Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Лабораторная работа №12

«Ассоциативные контейнеры библиотеки STL»

**Семестр 2**

Выполнил работу

Студент группы РИС-22-1Б

Юхновец В.Г.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**Постановка задачи**

**Задача 1.**

1. Создать последовательный 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 последовательный 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 последовательный 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**

Найти среднее арифметическое и добавить его в конец контейнера

**Задание 4**

Найти элементы ключами из заданного диапазона и удалить их из контейнера

**Задание 5**

К каждому элементу добавить сумму минимального и максимального элементов контейнера.

**Ответы на контрольные вопросы**

1. **Что представляет собой ассоциативный контейнер?**

Ассоциативный контейнер это контейнер данных, который хранит элементы в определенном порядке и предоставляет быстрый доступ к элементам по ключу.

1. **Перечислить ассоциативные контейнеры библиотеки STL.**

В STL есть несколько ассоциативных контейнеров, включая map, multimap, set и multiset.

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

В ассоциативных контейнерах доступ к элементам осуществляется по ключу, который может быть использован для поиска и получения значения элемента.

1. **Привести примеры методов, используемых в ассоциативных контейнерах.**

Некоторые методы, которые могут использоваться в ассоциативных контейнерах, включают insert() для добавления элементов, erase() для удаления элементов, find() для поиска элементов по ключу, и count() для подсчета числа элементов с определенным ключом.

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

Контейнер map можно создать с помощью конструктора по умолчанию или с помощью конструктора, который принимает пары ключ-значение.

Например:

std::map<int, std::string> myMap; // создание пустого контейнера map

myMap.insert(std::pair<int, std::string>(1, "one")); // добавление элемента в контейнер map

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

Элементы в контейнере map упорядочены по ключу по возрастанию. Чтобы изменить порядок на обратный, можно создать контейнер с компаратором, который инвертирует порядок сортировки.

Например:

std::map<int, std::string, std::greater<int>> myMap; // создание контейнера map с компаратором

1. **Какие операции определены для контейнера map?**

Операции, определенные для контейнера map, включают добавление и удаление элементов, поиск элементов по ключу, получение размера контейнера, проверку наличия элементов и итерирование по контейнеру.

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

void addElementToMap(std::map<int, std::string>& myMap, int key, std::string value) {

myMap.insert(std::make\_pair(key, value));

}

1. **Написать функцию для добавления элементов в контейнер map с помощью функции операции прямого доступа [].**

Для добавления элементов в контейнер map с помощью функции операции прямого доступа [] необходимо указать ключ и значение элемента. Если ключ уже существует в map, то значение элемента будет обновлено, иначе будет создан новый элемент с указанным ключом и значением.

Например:

int main() {

std::map<std::string, int> myMap;

myMap["one"] = 1;

myMap["two"] = 2;

myMap["three"] = 3;

std::cout << "Map size: " << myMap.size() << std::endl;

std::cout << "Map content:" << std::endl;

for (const auto& [key, value] : myMap) {

std::cout << key << " => " << value << std::endl;

}

return 0; }

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

Для печати контейнера map с помощью итератора можно использовать стандартный цикл for и обращаться к элементам контейнера через итератор.

Например:

void printMap(const std::map<std::string, int>& myMap) {

std::cout << "Map size: " << myMap.size() << std::endl;

std::cout << "Map content:" << std::endl;

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

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

}

}

int main() {

std::map<std::string, int> myMap = {

{"one", 1},

{"two", 2},

{"three", 3}

};

printMap(myMap);

return 0;

}

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

Можно использовать цикл for для обхода всех элементов контейнера и вывода их на экран с помощью операции прямого доступа [].

Пример:

void printMap(std::map<int, std::string>& myMap) {

for (auto const& [key, value] : myMap) {

std::cout << key << ": " << myMap[key] << std::endl;

}

}

Здесь используется структура данных std::pair, чтобы хранить пару ключ-значение, которая извлекается из контейнера с помощью итератора. В качестве альтернативы, можно использовать метод at(), чтобы получать значение элемента по ключу.

1. **Чем отличаются контейнеры map и multimap?**

Контейнеры map и multimap являются ассоциативными контейнерами, которые хранят элементы в отсортированном порядке по ключу. Основное отличие заключается в том, что контейнер map может содержать только уникальные ключи, в то время как в контейнере multimap допускается наличие нескольких элементов с одинаковыми ключами.

