Министерство образования и науки РФ

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

Лабораторная работа № 12

Тема: «Ассоциативные контейнеры библиотеки STL»

Выполнил: студент группы РИС-22-2б

Мизёв В.А. ф

Проверил: доцент кафедры ИТАС

Полякова О.А.

г. Пермь – 2024

*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.

*Задание вариант 1:*

|  |  |  |
| --- | --- | --- |
| **Задача 1**   1. Контейнер – multimap 2. Тип элементов – double   **Задача 2**  Тип элементов Time (см. лабораторную работу №3).  **Задача 3**  Параметризированный класс – Вектор(см. лабораторную работу № 7) | | |
| **Задание 3** | **Задание 4** | **Задание 5** |
| Найти максимальный элемент и добавить его в начало контейнера | Найти минимальный элемент и удалить его из контейнера | К каждому элементу добавить среднее арифметическое контейнера |

*Анализ задачи:*

В виду специфики заданий, обернуть каждое задание в отдельную функцию с соответствующим названием.

Анализируя первую задачу необходимо продумать элементы, которые возможно использовать повторно. Основываясь на задании необходимо заполнять контейнер некими данными, для этого реализуем основную функцию для работы с этим. void fill\_array(), который принимает в качестве аргумента контейнер multimap, шаблонный тип, кол-во элементов которое необходимо сгенерировать и лямбда функция для универсальности добавления любых значений. Остальные функции следует из заданий 3,4,5 это:

push\_front\_max\_element() – ищет максимальный элемент в векторе и вставляет его в начало.

pop\_min\_element() – ищет минимальный элемент и удаляет его из вектора

add\_average\_elements() – добавляет каждому значению среднее арифметическое контейнера

print() – выводит содержимое контейнера.

*Код программы:*

Прикреплён в приложении 1.

