МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ПНИПУ)

Факультет: Электротехнический

Кафедра: «Информационные технологии и автоматизированные системы» (ИТАС)

Направление: Разработка информационных систем (РИС)

**О Т Ч Ё Т**   
**о лабораторной работе №11**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: «Последовательные контейнеры библиотеки STL»

Выполнил:

Студент группы РИС-24-3б

Суяргулов А.О.

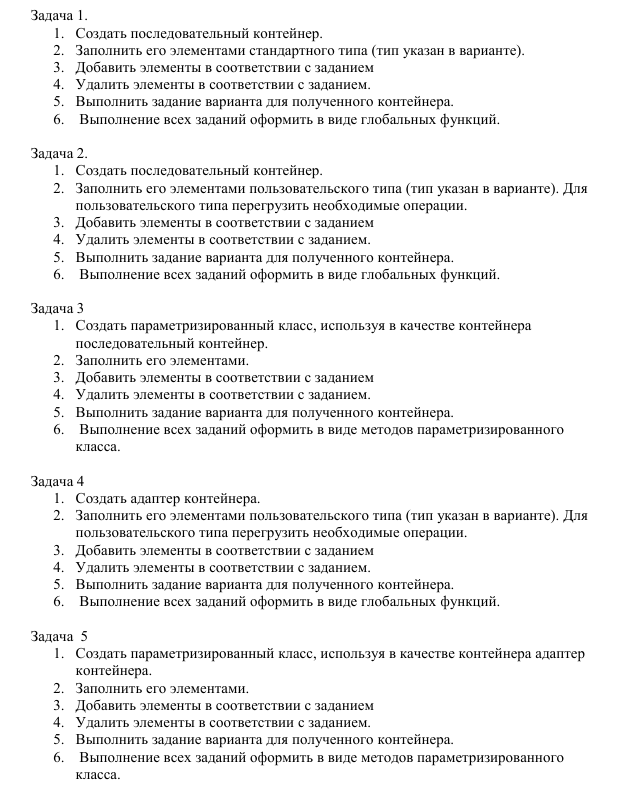
Проверил:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2025

ВВЕДЕНИЕ

ОСНОВНАЯ ЧАСТЬ

**Диаграмма классов:**

**Код:**

**Main:**

#include <iostream>

#include <vector>

#include <algorithm>

#include <stack>

using namespace std;

// ЗАДАЧА 1 (vector<double>)

void task1() {

vector<double> vec = { 3.14, 2.71, 1.618, 0.577 };

// Добавление элементов

vec.push\_back(4.669);

vec.insert(vec.begin() + 2, 1.414);

// Удаление элементов

vec.erase(remove(vec.begin(), vec.end(), 0.577), vec.end());

// Добавление максимума в начало

if (!vec.empty()) {

double max\_val = \*max\_element(vec.begin(), vec.end());

vec.insert(vec.begin(), max\_val);

}

cout << "Task1: ";

for (double num : vec) cout << num << " ";

cout << endl;

}

// ЗАДАЧА 2 (vector<Money>)

class Money {

public:

long rub;

int kop;

Money(long r = 0, int k = 0) : rub(r), kop(k) { normalize(); }

void normalize() {

if (kop >= 100) {

rub += kop / 100;

kop %= 100;

}

}

bool operator<(const Money& other) const {

return (rub < other.rub) || (rub == other.rub && kop < other.kop);

}

friend ostream& operator<<(ostream& os, const Money& m) {

os << m.rub << "." << (m.kop < 10 ? "0" : "") << m.kop;

return os;

}

};

void task2() {

vector<Money> vec = { Money(150, 75), Money(200, 50), Money(100, 99) };

// Добавление элементов

vec.push\_back(Money(300, 25));

vec.insert(vec.begin(), Money(50, 0));

// Удаление элемента

vec.erase(vec.begin() + 1);

// Удаление минимального элемента

if (!vec.empty()) {

auto min\_it = min\_element(vec.begin(), vec.end());

vec.erase(min\_it);

}

cout << "Task2: ";

for (const Money& m : vec) cout << m << " ";

cout << endl;

}

// ЗАДАЧА 3 (MySet template)

template <typename T>

class MySet {

vector<T> data;

public:

void add(const T& item) {

if (find(data.begin(), data.end(), item) == data.end())

data.push\_back(item);

}

void remove(const T& item) {

data.erase(remove(data.begin(), data.end(), item), data.end());

}

// Ручной подсчёт суммы вместо accumulate

void addAverageToElements() {

if (data.empty()) return;

T sum = T();

for (const T& item : data) sum = sum + item;

T avg = sum / data.size();

for (T& item : data) item = item + avg;

}

void print() {

cout << "Task3: ";

for (const T& item : data) cout << item << " ";

cout << endl;

}

};