1. **Что представляет собой контейнер set?**

Контейнер set также является ассоциативным контейнером и хранит уникальные элементы в отсортированном порядке. В отличие от контейнера map, он хранит только ключи, без значений.

1. **Чем отличаются контейнеры map и set?**

Контейнеры map и set имеют общую особенность - они являются ассоциативными контейнерами и хранят элементы в отсортированном порядке. Однако, контейнер map хранит пары ключ-значение, тогда как контейнер set хранит только ключи.

1. **Каким образом можно создать контейнер set? Привести примеры.**

Контейнер set можно создать следующим образом:

std::set<int> my\_set; // пустой контейнер set

Можно также инициализировать контейнер set с помощью списка инициализации:

std::set<int> my\_set = {1, 2, 3, 4, 5};

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

Элементы в контейнере set по умолчанию хранятся в возрастающем порядке. Чтобы изменить порядок на обратный, можно использовать реверсивный итератор:

std::set<int> my\_set = {1, 2, 3, 4, 5};

for (auto rit = my\_set.rbegin(); rit != my\_set.rend(); ++rit) {

std::cout << \*rit << " ";

}

1. **Какие операции определены для контейнера set?**

Контейнер set предоставляет следующие операции:

Вставка элемента: insert()

Удаление элемента: erase()

Поиск элемента: find()

Получение количества элементов: size()

Проверка на пустоту: empty()

Получение итераторов на первый и последний элементы: begin(), end()

1. **Написать функцию для добавления элементов в контейнер set.**

Можно использовать метод insert(), который принимает значение добавляемого элемента в качестве аргумента. Пример:

void addElementToSet(std::set<int>& mySet, int value) {

mySet.insert(value);

}

1. **Написать функцию для печати контейнера set.**

Можно использовать итераторы и цикл for для обхода всех элементов контейнера и вывода их на экран.

Пример:

void printSet(std::set<int>& mySet) {

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

std::cout << \*it << " ";

}

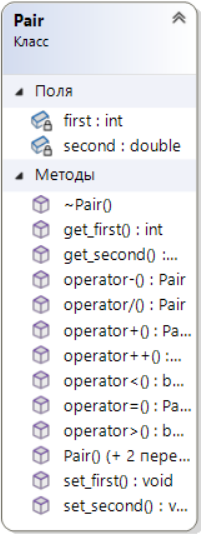
std::cout << std::endl;

}

1. **Чем отличаются контейнеры set и multiset?**

Контейнеры set и multiset отличаются тем, что set содержит только уникальные элементы, тогда как multiset может содержать несколько экземпляров одного и того же элемента. Если в set попытаться вставить уже существующий элемент, то операция вставки не будет выполнена. В multiset можно добавлять дубликаты. Оба контейнера упорядочены по возрастанию по умолчанию.ф

**Диаграммы классов**



%

Рисунок 1 – диаграмма класса Pair

**Описание задач**

**Задача 1**

#pragma once

typedef multiset<double> lst;

void fill\_list(lst& l)

{

int n = 0;

double el = 0;

cout << "Input size: "; cin >> n; cout << endl;

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

l.insert((double)(rand() % 500 - 50) / 10);

}

void show\_list(lst& l)

{

if (l.size() != 0)

{

cout << "multiset: ";

for (auto& i : l)

cout << i << " ";

cout << endl;

}

}

void average(lst& l)

{

if (l.size() != 0)

{

double av = 0;

for (auto& i : l)

av += i;

l.insert(av / l.size());

}

}

void remove(lst& l)

{

if (l.size() != 0)

{

double start, end;

cout << "\nStart position: "; cin >> start;

cout << "End position: "; cin >> end;

for (auto i = l.begin(); i != l.end();)

{

if (start <= \*i && \*i <= end)

i = l.erase(i);

else

++i;

}

}

}

void add\_min\_max(lst& l)

{

if (l.size() != 0)

{

double min = \*l.begin(), max = \*l.begin();

for (auto& i : l)

{

if (i > max)

max = i;

if (i < min)

min = i;

}

lst temp;

for (auto& i : l)

temp.insert(i + max + min);

l = temp;

