Министерство образования и науки РФ

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

Лабораторная работа № 13

Тема: «Стандартные обобщённые алгоритмы 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 последовательный контейнер.

2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.

3. Заменить элементы в соответствии с заданием (использовать алгоритмы replace\_if(), replace\_copy(), replace\_copy\_if(), fill()).

4. Удалить элементы в соответствии с заданием (использовать алгоритмы remove(), remove if() , remove copy if() , remove copy() )

5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort ()).

6. Найти в контейнере заданный элемент (использовать алгоритмы find () find if(), count(), count if()).,

7. Выполнить задание варианта для полученного контейнера (использовать алгоритм for each ()).

8. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

*Задача 2.*

1. Создать адаптер контейнера.

2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.

3. Заменить элементы в соответствии с заданием (использовать алгоритмы replace if(), replace copy(), replace copy if(), fill()).

4. Удалить элементы в соответствии с заданием (использовать алгоритмы remove(), remove if(), remove copy if(), remove copy())

5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort () ).

6. Найти в контейнере элемент с заданным ключевым полем (использовать алгоритмы find(), find if(), count(), count if()).

7. Выполнить задание варианта для полученного контейнера (использовать алгоритм for each()).

8. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

*Задача 3*

1. Создать ассоциативный контейнер.

2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.

3. Заменить элементы в соответствии с заданием (использовать алгоритмы replace if(), replace copy(), replace copy if(), fill() ).

4. Удалить элементы в соответствии с заданием (использовать алгоритмы remove() , remove if(), remove copy if(), remove copy())

5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort ()).

6. Найти в контейнере элемент с заданным ключевым полем (использовать алгоритмы find(), find if() , count(), count if()).

7. Выполнить задание варианта для полученного контейнера (использовать алгоритм for each()).

8. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

*Задание вариант 1:*

|  |  |  |
| --- | --- | --- |
| **Задача 1**   1. Контейнер – вектор 2. Тип элементов – Time(см. лабораторную работу № 3).   **Задача 2**  Адаптер контейнера – стек.  **Задача 3**  Ассоциативный контейнер – множество | | |
| **Задание 3** | **Задание 4** | **Задание 5** |
| Заменить максимальный элемент на заданное значение | 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, 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:

replace\_max\_element() – заменяет максимальное значение из контейнера на заданное.