// ЗАДАЧА 4 (stack<Money>)

void task4() {

stack<Money> st;

vector<Money> temp = { Money(100, 50), Money(500, 99), Money(200, 10) };

// Заполнение стека

for (const Money& m : temp) st.push(m);

// Поиск и удаление минимума через вектор

vector<Money> helper;

while (!st.empty()) {

helper.push\_back(st.top());

st.pop();

}

if (!helper.empty()) {

auto min\_it = min\_element(helper.begin(), helper.end());

helper.erase(min\_it);

}

// Восстановление стека

for (auto it = helper.rbegin(); it != helper.rend(); ++it)

st.push(\*it);

// Вывод

cout << "Task4: ";

while (!st.empty()) {

cout << st.top() << " ";

st.pop();

}

cout << endl;

}

// ЗАДАЧА 5 (StackSet template)

template <typename T>

class StackSet {

stack<T> data;

public:

void push(const T& item) {

data.push(item);

}

// Ручной подсчёт суммы

void addAverageToElements() {

if (data.empty()) return;

vector<T> temp;

T sum = T();

size\_t count = 0;

// Сбор элементов и суммы

while (!data.empty()) {

sum = sum + data.top();

temp.push\_back(data.top());

data.pop();

count++;

}

T avg = sum / count;

// Добавление среднего и восстановление стека

for (T& item : temp) item = item + avg;

for (auto it = temp.rbegin(); it != temp.rend(); ++it)

data.push(\*it);

}

void print() {

stack<T> temp = data;

cout << "Task5: ";

while (!temp.empty()) {

cout << temp.top() << " ";

temp.pop();

}

cout << endl;

}

};

// Перегрузка операторов для Money

Money operator+(const Money& a, const Money& b) {

return Money(a.rub + b.rub, a.kop + b.kop);

}

Money operator/(const Money& m, int div) {

long total = m.rub \* 100 + m.kop;

total /= div;

return Money(total / 100, total % 100);

}

int main() {

task1();

task2();

MySet<int> set3;

set3.add(10);

set3.add(20);

set3.addAverageToElements();

set3.print();

task4();

StackSet<Money> set5;

set5.push(Money(100, 50));

set5.push(Money(200, 75));

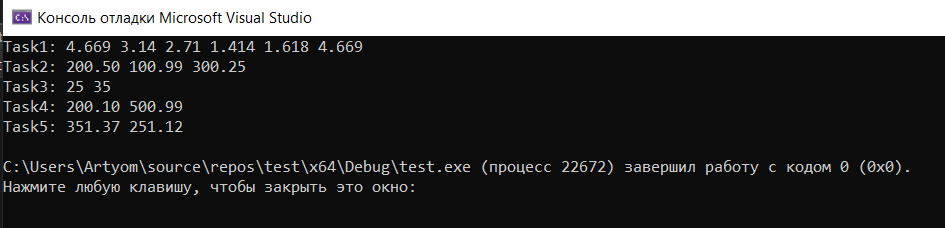
set5.addAverageToElements();

set5.print();

return 0;

}

**Результат работы кода:**

****

**Контрольные вопросы:**

1. Из каких частей состоит библиотека STL?

\* Контейнеры (containers): структуры данных для хранения объектов (например, vector, list, deque, set, map).

\* Итераторы (iterators): объекты, предоставляющие способ доступа к элементам контейнера (аналоги указателям).

\* Алгоритмы (algorithms): функции для выполнения различных операций над контейнерами (например, сортировка, поиск, копирование).

\* Функциональные объекты (function objects, functors): объекты, которые можно использовать как функции (например, предикаты для алгоритмов).

2. Какие типы контейнеров существуют в STL?

\* Последовательные контейнеры:

\* vector

\* deque

\* list

\* forward\_list (односвязный список)

\* array (статический массив)

\* Ассоциативные контейнеры:

\* set

\* multiset

\* map

\* multimap

\* Неупорядоченные ассоциативные контейнеры (C++11 и новее):

\* unordered\_set

\* unordered\_multiset

\* unordered\_map

\* unordered\_multimap

\* Адаптеры контейнеров:

\* stack

\* queue

\* priority\_queue

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

\* Включить соответствующий заголовочный файл с помощью директивы #include. Например, для vector нужно #include <vector>.

\* Использовать пространство имен std (либо с помощью using namespace std;, либо указывая std:: перед каждым элементом STL).

4. Что представляет собой итератор?

\* Итератор – это объект, который действует как указатель на элемент контейнера. Он позволяет перемещаться по контейнеру и получать доступ к элементам.

5. Какие операции можно выполнять над итераторами?

\* \* (разыменование): получение значения элемента, на который указывает итератор.

\* ++ (инкремент): перемещение итератора к следующему элементу.

\* -- (декремент): перемещение итератора к предыдущему элементу (для двунаправленных итераторов).