}

}

void foo\_1()

{

cout << "\t\t\tEXERCISE 1\n\n";

srand(time(0));

lst lst1;

fill\_list(lst1);

cout << "---------------------- [initial multiset] ----------------------\n";

show\_list(lst1);

average(lst1);

cout << "\n--------------------- [multiset + average] ---------------------\n";

show\_list(lst1);

remove(lst1);

cout << "\n------------------- [multiset after deletion] ------------------\n";

show\_list(lst1);

add\_min\_max(lst1);

cout << "\n------------------- [multiset + (min + max)] -------------------\n";

show\_list(lst1);

cout << endl;

system("pause");

}

**Задача 2**

#pragma once

typedef multiset<Pair> lst2;

void fill\_list(lst2& l)

{

int n = 0;

Pair p;

cout << "Input size: "; cin >> n; cout << endl;

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

{

p.set\_first((int)(rand() % 500 - 50) / 10);

p.set\_second((double)(rand() % 500 - 50) / 10);

l.insert(p);

}

}

void show\_list(lst2& l)

{

if (l.size() != 0)

{

cout << "multiset: ";

for (auto& i : l)

cout << i << " ";

cout << endl;

}

}

void average(lst2& l)

{

if (l.size() != 0)

{

int av\_first = 0;

double av\_second = 0;

Pair temp;

for (auto& i : l)

{

temp = i;

av\_first += temp.get\_first();

av\_second += temp.get\_second();

}

av\_first /= l.size();

av\_second /= l.size();

temp.set\_first(av\_first);

temp.set\_second(av\_second);

l.insert(temp);

}

}

void remove(lst2& l)

{

if (l.size() != 0)

{

Pair start, end;

cout << "\n\t[Start position]\n"; cin >> start;

cout << "\t[End position]\n"; cin >> end;

Pair temp;

for (auto i = l.begin(); i != l.end();)

{

temp = \*i;

if ((start.get\_first() <= temp.get\_first() && start.get\_second() <= temp.get\_second()) && (temp.get\_first() <= end.get\_first() && temp.get\_second() <= end.get\_second()))

i = l.erase(i);

else

++i;

}

}

}

void add\_min\_max(lst2& l)

{

if (l.size() != 0)

{

Pair p\_min, p\_max;

Pair tempP = \*l.begin();

p\_max.set\_first(tempP.get\_first());

p\_max.set\_second(tempP.get\_second());

p\_min.set\_first(tempP.get\_first());

p\_min.set\_second(tempP.get\_second());

for (auto& i : l)

{

tempP = i;

if ((p\_max.get\_first() + p\_max.get\_second()) < (tempP.get\_first() + tempP.get\_second()))

{

p\_max.set\_first(tempP.get\_first());

p\_max.set\_second(tempP.get\_second());

}

else if ((p\_min.get\_first() + p\_min.get\_second()) > (tempP.get\_first() + tempP.get\_second()))

{

p\_min.set\_first(tempP.get\_first());

p\_min.set\_second(tempP.get\_second());

}

}

lst2 tempM;

Pair tempP2;

for (auto& i : l)

{

tempP = i;

tempP2.set\_first(tempP.get\_first() + p\_max.get\_first() + p\_min.get\_first());

tempP2.set\_second(tempP.get\_second() + p\_max.get\_second() + p\_min.get\_second());

tempM.insert(tempP2);

}

l = tempM;

}

}

void foo\_2()

{

system("cls");

cout << "\t\t\tEXERCISE 2\n\n";

srand(time(0));

lst2 lst2;

fill\_list(lst2);

cout << "---------------------- [initial multiset] ----------------------\n";

show\_list(lst2);

average(lst2);

cout << "\n--------------------- [multiset + average] ---------------------\n";

show\_list(lst2);

remove(lst2);

cout << "\n------------------- [multiset after deletion] ------------------\n";

show\_list(lst2);

add\_min\_max(lst2);

cout << "\n------------------- [multiset + (min + max)] -------------------\n";

show\_list(lst2);

cout << endl;

system("pause");

}

**Задача 3**

#pragma once

#include <iostream>

#include <set>

#include <string>

using namespace std;

template <typename T>

class List\_3

{

private:

multiset<T> st;

int len;

public:

List\_3();

