Министерство образования и науки РФ

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

Лабораторная работа № 11

Тема: «Последовательные 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 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.

*Задача 4*

1. Создать адаптер контейнера.

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 5*

1. Создать параметризированный класс, используя в качестве контейнера адаптер

контейнера.

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 – вектор 2. Тип элементов – double   **Задача 2**  Тип элементов Time (см. лабораторную работу №3).  **Задача 3**  Параметризированный класс – Вектор(см. лабораторную работу № 7)  **Задача 4**  Адаптер контейнера – стек.  **Задача 5**  Параметризированный класс – Вектор  Адаптер контейнера – стек. | | |
| **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 вектор, 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 вектора.

Работа со стеком будет выполняться через копирования значений в вектор и вызов написанных ранее функций, что опять же даёт большой плюс к использованию уже написанного инструмента.

*Код программы:*

Прикреплён в приложении 1.

*UML-диаграмма класса:*

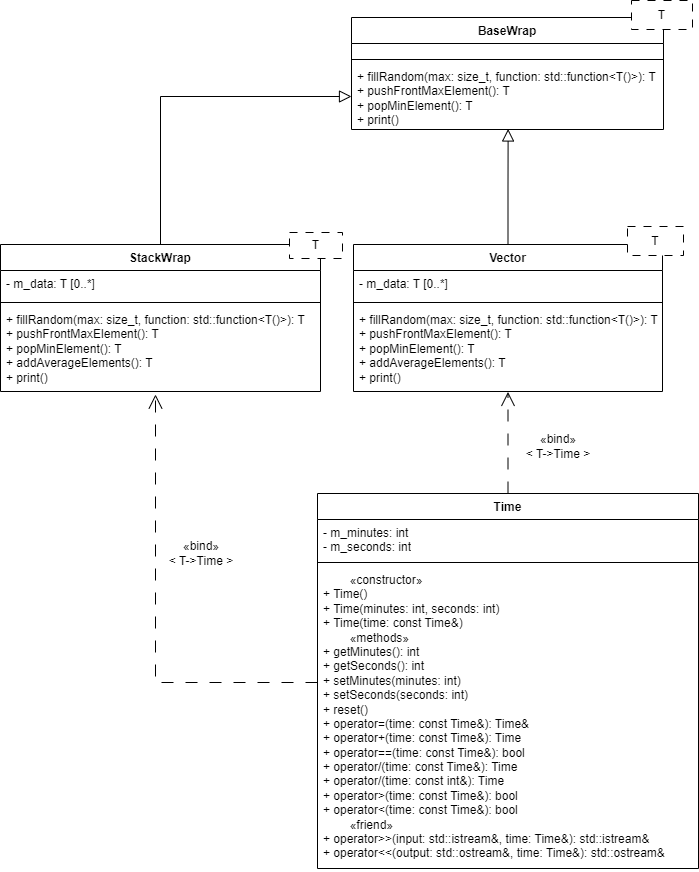


Рисунок 1 - UML-диаграмма классов

*Скриншот работы программы:*

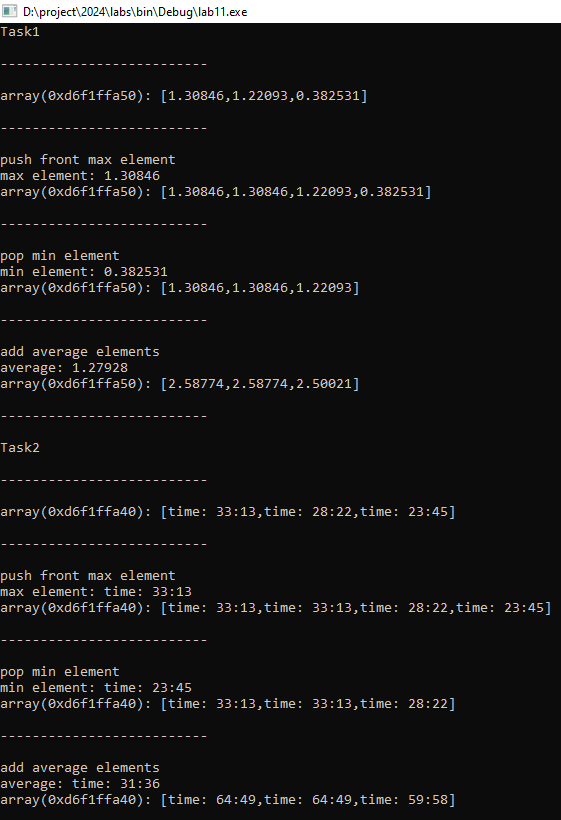


Рисунок 2 – Результат выполнения работы программы часть 1

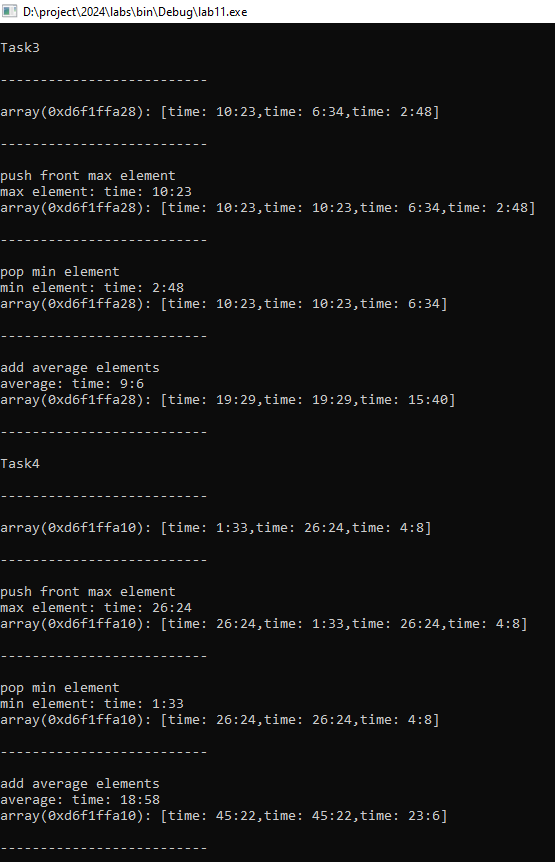


Рисунок 3 – Результат выполнения программы часть 2

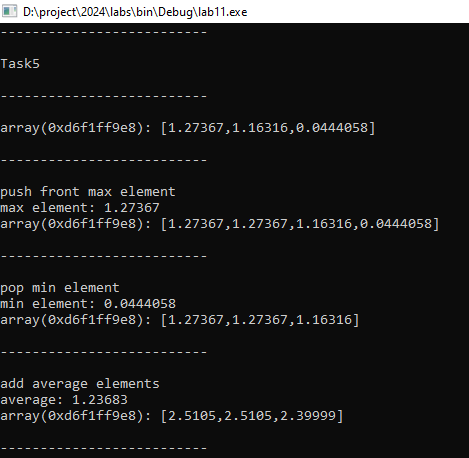


Рисунок 4 – Результат выполнения программы часть 3

*Контрольные вопросы:*

1. Из каких частей состоит библиотека STL?

Библиотека STL состоит из контейнеров, итераторов, алгоритмов и функциональных объектов.

2. Какие типы контейнеров существуют в STL?

Существует несколько типов контейнеров:

- Последовательные контейнеры: vector, list, deque

- Ассоциативные контейнеры: set, map, multiset, multimap

- Непосредственные контейнеры: stack, queue, priority\_queue

3. Что нужно сделать для использования контейнера STL в своей программе?

Необходимо подключить соответствующий заголовочный файл, инициализировать контейнер.

4. Что представляет собой итератор?

Итератор — это объект, который предоставляет доступ к элементам контейнера, позволяя перебрать их.

5. Какие операции можно выполнять над итераторами?