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:*

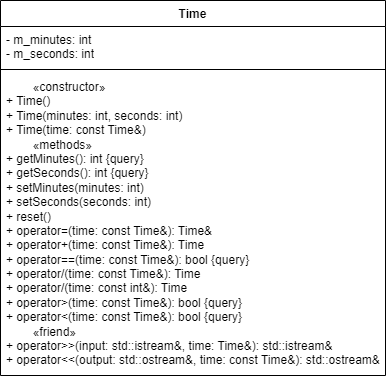


Рисунок 1 - UML-диаграмма классов

*Скриншот работы программы:*

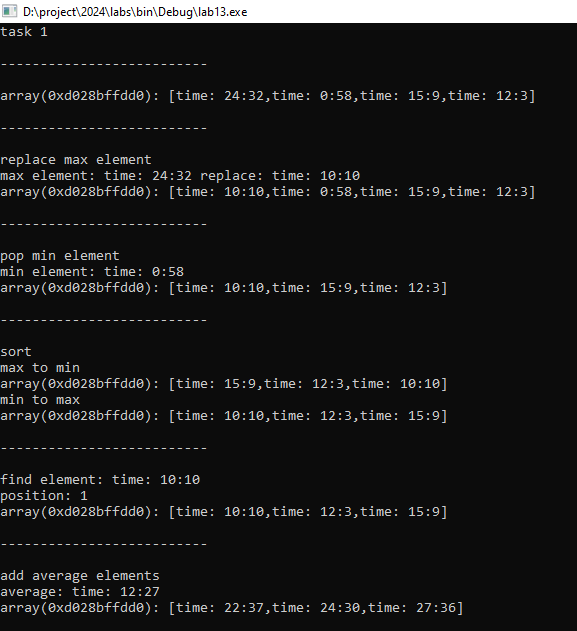


Рисунок 2 – Результат выполнения работы программы часть 1

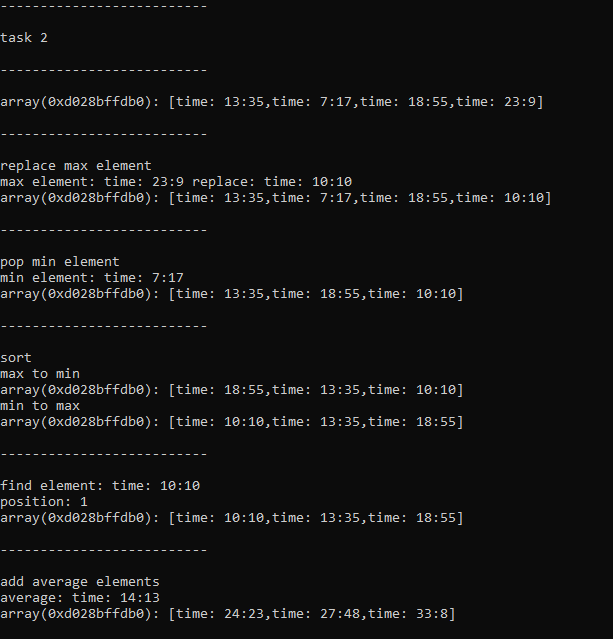


Рисунок 3 – Результат выполнения программы часть 2

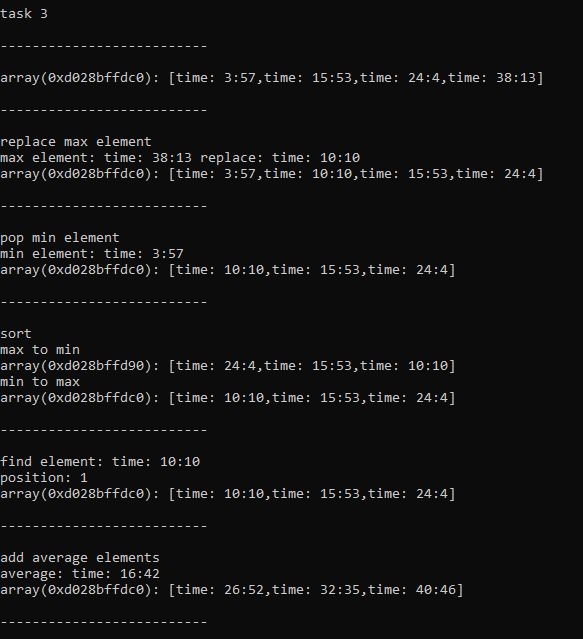


Рисунок 4 - Результат выполнения программы часть 3

**ПРИЛОЖЕНИЕ 1**

lab13.cpp

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

#include <iostream>

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

void task1();

void task2();

void task3();

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

int main() {

srand(time(0));

task1();

task2();

task3();

return 0;

}

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

lab13\_tasks.cpp

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

#include <iostream>

#include <map>

#include "lab13\_time.hpp"

#include "lab13\_task1.hpp"

#include "lab13\_task2.hpp"

#include "lab13\_task3.hpp"

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

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

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

void task1() {

std::cout << "task 1\n";

IDENT\_PRINT;

std::vector<Time> data;

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

Task1::print(data);

IDENT\_PRINT;

Time buff;

Time replace(10, 10);

std::cout << "replace max element\n";

buff = Task1::replace\_max\_element<Time>(data, replace);

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

Task1::print(data);

IDENT\_PRINT;

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

buff = Task1::pop\_min\_element(data);

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

Task1::print(data);

IDENT\_PRINT;

