i = tmp\_multiset.begin(); i != tmp\_multiset.end(); i++) {

double NewValue = \*i + MaxValue + MinValue;

m\_multiset.insert(NewValue);

}

tmp\_multiset = {};

}

}

void ShowMultiset(const multiset<double>& m\_multiset) {

if (m\_multiset.size() > 0) {

for (auto i = m\_multiset.begin(); i != m\_multiset.end(); i++) {

cout << \*i << ' ';

}

}

else {

cout << "Множество пусто!";

}

cout << '\n';

}

int main() {

system("chcp 1251 > NULL");

multiset<double> a = { 1, 2, 3, 4, 5, 6, 7};

cout << "Множество с дубликатами а:\n";

ShowMultiset(a);

cout << "Добавляем среднее арифмитическое множестава а в контейнер\n";

PushArithmeticMiddle(a);

cout << "Множество а:\n";

ShowMultiset(a);

cout << "Удаляем из множества а элементы, значения которых входят в диапазон от 2 до 3\n";

EraseIfInRange(a, 2, 3);

cout << "Множество а:\n";

ShowMultiset(a);

cout << "К каждому элементу добавляем сумму минимального и максимального элементов контейнера\n";

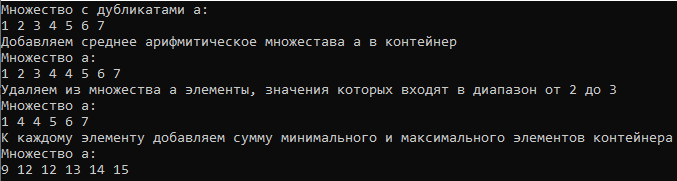
PlusMinAndMax(a);

cout << "Множество а:\n";

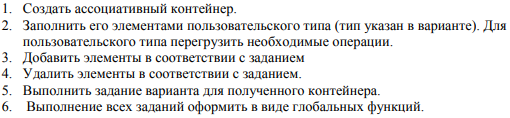
ShowMultiset(a);

return 0;

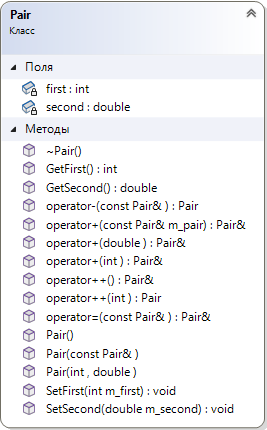
}



**Задача 2:**



**UML-Диаграмма**



**Программное решение**

**Pair.h**

#pragma once

#include <iostream>

using namespace std;

class Pair {

int first;

double second;

public:

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

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

Pair(int, double);

Pair();

Pair(const Pair&);

~Pair();

Pair& operator=(const Pair&);

Pair operator-(const Pair&);

Pair& operator+(int);

Pair& operator+(double);

Pair& operator++();

Pair operator++(int);

int GetFirst() const;

double GetSecond() const;

friend bool operator<(const Pair&, const Pair&);

void SetFirst(int m\_first);

void SetSecond(double m\_second);

Pair& operator + (const Pair& m\_pair);

};

**Pair.cpp**

#include "Pair.h"

Pair::Pair(int first, double second) {//конструктор с параметрами

this->first = first;

this->second = second;

}

istream& operator>>(istream& in, Pair& m\_pair) {//перегрузка оператора ввода

in >> m\_pair.first;

in >> m\_pair.second;

return in;

}

ostream& operator<<(ostream& out, const Pair& m\_pair) {//перегрузка оператора вывода

return (out << m\_pair.first << " : " << m\_pair.second);

}

Pair::Pair() {//конструктор без параметров

first = 0;

second = 0;

}

Pair::~Pair() {//деструктор

}

Pair::Pair(const Pair& m\_pair) {//конструктор копирования

first = m\_pair.first;

second = m\_pair.second;

}

Pair& Pair::operator=(const Pair& m\_pair) {//перегрузка оператора присваивания

if (&m\_pair != this) {

first = m\_pair.first;

second = m\_pair.second;

}

return \*this;

}

Pair Pair::operator-(const Pair& m\_pair) {//перегрузка оператора вычитания

Pair result(first - m\_pair.first, second - m\_pair.second);

return result;

}

Pair& Pair::operator+(int first) {//перегрузка оператора сложения - целый аргумент

this->first += first;

return \*this;

}

Pair& Pair::operator+(double second) {//перегрузка оператора сложения - дробный аргумент

this->second += second;

return \*this;

}

Pair& Pair::operator++() {//перегрузка оператора ++ префикс

++first;

++second;

return \*this;