Можно выполнять операции как:

- Инкремент (++iterator)

- Декремент (--iterator)

- Доступ к элементу (\*iterator)

- Сравнение (iterator1 == iterator2, iterator1 != iterator2)

6. Каким образом можно организовать цикл для перебора контейнера с

использованием итератора?

std::vector<int> vec = {1, 2, 3, 4, 5};

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

std::cout << \*it << " "; // доступ к элементу

}

7. Какие типы итераторов существуют?

Существуют следующие типы итераторов:

- Входные (input)

- Выходные (output)

- Произвольные (random access)

- Двухсторонние (bidirectional)

- Односторонние (forward)

8. Перечислить операции и методы общие для всех контейнеров.

Некоторые общие операции:

- begin(), end()

- size()

- empty()

- clear()

9. Какие операции являются эффективными для контейнера vector? Почему?

Эффективны:

- Доступ по индексу (operator[])

- Добавление в конец (push\_back)

Потому что vector хранит элементы в непрерывной памяти.

10. Какие операции являются эффективными для контейнера list? Почему?

- Вставка/удаление в любом месте (insert, erase)

Потому что list реализован как двусвязный список.

11. Какие операции являются эффективными для контейнера deque? Почему?

Эффективны:

- Доступ по индексу

- Вставка/удаление с обоих концов (push\_front, push\_back)

Потому что deque организован как двусторонняя очередь.

12. Перечислить методы, которые поддерживает последовательный контейнер

vector.

Методы vector:

- push\_back()

- pop\_back()

- insert()

- erase()

- at()

- resize()

- clear()

13. Перечислить методы, которые поддерживает последовательный контейнер list.

Методы list:

- push\_back()

- push\_front()

- pop\_back()

- pop\_front()

- insert()

- erase()

- clear()

14. Перечислить методы, которые поддерживает последовательный контейнер

deque.

Методы deque:

- push\_back()

- push\_front()

- pop\_back()

- pop\_front()

- insert()

- erase()

- clear()

15. Задан контейнер vector. Как удалить из него элементы со 2 по 5?

std::vector<int> vec = {1, 2, 3, 4, 5, 6, 7, 8};

vec.erase(vec.begin() + 1, vec.begin() + 5); // Удалим элементы 2, 3, 4, 5

16. Задан контейнер vector. Как удалит из него последний элемент?

#include <vector>

#include <iostream>

int main() {

std::vector<int> vec = {1, 2, 3, 4, 5};

vec.pop\_back(); // Удаляет последний элемент

}

17. Задан контейнер list. Как удалить из него элементы со 2 по 5?

std::list<int> lst = {1, 2, 3, 4, 5, 6, 7, 8, 9};

auto it = lst.begin();

std::advance(it, 1); // позиция 2

auto it\_end = it;

std::advance(it\_end, 4); // позиция 5

lst.erase(it, it\_end); // Удаляет элементы со 2 по 5

18. Задан контейнер list. Как удалить из него последний элемент?

pop\_back();

19. Задан контейнер deque. Как удалить из него элементы со 2 по 5?

std::deque<int> deq = {1, 2, 3, 4, 5, 6, 7, 8, 9};

auto it = deq.begin();

std::advance(it, 1); // позиция 2

auto it\_end = it;

std::advance(it\_end, 4); // позиция 5

deq.erase(it, it\_end); // Удаляет элементы со 2 по 5

20. Задан контейнер deque. Как удалить из него последний элемент?

pop\_back();

21. Написать функцию для печати последовательного контейнера с использованием

итератора.

void printVector(const std::vector<int>& vec) {

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

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

}

}

22. Что представляют собой адаптеры контейнеров?

Адаптеры контейнеров — это шаблоны, которые изменяют интерфейс базового контейнера, предоставляя дополнительные функциональные возможности. Например, `stack`, `queue` и `priority\_queue` являются адаптерами, которые используют другие контейнеры (например, `deque`, `vector` или `list`) в качестве основы.

23. Чем отличаются друг от друга объявления stack<int> s и stack<int, list<int> > s?

