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**

**о работе по информатике**

Семестр: 2

На тему: Лабораторная работа №12: «Ассоциативные контейнеры библиотеки STL».

**Вариант 13**

Выполнил студент ИВТ-22-2б:

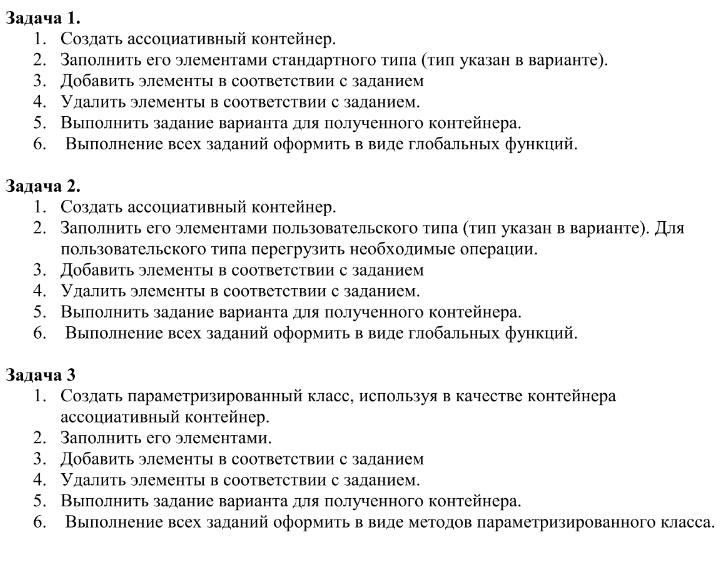
Коняев Александр Сергеевич

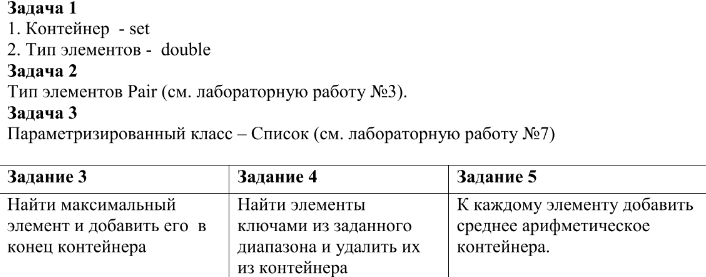
Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

Пермь 2023

**Постановка задачи**





**Анализ задачи**

Класс Pair

* Конструктор без параметров
* Конструктор с параметрами
* Конструктор копирования
* Метод get\_first
* Метод set\_first
* Метод get\_second
* Метод set\_second
* Перегруженные операторы: =, <, >, --, <<, >>

Класс LinkedList

* Конструктор с параметрами
* Конструктор без параметров
* Конструктор копирования
* Метод clear
* Метод push\_back
* Метод push\_front
* Метод pop\_back
* Метод pop\_front
* Метод get\_At
* Метод insert
* Метод erase
* Метод print
* Перегруженные операторы: =, [], +, <<, >>

STL библиотеки: set

**UML – диаграмма**

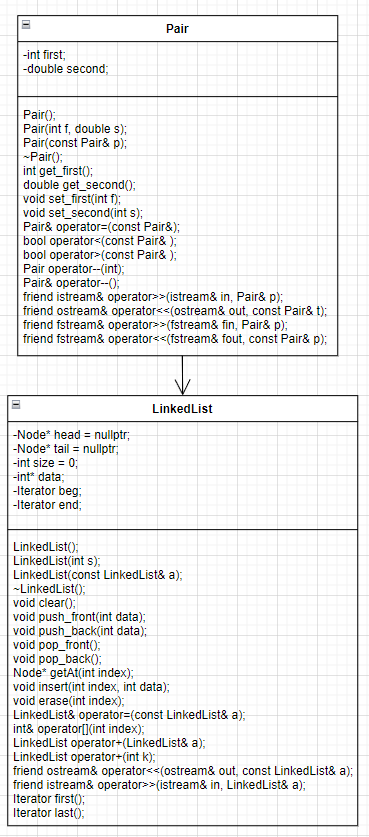


Рисунок 1 – UML-диаграмма.

**Код программы.**

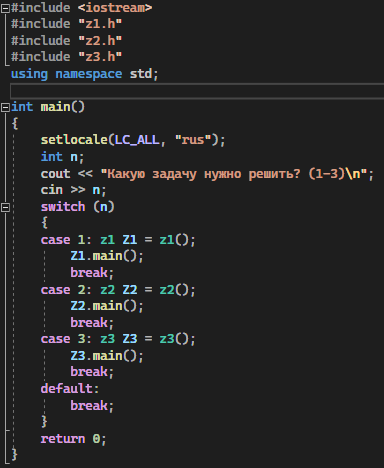


Рисунок 2 – Функция main

**Вывод программы.**

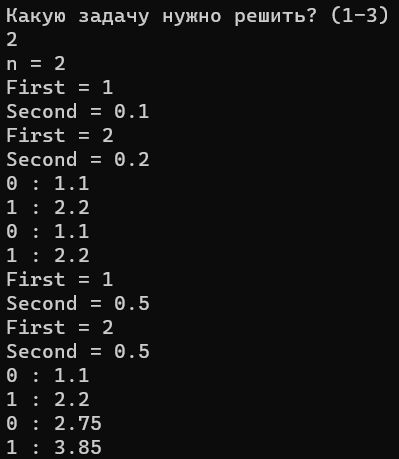
****

Рисунок 3 – Вывод программы

**Ответы на вопросы**

1. Ассоциативный контейнер - это контейнер, который хранит элементы в отсортированном порядке и позволяет быстро получать доступ к элементам по ключу.