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

std::cout << "max to min\n";

std::sort(data.begin(), data.end(), [&](Time t0, Time t1){ return t0 > t1; });

Task1::print(data);

std::cout << "min to max\n";

std::sort(data.begin(), data.end(), [&](Time t0, Time t1){ return t0 < t1; });

Task1::print(data);

IDENT\_PRINT;

Time find(10, 10);

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

auto pos = std::find(data.begin(), data.end(), find);

if(pos != data.end())

std::cout << "position: " << (pos - data.begin()) + 1 << std::endl;

else

std::cout << "not found!\n";

Task1::print(data);

IDENT\_PRINT;

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

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

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

Task1::print(data);

IDENT\_PRINT;

}

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

void task2() {

std::cout << "task 2\n";

IDENT\_PRINT;

std::stack<Time> data;

Task2::fill\_array<Time>(data, 4, []{ return Time(std::rand()%40, std::rand() % MAX\_SECONDS); });

Task2::print(data);

IDENT\_PRINT;

Time buff;

Time replace(10, 10);

std::cout << "replace max element\n";

buff = Task2::replace\_max\_element<Time>(data, replace);

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

Task2::print(data);

IDENT\_PRINT;

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

buff = Task2::pop\_min\_element<Time>(data);

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

Task2::print(data);

IDENT\_PRINT;

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

std::cout << "max to min\n";

Time \*end = &data.top() + 1,

\*begin = end - data.size();

std::sort(begin, end, [&](Time& num0, Time& num1){ return num0 > num1; });

Task2::print(data);

std::cout << "min to max\n";

std::sort(begin, end, [&](Time& num0, Time& num1){ return num0 < num1; });

Task2::print(data);

IDENT\_PRINT;

Time find(10, 10);

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

auto pos = std::find(begin, end, find);

if(pos != end)

std::cout << "position: " << (pos - begin) + 1 << std::endl;

else

std::cout << "not found!\n";

Task2::print(data);

IDENT\_PRINT;

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

buff = Task2::add\_average\_elements<Time, int>(data);

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

Task2::print(data);

IDENT\_PRINT;

}

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

void task3() {

std::cout << "task 3\n";

IDENT\_PRINT;

std::set<Time> data;

Task3::fill\_array<Time>(data, 4, []{ return Time(std::rand()%40, std::rand() % MAX\_SECONDS); });

Task3::print<Time>(data);

IDENT\_PRINT;

Time buff;

Time replace(10, 10);

std::cout << "replace max element\n";

buff = Task3::replace\_max\_element<Time>(data, replace);

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

Task3::print<Time>(data);

IDENT\_PRINT;

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

buff = Task3::pop\_min\_element(data);

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

Task3::print(data);

IDENT\_PRINT;

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

std::cout << "max to min\n";

std::vector<Time> buff\_arr;

for(auto it : data)

buff\_arr.push\_back(it);

std::sort(buff\_arr.begin(), buff\_arr.end(), [&](Time t0, Time t1){ return t0 > t1; });

Task1::print<Time>(buff\_arr);

std::cout << "min to max\n";

Task3::print<Time>(data);

IDENT\_PRINT;

Time find(10, 10);

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

auto pos = std::find(data.begin(), data.end(), find);

if(pos != data.end())

std::cout << "position: " << std::distance(data.begin(), pos) + 1 << std::endl;

else

std::cout << "not found!\n";

Task3::print<Time>(data);

IDENT\_PRINT;

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

buff = Task3::add\_average\_elements<Time, int>(data);

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

Task3::print<Time>(data);

IDENT\_PRINT;

}

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

lab13\_task1.hpp

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

#ifndef LAB13\_TASK1\_HPP\_INCLUDED

#define LAB13\_TASK1\_HPP\_INCLUDED

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

#include <vector>

#include <numeric>

#include <functional>

#include <cmath>

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

namespace Task1 {

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

template<class T>

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

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

data.push\_back(function());