Первый пример использует `deque` по умолчанию для хранения элементов, тогда как второй пример явно указывает, что в качестве базового контейнера используется `list`.

24. Перечислить методы, которые поддерживает контейнер stack.

- push();

- pop();

- top();

- empty();

- size();

25. Перечислить методы, которые поддерживает контейнер queue.

- push();

- pop();

- front();

- back();

- empty();

- size();

26. Чем отличаются друг от друга контейнеры queue и priority\_queue?

`queue` обеспечивает доступ к элементам в порядке очереди (FIFO), тогда как `priority\_queue` гарантирует, что элемент с наивысшим приоритетом будет доступен первым (по убыванию).

27. Задан контейнер stack. Как удалить из него элемент с заданным номером?

Удаление элемента по номеру в `stack` не поддерживается, так как он не предоставляет прямого доступа к элементам.

28. Задан контейнер queue. Как удалить из него элемент с заданным номером?

Аналогично, в `queue` нет прямого доступа для удаления элемента по номеру. Элементы можно удалять только по очереди (FIFO).

29. Написать функцию для печати контейнера stack с использованием итератора.

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";

}

30. Написать функцию для печати контейнера queue с использованием итератора.

void printQueue(std::queue<int> q) {

while (!q.empty()) {

std::cout << q.front() << " ";

q.pop();

}

}

**ПРИЛОЖЕНИЕ 1**

lab11.cpp

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

#include <ctime>

#include <cmath>

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

void task1();

void task2();

void task3();

void task4();

void task5();

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

int main() {

srand(time(0));

task1();

task2();

task3();

task4();

task5();

return 0;

}

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

lab11\_tasks.cpp

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

#include <iostream>

#include <vector>

#include "lab11\_time.hpp"

#include "lab11\_task1.hpp"

#include "lab11\_task3.hpp"

#include "lab11\_task4.hpp"

#include "lab11\_task5.hpp"

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

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

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

template <class T>

void template\_task(BaseWrap<T>\* data, std::function<T(void)> fillFunction, std::function<void(void)> custom = 0) {

IDENT\_PRINT;

data->fillRandom(3, fillFunction);

data->print();

IDENT\_PRINT;

T 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;

if(custom) {

custom();

IDENT\_PRINT;

}

}

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

void task1() {

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

std::vector<double> data;

IDENT\_PRINT;

Task1::fill\_array<double>(data, 3, []{ return (float)std::rand() / (float)(RAND\_MAX / 3); });

Task1::print(data);

IDENT\_PRINT;

double buff;

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

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

std::cout << "max element: " << buff << 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 << "add average elements\n";

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

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

Task1::print(data);

IDENT\_PRINT;

}

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

void task2() {

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

std::vector<Time> data;

IDENT\_PRINT;

Task1::fill\_array<Time>(data, 3, []{ return Time(std::rand()%40, std::rand() % MAX\_SECONDS); });

Task1::print(data);

IDENT\_PRINT;

Time buff;

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

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

std::cout << "max element: " << buff << 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 << "add average elements\n";

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

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

Task1::print(data);

IDENT\_PRINT;

}

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

void task3() {

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

Vector<Time> data;

template\_task<Time>(&data,

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

[&]{

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

Time buff = data.addAverageElements<int>();

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

data.print();

});

}

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

void task4() {

std::cout << "Task4\n";

std::stack<Time> data;

IDENT\_PRINT;

Task4::fill\_array<Time>(data, 3, []{ return Time(std::rand()%40, std::rand() % MAX\_SECONDS); });

Task4::print(data);

IDENT\_PRINT;

Time buff;

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

buff = Task4::push\_front\_max\_element(data);

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

Task4::print(data);

IDENT\_PRINT;

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

buff = Task4::pop\_min\_element(data);

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

Task4::print(data);

IDENT\_PRINT;

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

buff = Task4::add\_average\_elements<Time, int>(data);

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

Task4::print(data);

IDENT\_PRINT;

}

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

void task5() {

std::cout << "Task5\n";