*UML-диаграмма класса:*

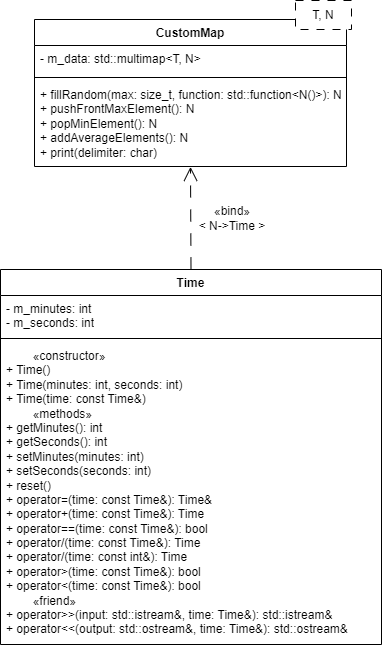


Рисунок 1 - UML-диаграмма классов

*Скриншот работы программы:*

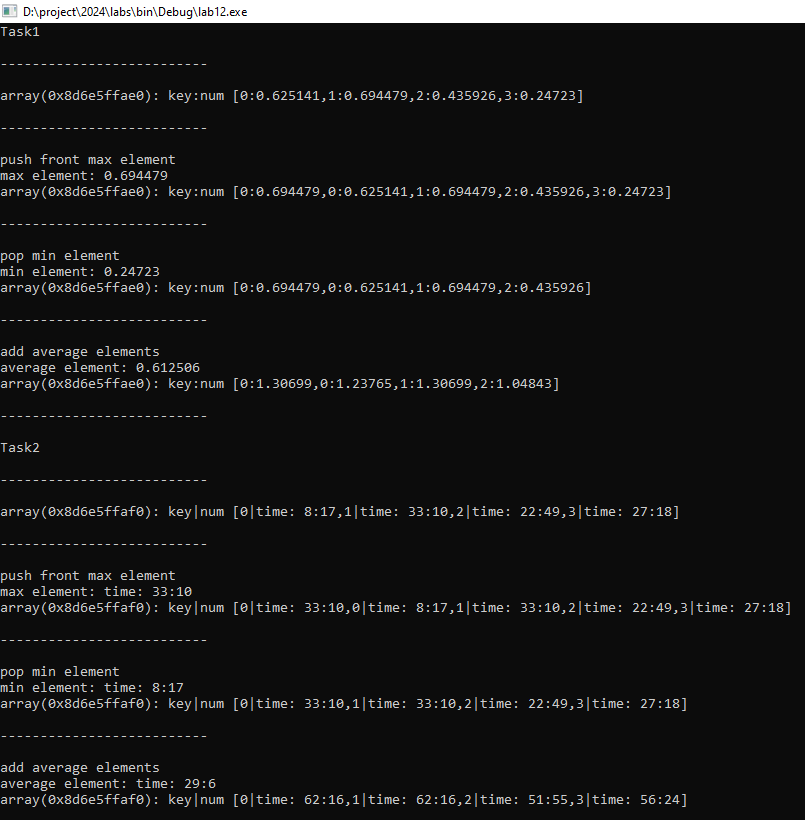


Рисунок 2 – Результат выполнения работы программы часть 1

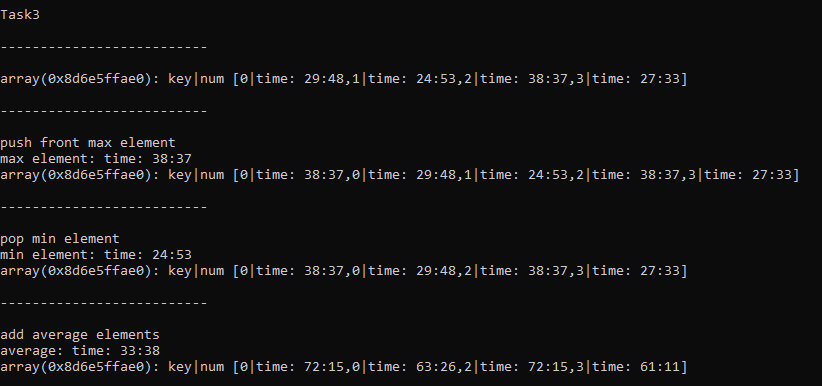


Рисунок 3 – Результат выполнения программы часть 2

*Контрольные вопросы:*

1. Что представляет собой ассоциативный контейнер?

Ассоциативный контейнер — это коллекция объектов, которые могут быть быстро найдены по ключу. Ассоциативные контейнеры обеспечивают ассоциативный доступ, что позволяет получать доступ к данным по ключу, а не по индексу, как в последовательных контейнерах.

2. Перечислить ассоциативные контейнеры библиотеки STL.

Основные ассоциативные контейнеры STL:

- std::map

- std::multimap

- std::set

- std::multiset

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

В ассоциативных контейнерах доступ к элементам можно получить с помощью ключа. Например, с помощью метода operator[] для контейнера map.

4. Привести примеры методов, используемых в ассоциативных контейнерах.

- insert(): для добавления элементов.

- find(): для поиска элемента по ключу.

- erase(): для удаления элемента.

- clear(): для очистки контейнера.

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

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

6. Каким образом упорядочены элементы в контейнере mар по умолчанию? Как изменить порядок на обратный?

В контейнере map элементы упорядочены по возрастанию ключей. Чтобы изменить порядок на обратный, можно использовать std::greater<>() как компаратор:

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

7. Какие операции определены для контейнера map?

Операции для map включают:

- insert()

- find()

- erase()

- clear()

- operator[]

8. Написать функцию для добавления элементов в контейнер map с помощью функции make pair().

void addToMapWithPair(std::map<int, std::string>& m, int key, const std::string& value) {

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

}

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

void addToMapWithBrackets(std::map<int, std::string>& m, int key, const std::string& value) {

m[key] = value;

}

10. Написать функцию для печати контейнера maр с помощью итератора.

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

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

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

}

}

11. Написать функцию для печати контейнера mаp с помощью функции операции

прямого доступа [].

#include <iostream>

#include <map>

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

for (const auto& pair : myMap) {

std::cout << pair.first << ": " << myMap[pair.first] << std::endl;

}

}

int main() {

std::map<int, std::string> myMap = {{1, "one"}, {2, "two"}, {3, "three"}};

printMap(myMap);

return 0;

}

12. Чем отличаются контейнеры map и multimap?

- map содержит уникальные ключи, без дубликатов.