List\_3(int n);

~List\_3();

void make\_list();

void show();

void average();

void remove();

void add\_min\_max();

int get\_size() { return size; }

List\_3& operator=(const List\_3& object);

};

template <typename T>

List\_3<T>::List\_3()

{

len = 0;

}

template<typename T>

List\_3<T>::List\_3(int n)

{

T a;

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

{

cin >> a;

st.push();

}

}

template <typename T>

List\_3<T>::~List\_3()

{

}

template<typename T>

void List\_3<T>::make\_list()

{

int n;

cout << "Input size: "; cin >> n; cout << endl;

T a;

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

{

a.set\_first((int)(rand() % 500 - 50) / 10);

a.set\_second((double)(rand() % 500 - 50) / 10);

st.insert(a);

}

len = st.size();

}

template<typename T>

void List\_3<T>::show()

{

if (len != 0)

{

cout << "multiset: ";

for (auto& i : st)

cout << i << " ";

cout << endl;

}

}

template<typename T>

void List\_3<T>::average()

{

if (len != 0)

{

int av\_first = 0;

double av\_second = 0;

Pair temp;

for (auto& i : st)

{

temp = i;

av\_first += temp.get\_first();

av\_second += temp.get\_second();

}

av\_first /= st.size();

av\_second /= st.size();

temp.set\_first(av\_first);

temp.set\_second(av\_second);

st.insert(temp);

}

}

template<typename T>

void List\_3<T>::remove()

{

if (len != 0)

{

Pair start, end;

cout << "\n\t[Start position]\n"; cin >> start;

cout << "\t[End position]\n"; cin >> end;

Pair temp;

for (auto i = st.begin(); i != st.end();)

{

temp = \*i;

if ((start.get\_first() <= temp.get\_first() && start.get\_second() <= temp.get\_second()) && (temp.get\_first() <= end.get\_first() && temp.get\_second() <= end.get\_second()))

i = st.erase(i);

else

++i;

}

}

}

template<typename T>

void List\_3<T>::add\_min\_max()

{

if (len != 0)

{

Pair p\_min, p\_max;

Pair tempP = \*st.begin();

p\_max.set\_first(tempP.get\_first());

p\_max.set\_second(tempP.get\_second());

p\_min.set\_first(tempP.get\_first());

p\_min.set\_second(tempP.get\_second());

for (auto& i : st)

{

tempP = i;

if ((p\_max.get\_first() + p\_max.get\_second()) < (tempP.get\_first() + tempP.get\_second()))

{

p\_max.set\_first(tempP.get\_first());

p\_max.set\_second(tempP.get\_second());

}

else if ((p\_min.get\_first() + p\_min.get\_second()) > (tempP.get\_first() + tempP.get\_second()))

{

p\_min.set\_first(tempP.get\_first());

p\_min.set\_second(tempP.get\_second());

}

}

lst2 tempM;

Pair tempP2;

for (auto& i : st)

{

tempP = i;

tempP2.set\_first(tempP.get\_first() + p\_max.get\_first() + p\_min.get\_first());

tempP2.set\_second(tempP.get\_second() + p\_max.get\_second() + p\_min.get\_second());

tempM.insert(tempP2);

}

st = tempM;

}

}

template <typename T>

List\_3<T>& List\_3<T>::operator=(const List\_3& object)

{

if (this != &object)

if (this->head != nullptr)

{

this->~List\_3();

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

this->push\_back(object[i]);

}

else

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

this->push\_back(object[i]);

return \*this;

}

void foo\_3()

{

system("cls");

cout << "\t\t\tEXERCISE 3\n\n";

srand(time(0));

List\_3<Pair> lst5;

lst5.make\_list();

cout << "---------------------- [initial multiset] ----------------------\n";

lst5.show();

lst5.average();

cout << "\n--------------------- [multiset + average] ---------------------\n";

lst5.show();

lst5.remove();

cout << "\n------------------- [multiset after deletion] ------------------\n";

lst5.show();

lst5.add\_min\_max();

cout << "\n------------------- [multiset + (min + max)] -------------------\n";

lst5.show();

cout << endl;

system("pause");