}

Pair Pair::operator ++(int) {//перегрузка оператора ++ постфикс

Pair temp = \*this;

this->first++;

this->second++;

return temp;//сначала нужно вывести без изменений

}

int Pair::GetFirst() const{

return first;

}

double Pair::GetSecond() const{

return second;

}

bool operator < (const Pair& pair1, const Pair& pair) {

return (pair1.GetFirst() + pair1.GetSecond() < pair.GetFirst() + pair.GetSecond());

}

void Pair::SetFirst(int m\_first) {

first = m\_first;

}

void Pair::SetSecond(double m\_second) {

second = m\_second;

}

Pair& Pair::operator + (const Pair& m\_pair) {

first += m\_pair.first;

second += m\_pair.second;

return \*this;

}

**task2.cpp**

#include <iostream>

#include "Pair.h"

using namespace std;

#include <set>

void PushArithmeticMiddle(multiset<Pair>& m\_multiset) {

if (m\_multiset.size() > 0) {

int SumFirst = 0;

double SumSecond = 0;

for (auto i = m\_multiset.begin(); i != m\_multiset.end(); i++) {

SumFirst += (\*i).GetFirst();

SumSecond += (\*i).GetSecond();

}

m\_multiset.insert(Pair(SumFirst / m\_multiset.size(), SumSecond / m\_multiset.size()));

}

else {

cout << "Множество пусто!\n";

}

}

void EraseIfInRange(multiset<Pair>& m\_multiset, const double BeginValue, const double EndValue) {

auto i = m\_multiset.begin();

while (i != m\_multiset.end()) {

if (((\*i).GetFirst() >= BeginValue && (\*i).GetFirst() <= EndValue) || ((\*i).GetSecond() >= BeginValue && (\*i).GetSecond() <= EndValue)) {

i = m\_multiset.erase(i);

}

else {

i++;

}

}

}

void PlusMinAndMax(multiset<Pair>& m\_multiset) {

if (m\_multiset.size() > 0) {

multiset<Pair> tmp\_multiset;

Pair MaxValue = \*(m\_multiset.begin());

Pair MinValue = \*(m\_multiset.begin());

for (auto i = m\_multiset.begin(); i != m\_multiset.end(); i++) {

if (MaxValue < \*i) {

MaxValue = \*i;

}

else if (\*i < MinValue) {

MinValue = \*i;

}

}

for (auto i = m\_multiset.begin(); i != m\_multiset.end(); i++) {

Pair c = (\*i);

c + MaxValue + MinValue;

tmp\_multiset.insert(c);

}

m\_multiset = tmp\_multiset;

}

}

void ShowMultiset(const multiset<Pair>& m\_multiset) {

if (m\_multiset.size() > 0) {

for (auto i = m\_multiset.begin(); i != m\_multiset.end(); i++) {

cout << \*i << '\n';

}

}

else {

cout << "Множество пусто!";

}

cout << '\n';

}

int main() {

system("chcp 1251 > NULL");

multiset<Pair> a = { Pair(6, 4), Pair(7, 9), Pair(2, 3.5) };

cout << "Множество с дубликатами а:\n";

ShowMultiset(a);

cout << "Добавляем среднее арифмитическое множества а в контейнер\n";

PushArithmeticMiddle(a);

cout << "Множество а:\n";

ShowMultiset(a);

cout << "Удаляем из множества а элементы, значения которых входят в диапазон от 2 до 3\n";

EraseIfInRange(a, 2, 3);

cout << "Множество а:\n";

ShowMultiset(a);

cout << "К каждому элементу добавляем сумму минимального и максимального элементов контейнера\n";

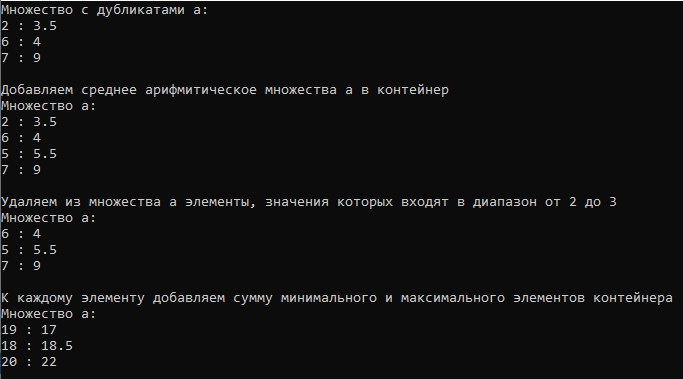
PlusMinAndMax(a);

cout << "Множество а:\n";

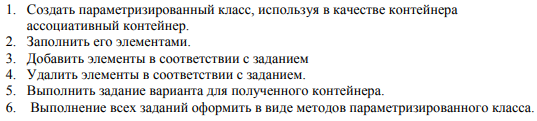
ShowMultiset(a);

return 0;