\* == и != (сравнение): проверка, указывают ли два итератора на один и тот же элемент.

\* -> (доступ к члену объекта): если итератор указывает на объект, получить доступ к его членам.

\* +, -, +=, -=, [] (для итераторов произвольного доступа, например, для vector).

6. Каким образом можно организовать цикл для перебора контейнера с использованием итератора?

#include <iostream>

#include <vector>

int main() {

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

// Способ 1: Используя begin() и end()

for (std::vector<int>::iterator it = myVector.begin(); it != myVector.end(); ++it) {

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

}

std::cout << std::endl;

// Способ 2: Используя range-based for loop - самый простой

for (int element : myVector) {

std::cout << element << " ";

}

std::cout << std::endl;

// Способ 3: Используя const\_iterator

for (std::vector<int>::const\_iterator it = myVector.cbegin(); it != myVector.cend(); ++it) {

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

}

std::cout << std::endl;

return 0;

}

7. Какие типы итераторов существуют?

\* Итераторы ввода (input iterators)

\* Итераторы вывода (output iterators)

\* Однонаправленные итераторы (forward iterators)

\* Двунаправленные итераторы (bidirectional iterators)

\* Итераторы произвольного доступа (random access iterators)

8. Перечислить операции и методы общие для всех контейнеров.

\* empty(): проверка, пуст ли контейнер.

\* size(): возвращает количество элементов в контейнере.

\* max\_size(): возвращает максимальное количество элементов, которое может содержать контейнер.

\* begin(): возвращает итератор на начало контейнера.

\* end(): возвращает итератор на конец контейнера.

\* swap(): обменивает содержимое двух контейнеров.

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

\* Доступ к элементу по индексу ([], at()): O(1), поскольку элементы хранятся в непрерывном блоке памяти.

\* Добавление и удаление элементов в конце (push\_back(), pop\_back()): O(1) амортизированное, поскольку иногда может потребоваться перевыделение памяти.

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

\* Вставка и удаление элементов в любом месте (insert(), erase()): O(1), поскольку требуется только изменение указателей.

\* Перемещение элементов между списками (splice()): O(1).

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

\* Добавление и удаление элементов в начале и в конце (push\_front(), pop\_front(), push\_back(), pop\_back()): O(1) амортизированное.

\* Доступ к элементу по индексу ([], at()): O(1) (медленнее, чем у vector, но все еще константное время).

12. Перечислить методы, которые поддерживает последовательный контейнер vector.

\* assign()

\* at()

\* back()

\* begin()

\* capacity()

\* cbegin()

\* cend()

\* clear()

\* crbegin()

\* crend()

\* data()

\* emplace()

\* emplace\_back()

\* empty()

\* end()

\* erase()

\* front()

\* get\_allocator()

\* insert()

\* max\_size()

\* operator[]

\* pop\_back()

\* push\_back()

\* rbegin()

\* rend()

\* reserve()

\* resize()

\* shrink\_to\_fit()

\* size()

\* swap()

13. Перечислить методы, которые поддерживает последовательный контейнер list.

\* assign()

\* back()

\* begin()

\* cbegin()

\* cend()

\* clear()

\* crbegin()

\* crend()

\* emplace()

\* emplace\_back()

\* emplace\_front()

\* empty()

\* end()

\* erase()

\* front()

\* get\_allocator()

\* insert()

\* max\_size()

\* merge()

\* pop\_back()

\* pop\_front()

\* push\_back()

\* push\_front()

\* rbegin()

\* remove()

\* remove\_if()

\* rend()

\* resize()

\* reverse()

\* size()

\* sort()

\* splice()

\* swap()

\* unique()

14. Перечислить методы, которые поддерживает последовательный контейнер deque.

\* assign()

\* at()

\* back()

\* begin()

\* cbegin()

\* cend()

\* clear()

\* crbegin()

\* crend()

\* emplace()

\* emplace\_back()

\* emplace\_front()

\* empty()

\* end()

\* erase()

\* front()

\* get\_allocator()

\* insert()

\* max\_size()

\* operator[]

\* pop\_back()

\* pop\_front()

\* push\_back()

\* push

front()

\* rbegin()

\* rend()

\* resize()

\* shrink\_to\_fit()

\* size()

\* swap()

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

#include <iostream>

#include <vector>

int main() {

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

// Удаляем элементы с индекса 1 (второй элемент) по индекс 4 (пятый элемент)

myVector.erase(myVector.begin() + 1, myVector.begin() + 5);

// Вывод содержимого вектора

for (int element : myVector) {

std::cout << element << " ";

}

std::cout << std::endl; // Вывод: 1 6 7

return 0;

}

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

#include <iostream>

#include <vector>

int main() {

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

if (!myVector.empty()) {

myVector.pop\_back();