}

**Результаты работы программы**

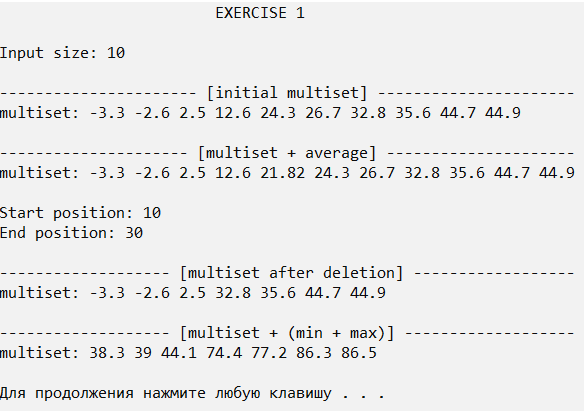


Рисунок 2 – задача 1

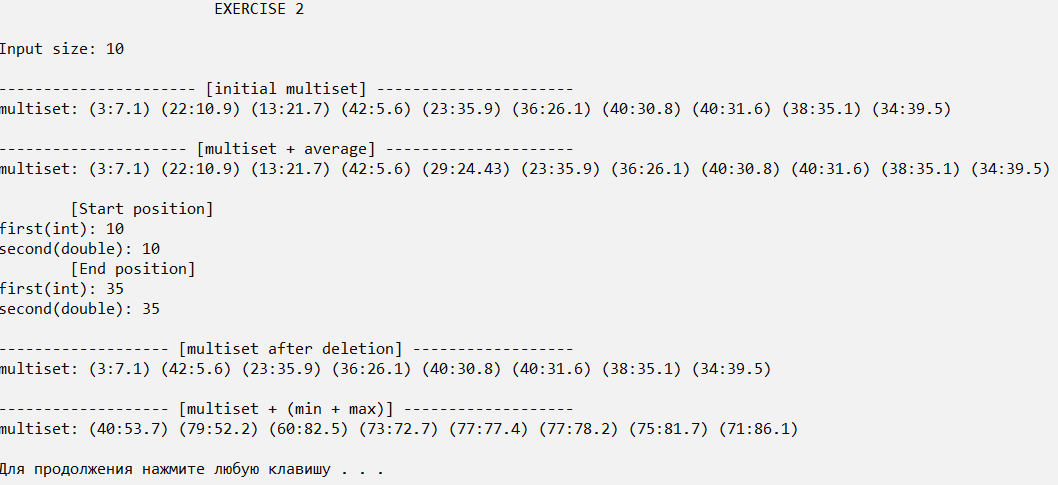


Рисунок 3 – задача 2

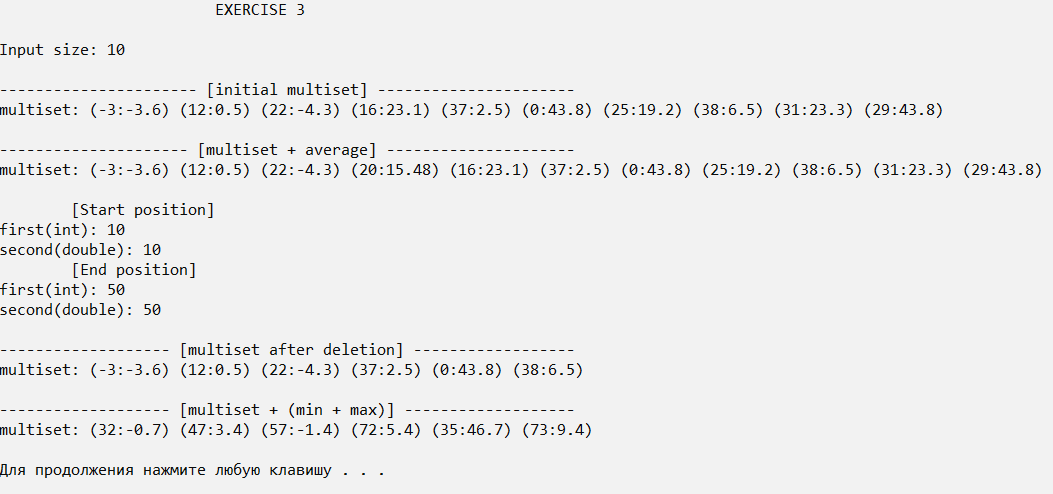


Рисунок 4 – задача 3