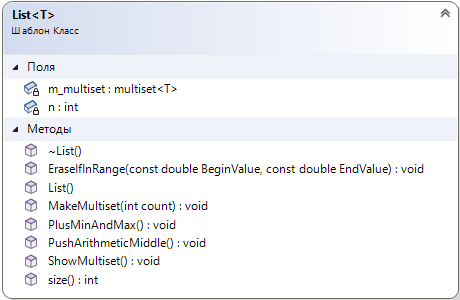
}



**Задача 3:**



**UML-Диаграмма**



**Программное решение**

**List.h**

#pragma once

#include <iostream>

#include <set>

using namespace std;

template <typename T>

class List {

private:

multiset<T> m\_multiset;

int n = 0;

public:

List() {};

~List() {};

void MakeMultiset(int count) {

T a;

for (size\_t i = 0; i < count; i++) {

cout << "Введите " << i + 1 << " элемент: ";

cin >> a;

m\_multiset.insert(a);

}

n = count;

};

void ShowMultiset() {

if (m\_multiset.size() > 0) {

for (auto i = m\_multiset.begin(); i != m\_multiset.end(); i++) {

cout << \*i << ' ';

}

}

else {

cout << "Множество пусто!";

}

cout << '\n';

};

void PushArithmeticMiddle() {

if (m\_multiset.size() > 0) {

T Sum = 0;

for (auto i = m\_multiset.begin(); i != m\_multiset.end(); i++) {

Sum += \*i;

}

m\_multiset.insert(Sum / m\_multiset.size());

}

else {

cout << "Множество пусто!\n";

}

};

void EraseIfInRange(const double BeginValue, const double EndValue) {

auto i = m\_multiset.begin();

while (i != m\_multiset.end()) {

if (\*i >= BeginValue && \*i <= EndValue) {

i = m\_multiset.erase(i);

}

else {

i++;

}

}

};

int size() {

return n;

};

void PlusMinAndMax() {

if (m\_multiset.size() > 0) {

multiset<T> tmp\_multiset = m\_multiset;

T MaxValue, MinValue;

MaxValue = MinValue = \*m\_multiset.begin();

for (auto i = m\_multiset.begin(); i != m\_multiset.end(); i++) {

if (\*i > MaxValue) {

MaxValue = \*i;

}

else if (\*i < MinValue) {

MinValue = \*i;

}

}

m\_multiset = {};

for (auto i = tmp\_multiset.begin(); i != tmp\_multiset.end(); i++) {

T NewValue = \*i + MaxValue + MinValue;

m\_multiset.insert(NewValue);

}

tmp\_multiset = {};

}

}

};

**task3.cpp**

#include <iostream>

#include "List.h"

using namespace std;

int main() {

system("chcp 1251 > NULL");

List<double> a;

a.MakeMultiset(7);

cout << "Множество с дубликатами а:\n";

a.ShowMultiset();

cout << "Добавляем среднее арифмитическое множества a в контейнер\n";

a.PushArithmeticMiddle();

cout << "Множество а:\n";

a.ShowMultiset();

cout << "Удаляем из множества а элементы, значения которых входят в диапазон от 2 до 3\n";

a.EraseIfInRange(2, 3);

cout << "Множество а:\n";

a.ShowMultiset();

cout << "К каждому элементу добавляем сумму минимального и максимального элементов контейнера\n";

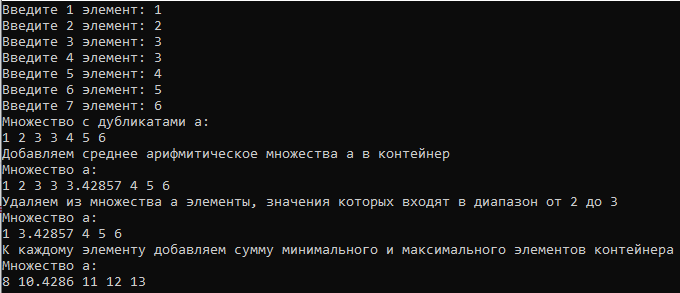
a.PlusMinAndMax();

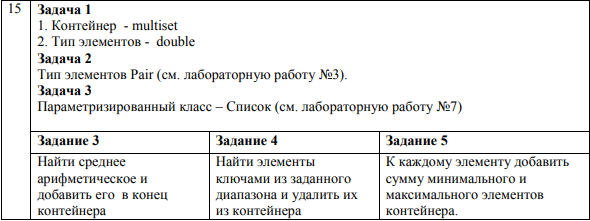
cout << "Множество а:\n";

a.ShowMultiset();

return 0;

}





**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?

multiset – может хранить в себе дубликаты