StackWrap<double> data;

template\_task<double>(&data,

[]{ return (float)std::rand() / (float)(RAND\_MAX / 3); },

[&]{

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

double buff = data.addAverageElements<int>();

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

data.print();

});

}

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

lab11\_base\_wrap.hpp

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

#ifndef LAB11\_BASE\_WRAP\_HPP\_INCLUDED

#define LAB11\_BASE\_WRAP\_HPP\_INCLUDED

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

template <class T>

class BaseWrap {

public:

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

virtual void fillRandom(size\_t max, std::function<T()> function) = 0;

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

virtual T pushFrontMaxElement() = 0;

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

virtual T popMinElement() = 0;

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

virtual void print() = 0;

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

};

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

#endif // LAB11\_BASE\_WRAP\_HPP\_INCLUDED

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

lab11\_task1.hpp

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

#ifndef LAB11\_TASK1\_HPP\_INCLUDED

#define LAB11\_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 push\_front\_max\_element(std::vector<T>& data) {

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

T buff = \*it;

data.insert(data.begin(), \*it);

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;

data.erase(it);

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();

for(auto& it : data)

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 // LAB11\_TASK1\_HPP\_INCLUDED

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

lab11\_task3.hpp

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

#ifndef LAB11\_TASK3\_HPP\_INCLUDED

#define LAB11\_TASK3\_HPP\_INCLUDED

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

#include "lab11\_base\_wrap.hpp"

#include "lab11\_task1.hpp"

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

template<class T>

class Vector : public BaseWrap<T> {

private:

typedef std::vector<T> data\_t;

public:

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

~Vector() = default;

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

/\*\*

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

\*/

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

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

}

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

T pushFrontMaxElement() {

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

}

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

T popMinElement() {

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

}

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

template <class N>

T addAverageElements() {

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

}

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

void print() {

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

}

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

private:

data\_t m\_data;

};

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

#endif // LAB11\_TASK3\_HPP\_INCLUDED

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

lab11\_task4.hpp

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

#ifndef LAB11\_TASK4\_H\_INCLUDED

#define LAB11\_TASK4\_H\_INCLUDED

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

#include <stack>

#include "lab11\_task1.hpp"

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

namespace Task4 {

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

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 push\_front\_max\_element(std::stack<T>& data) {

std::vector<T> vector;

copy\_stack\_to\_vector(data, vector);

T buff = Task1::push\_front\_max\_element<T>(vector);

for(auto& it : vector)

data.push(it);

return buff;

}

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

template<class T>

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

std::vector<T> vector;

copy\_stack\_to\_vector(data, vector);

T buff = Task1::pop\_min\_element<T>(vector);

for(auto& it : vector)

data.push(it);

return buff;

}

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

template<class T, class N>

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

std::vector<T> vector;

copy\_stack\_to\_vector(data, vector);

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

for(auto& it : vector)

data.push(it);

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 // LAB11\_TASK4\_H\_INCLUDED

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

lab11\_task5.hpp

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

#ifndef LAB11\_TASK5\_HPP\_INCLUDED

#define LAB11\_TASK5\_HPP\_INCLUDED

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

#include <stack>

#include "lab11\_base\_wrap.hpp"

#include "lab11\_task4.hpp"

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

template <class T>

class StackWrap : public BaseWrap<T> {

private:

typedef std::stack<T> data\_t;

public:

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

~StackWrap() = default;

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

/\*\*

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

\*/

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

Task4::fill\_array<T>(m\_data, max, function);

}

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

T pushFrontMaxElement() {

return Task4::push\_front\_max\_element<T>(m\_data);

}

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

T popMinElement() {

return Task4::pop\_min\_element<T>(m\_data);

}

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

template <class N>

T addAverageElements() {

return Task4::add\_average\_elements<T, N>(m\_data);

}

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

void print() {

Task4::print<T>(m\_data);

}

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

private:

data\_t m\_data;

};

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

#endif // LAB11\_TASK5\_HPP\_INCLUDED

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