}

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

template<class T>

T replace\_max\_element(std::vector<T>& data, T t) {

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

T buff = \*it;

std::replace\_if(data.begin(), data.end(), [&](T a){ return a == \*it; }, t);

return buff;

}

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

template<class T>

T pop\_min\_element(std::vector<T>& data) {

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

T buff = \*it;

auto pos = std::remove(data.begin(), data.end(), buff);

data.erase(pos, data.end());

return buff;

}

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

template<class T, class N>

T add\_average\_elements(std::vector<T>& data) {

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

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

std::for\_each(data.begin(), data.end(), [&](T& it){ it = it + average; });

return average;

}

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

template<class T>

void print(std::vector<T>& data) {

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

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

std::cout << data[i];

if(i != data.size()-1)

std::cout << ',';

}

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

}

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

};

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

#endif // LAB13\_TASK1\_HPP\_INCLUDED

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

lab13\_task2.hpp

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

#ifndef LAB13\_TASK2\_H\_INCLUDED

#define LAB13\_TASK2\_H\_INCLUDED

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

#include <stack>

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

namespace Task2 {

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

template <class T>

void copy\_stack\_to\_vector(std::stack<T>& stack, std::vector<T>& vector) {

while(!stack.empty()) {

vector.push\_back(stack.top());

stack.pop();

}

std::reverse(vector.begin(), vector.end());

}

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

template<class T>

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

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

data.push(function());

}

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

template<class T>

T replace\_max\_element(std::stack<T>& data, T t) {

T\* end = &data.top() + 1;

T\* begin = end - data.size();

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

T buff = \*it;

std::replace\_if(begin, end, [&](T a){ return a == \*it; }, t);

return buff;

}

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

template<class T>

T pop\_min\_element(std::stack<T>& data) {

T\* end = &data.top() + 1;

T\* begin = end - data.size();

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

T buff = \*it;

std::remove(begin, end, buff);

data.pop();

return buff;

}

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

template<class T, class N>

T add\_average\_elements(std::stack<T>& data) {

T\* end = &data.top() + 1;

T\* begin = end - data.size();

T average = std::accumulate(begin, end, T());

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

std::for\_each(begin, end, [&](T& it){ it = it + average; });

return average;

}

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

template<class T>

void print(std::stack<T>& data) {

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

T\* end = &data.top() + 1;

T\* begin = end - data.size();

std::vector<T> vector(begin, end);

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

std::cout << vector[i];

if(i != vector.size()-1)

std::cout << ',';

}

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

}

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

}

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

#endif // LAB13\_TASK2\_H\_INCLUDED

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

lab13\_task3.hpp

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

#ifndef LAB13\_TASK3\_HPP\_INCLUDED

#define LAB13\_TASK3\_HPP\_INCLUDED

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

#include <vector>

#include <numeric>

#include <functional>

#include <set>

#include "lab13\_task1.hpp"

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

namespace Task3 {

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

template<class T>

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

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

data.insert(function());

}

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

template<class T>

T replace\_max\_element(std::set<T>& data, T t) {

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

T buff = \*it;

data.erase(it);

data.insert(t);

return buff;

}

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

template<class T>

T pop\_min\_element(std::set<T>& data) {

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

T buff = \*it;

data.erase(it);

return buff;

}

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

template<class T, class N>

T add\_average\_elements(std::set<T>& data) {

std::vector<T> buff;

for(auto it : data)

buff.push\_back(it);

T average = Task1::add\_average\_elements<T, N>(buff);

data.clear();

std::for\_each(buff.begin(), buff.end(), [&](T& it){ data.insert(it); });

return average;

}

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

template<class T>

void print(std::set<T>& data) {

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

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

std::cout << \*it;

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

std::cout << ',';

}

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

}

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

};

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

#endif // LAB13\_TASK1\_HPP\_INCLUDED

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