- multimap позволяет иметь несколько значений для одного ключа, то есть дубликаты ключей допустимы.

13. Что представляет собой контейнер set?

set — это ассоциативный контейнер, который хранит уникальные элементы в отсортированном порядке. Элементы не могут быть дубликатами.

14. Чем отличаются контейнеры map и set?

- map хранит пары "ключ-значение" и является ассоциативным массивом.

- set хранит только уникальные значения, без пар.

15. Каким образом можно создать контейнер set? Привести примеры.

#include <iostream>

#include <set>

int main() {

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

return 0;

}

16. Каким образом упорядочены элементы в контейнере set по умолчанию? Как

изменить порядок на обратный?

- Элементы в set по умолчанию упорядочены по возрастанию.

- Чтобы изменить порядок на обратный, можно использовать std::greater при объявлении.

17. Какие операции определены для контейнера set?

Операции включают:

- Вставка (insert)

- Удаление (erase)

- Поиск (find)

- Проверка на наличие элемента (count, contains)

- Итерация по элементам

18. Написать функцию для добавления элементов в контейнер set.

#include <set>

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

mySet.insert(value);

}

19. Написать функцию для печати контейнера set.

#include <iostream>

#include <set>

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

for (const auto& el : mySet) {

std::cout << el << " ";

}

std::cout << std::endl;

}

20. Чем отличаются контейнеры set и multiset?

- set хранит уникальные элементы, без дубликатов.

- multiset позволяет хранить несколько экземпляров одного и того же элемента.

**ПРИЛОЖЕНИЕ 1**

lab12.cpp

//-------------------------------------

#include <cmath>

#include <ctime>

//-------------------------------------

void task1();

void task2();

void task3();

//-------------------------------------

int main() {

srand(time(0));

task1();

task2();

task3();

return 0;

}

//-------------------------------------

lab12\_tasks.cpp

//-------------------------------------

#include <iostream>

#include <map>

#include "lab12\_time.hpp"

#include "lab12\_task1.hpp"

#include "lab12\_task3.hpp"

//-------------------------------------

#define IDENT\_PRINT printf("\n--------------------------\n\n")

//-------------------------------------

void task1() {

std::cout << "Task1\n";

IDENT\_PRINT;

std::multimap<int, double> data;

Task1::fill\_array<int, double>(data, 4, []{ return (double)std::rand() / (double)RAND\_MAX; });

Task1::print(data);

IDENT\_PRINT;

std::cout << "push front max element\n";

double max = Task1::push\_front\_max\_element(data);

std::cout << "max element: " << max << std::endl;

Task1::print(data);

IDENT\_PRINT;

std::cout << "pop min element\n";

double min = Task1::pop\_min\_element<int, double>(data);

std::cout << "min element: " << min << std::endl;

Task1::print(data);

IDENT\_PRINT;

std::cout << "add average elements\n";

double average = Task1::add\_average\_elements<int, double, double>(data);

std::cout << "average element: " << average << std::endl;

Task1::print(data);

IDENT\_PRINT;

}

//-------------------------------------

void task2() {

std::cout << "Task2\n";

IDENT\_PRINT;

std::multimap<int, Time> data;

Task1::fill\_array<int, Time>(data, 4, []{ return Time(std::rand()%40, std::rand() % MAX\_SECONDS); });

Task1::print(data, '|');

IDENT\_PRINT;

std::cout << "push front max element\n";

Time max = Task1::push\_front\_max\_element<int, Time>(data);

std::cout << "max element: " << max << std::endl;

Task1::print(data, '|');

IDENT\_PRINT;

std::cout << "pop min element\n";

Time min = Task1::pop\_min\_element(data);

std::cout << "min element: " << min << std::endl;

Task1::print(data, '|');

IDENT\_PRINT;

std::cout << "add average elements\n";

Time average = Task1::add\_average\_elements<int, Time, int>(data);

std::cout << "average element: " << average << std::endl;

Task1::print(data, '|');

IDENT\_PRINT;

}

//-------------------------------------

void task3() {

std::cout << "Task3\n";

CustomMap<int, Time> data;