2. Ассоциативные контейнеры библиотеки STL:

- map

- multimap

- set

- multiset

3. Доступ к элементам ассоциативного контейнера осуществляется по ключу. Для этого используется оператор или методы at() и find().

4. Примеры методов, используемых в ассоциативных контейнерах:

- insert() - вставка элемента в контейнер

- erase() - удаление элемента из контейнера

- size() - количество элементов в контейнере

- empty() - проверка на пустоту контейнера

- begin() и end() - итераторы на начало и конец контейнера

- clear() - удаление всех элементов из контейнера

5. Контейнер map создается следующим образом:

#include <map>

#include <string>

std::map<int, std::string> myMap; // создание пустого контейнера map, где ключи - целые числа, значения - строки

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

myMap[2] = "two"; // добавление элемента в контейнер с помощью оператора []

6. Элементы в контейнере map упорядочены по ключу по возрастанию. Чтобы изменить порядок на обратный, можно передать в конструктор контейнера map дополнительный аргумент - объект класса сравнения, который будет сравнивать ключи в обратном порядке. Например:

```

#include <map>

#include <string>

struct Compare

{

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

{

return a > b;

}

};

std::map<int, std::string, Compare> myMap; // создание контейнера map с порядком ключей по убыванию

```

7. Операции, определенные для контейнера map:

- insert() - вставка элемента в контейнер

- erase() - удаление элемента из контейнера

- size() - количество элементов в контейнере

- empty() - проверка на пустоту контейнера

- begin() и end() - итераторы на начало и конец контейнера

- clear() - удаление всех элементов из контейнера

- find() - поиск элемента по ключу

- count() - подсчет количества элементов с заданным ключом

- operator[] - доступ к элементу по ключу

8. Пример функции для добавления элементов в контейнер map с помощью функции make\_pair():

```

#include <map>

#include <string>

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

{

myMap.insert(std::make\_pair(1, "one"));

myMap.insert(std::make\_pair(2, "two"));

myMap.insert(std::make\_pair(3, "three"));

}

```

9. Пример функции для добавления элементов в контейнер map с помощью операции прямого доступа []:

```

#include <map>

#include <string>

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

{

myMap[1] = "one";

myMap[2] = "two";

myMap[3] = "three";

}

```

10. Пример функции для печати контейнера map с помощью итератора:

```

#include <map>

#include <string>

#include <iostream>

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

{

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

{

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

}

}

```

11. Пример функции для печати контейнера map с помощью функции операции прямого доступа :

#include <map>

#include <string>

#include <iostream>

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

{

for (int i = 1; i <= myMap.size(); ++i)

{

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

}

}

12. Контейнер map хранит уникальные элементы, каждый из которых связан с уникальным ключом, а контейнер multimap позволяет хранить несколько элементов с одинаковым ключом.

13. Контейнер set - это контейнер, который хранит уникальные элементы в отсортированном порядке. Каждый элемент может быть добавлен в контейнер только один раз.

14. Основное отличие контейнеров map и set заключается в том, что контейнер map хранит пары ключ-значение, а контейнер set - только значения. Кроме того, в контейнере map элементы упорядочены по ключу, а в контейнере set - по значению.

15. Контейнер set создается следующим образом:

#include <set>

#include <string>

std::set<std::string> mySet; // создание пустого контейнера set со значениями типа std::string

mySet.insert("one"); // добавление элемента в контейнер

mySet.insert("two");

mySet.insert("three");

16. Элементы в контейнере set упорядочены по значению по возрастанию. Чтобы изменить порядок на обратный, можно передать в конструктор контейнера set дополнительный аргумент - объект класса сравнения, который будет сравнивать значения в обратном порядке. Например:

#include <set>

#include <string>

struct Compare

{

bool operator()(const std::string& a, const std::string& b) const

{

return a > b;

}

};

std::set<std::string, Compare> mySet; // создание контейнера set с порядком значений по убыванию

17. Операции, определенные для контейнера set:

- insert() - вставка элемента в контейнер

- erase() - удаление элемента из контейнера

- size() - количество элементов в контейнере

- empty() - проверка на пустоту контейнера

- begin() и end() - итераторы на начало и конец контейнера

- clear() - удаление всех элементов из контейнера

- find() - поиск элемента по значению

- count() - подсчет количества элементов с заданным значением

18. Пример функции для добавления элементов в контейнер set:

#include <set>

#include <iostream>

void addElements(std::set<int>& mySet, int n)

{

for (int i = 1; i <= n; ++i)

{

mySet.insert(i);

}

}

19. Пример функции для печати контейнера set:

#include <set>

#include <iostream>

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

{

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

{

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

}

std::cout << std::endl;

}

20. Основное отличие контейнеров set и multiset заключается в том, что контейнер set хранит только уникальные элементы, а контейнер multiset может хранить несколько элементов с одинаковым значением.