IDENT\_PRINT;

data.fillRandom(4, []{ return Time(std::rand()%40, std::rand() % MAX\_SECONDS); });

data.print('|');

IDENT\_PRINT;

Time buff;

std::cout << "push front max element\n";

buff = data.pushFrontMaxElement();

std::cout << "max element: " << buff << std::endl;

data.print('|');

IDENT\_PRINT;

std::cout << "pop min element\n";

buff = data.popMinElement();

std::cout << "min element: " << buff << std::endl;

data.print('|');

IDENT\_PRINT;

std::cout << "add average elements\n";

buff = data.addAverageElements<int>();

std::cout << "average: " << buff << std::endl;

data.print('|');

}

//-------------------------------------

lab12\_task1.hpp

//-------------------------------------

#ifndef LAB12\_TASK1\_HPP\_INCLUDED

#define LAB12\_TASK1\_HPP\_INCLUDED

//-------------------------------------

#include <map>

#include <functional>

#include <numeric>

#include <cmath>

//-------------------------------------

namespace Task1 {

//-------------------------------------

template<class T, class N>

void fill\_array(std::multimap<T, N>& data, size\_t max, std::function<N(void)> function) {

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

data.insert({ i, function() });

}

//-------------------------------------

template<class T, class N>

N push\_front\_max\_element(std::multimap<T, N>& data) {

auto it = std::max\_element(data.begin(), data.end(),

[](std::pair<T, N> it0, std::pair<T, N> it1){ return it0.second < it1.second; });

N buff = it->second;

data.insert(data.begin(), { data.begin()->first, it->second });

return buff;

}

//-------------------------------------

template<class T, class N>

N pop\_min\_element(std::multimap<T, N>& data) {

auto it = std::min\_element(data.begin(), data.end(),

[](std::pair<T, N> it0, std::pair<T, N> it1){ return it0.second < it1.second; });

N buff = it->second;

data.erase(it);

return buff;

}

//-------------------------------------

template<class T, class N, class Custom>

N add\_average\_elements(std::multimap<T, N>& data) {

N average = std::accumulate(data.begin(), data.end(), N(),

[](N buff, std::pair<T, N> it){ return buff + it.second; });

average = average / (Custom)data.size();

for(auto& it : data)

it.second = it.second + average;

return average;

}

//-------------------------------------

template<class T, class N>

void print(std::multimap<T, N>& data, char delimiter = ':') {

std::cout << "array(" << &data << "): key";

std::cout << delimiter << "num [";

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

std::cout << it->first << delimiter << it->second;

if(it != (--data.end()))

std::cout << ",";

}

std::cout << "]\n";

}

//-------------------------------------

};

//-------------------------------------

#endif // LAB12\_TASK1\_HPP\_INCLUDED

//-------------------------------------

lab12\_task3.hpp

//-------------------------------------

#ifndef LAB12\_TASK3\_HPP\_INCLUDED

#define LAB12\_TASK3\_HPP\_INCLUDED

//-------------------------------------

#include "lab12\_task1.hpp"

//-------------------------------------

template<class T, class N>

class CustomMap {

private:

typedef std::multimap<T, N> data\_t;

public:

//-------------------------------------

~CustomMap() = default;

//-------------------------------------

/\*\*

\* example Vector::fillRandom<double>(10, []{ return (double)std::rand() / (double)(RAND\_MAX / 3.0d); });

\*/

void fillRandom(size\_t max, std::function<N()> function) {

Task1::fill\_array<T, N>(m\_data, max, function);

}

//-------------------------------------

N pushFrontMaxElement() {

return Task1::push\_front\_max\_element<T, N>(m\_data);

}

//-------------------------------------

N popMinElement() {

return Task1::pop\_min\_element<T, N>(m\_data);

}

//-------------------------------------

template <class Custom>

N addAverageElements() {

return Task1::add\_average\_elements<T, N, Custom>(m\_data);

}

//-------------------------------------

void print(char delimiter = ':') {

Task1::print<T, N>(m\_data, delimiter);

}

//-------------------------------------

private:

data\_t m\_data;

};

//-------------------------------------

#endif // LAB12\_TASK3\_HPP\_INCLUDED

//-------------------------------------