}

// Вывод содержимого вектора

for (int element : myVector) {

std::cout << element << " ";

}

std::cout << std::endl; // Вывод: 1 2 3 4

return 0;

}

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

#include <iostream>

#include <list>

int main() {

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

// Получаем итератор на второй элемент

std::list<int>::iterator itBegin = myList.begin();

std::advance(itBegin, 1); // Перемещаем итератор на 1 позицию

// Получаем итератор на шестой элемент (конец диапазона удаления)

std::list<int>::iterator itEnd = myList.begin();

std::advance(itEnd, 5); // Перемещаем итератор на 5 позиций

// Удаляем элементы

myList.erase(itBegin, itEnd);

// Вывод содержимого списка

for (int element : myList) {

std::cout << element << " ";

}

std::cout << std::endl; // Вывод: 1 6 7

return 0;

}

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

#include <iostream>

#include <list>

int main() {

std::list<int> myList = {1, 2, 3, 4, 5};

if (!myList.empty()) {

myList.pop\_back();

}

// Вывод содержимого списка

for (int element : myList) {

std::cout << element << " ";

}

std::cout << std::endl; // Вывод: 1 2 3 4

return 0;

}

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

#include <iostream>

#include <deque>

int main() {

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

// Удаляем элементы с индекса 1 (второй элемент) по индекс 4 (пятый элемент)

myDeque.erase(myDeque.begin() + 1, myDeque.begin() + 5);

// Вывод содержимого deque

for (int element : myDeque) {

std::cout << element << " ";

}

std::cout << std::endl; // Вывод: 1 6 7

return 0;

}

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

#include <iostream>

#include <deque>

int main() {

std::deque<int> myDeque = {1, 2, 3, 4, 5};

if (!myDeque.empty()) {

myDeque.pop\_back();

}

// Вывод содержимого deque

for (int element : myDeque) {

std::cout << element << " ";

}

std::cout << std::endl; // Вывод: 1 2 3 4

return 0;

}

21. Написать функцию для печати последовательного контейнера с использованием итератора.

#include <iostream>

#include <vector>

template <typename Container>

void printContainer(const Container& container) {

for (typename Container::const\_iterator it = container.begin(); it != container.end(); ++it) {

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

}

std::cout << std::endl;

}

int main() {

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

printContainer(myVector); // Вывод: 1 2 3 4 5

return 0;

}

22. Что представляют собой адаптеры контейнеров?

\* Адаптеры контейнеров – это классы, которые предоставляют другой интерфейс к существующим контейнерам. Они не хранят данные напрямую, а используют один из существующих контейнеров для хранения.

\* Примеры: stack, queue, priority\_queue.

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

\* stack<int> s: Создает стек, использующий deque (двусторонняя очередь) по умолчанию в качестве базового контейнера.

\* stack<int, list<int>> s: Создает стек, использующий list (двусвязный список) в качестве базового контейнера.

24. Перечислить методы, которые поддерживает контейнер stack.

\* empty(): Проверяет, пуст ли стек.

\* size(): Возвращает количество элементов в стеке.

\* top(): Возвращает ссылку на верхний элемент стека.

\* push(element): Добавляет элемент на верх стека.

\* pop(): Удаляет верхний элемент стека.

25. Перечислить методы, которые поддерживает контейнер queue.

\* empty(): Проверяет, пуста ли очередь.

\* size(): Возвращает количество элементов в очереди.

\* front(): Возвращает ссылку на первый элемент в очереди.

\* back(): Возвращает ссылку на последний элемент в очереди.

\* push(element): Добавляет элемент в конец очереди.

\* pop(): Удаляет первый элемент из очереди.

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

\* queue: Элементы добавляются в конец и удаляются из начала (FIFO – First-In, First-Out).

\* priority\_queue: Элементы добавляются в произвольном порядке, но удаляется всегда элемент с наивысшим приоритетом (обычно наибольший элемент).

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

\* В стеке нельзя напрямую удалить элемент по номеру, так как стек предоставляет доступ только к верхнему элементу. Чтобы удалить элемент с заданным номером, нужно временно перенести элементы из стека во вспомогательный контейнер (например, vector или deque), удалить нужный элемент из вспомогательного контейнера, а затем вернуть элементы обратно в стек.

#include <iostream>

#include <stack>

#include <vector>

int main() {

std::stack<int> myStack;

myStack.push(1);

myStack.push(2);

myStack.push(3);

myStack.push(4);

myStack.push(5);

int indexToRemove = 2; // Номер элемента для удаления (считая сверху, с 0)

std::vector<int> tempVector;

int currentIndex = 0;

while (!myStack.empty()) {

if (currentIndex != indexToRemove) {

tempVector.push\_back(myStack.top());

}

myStack.pop();

currentIndex++;

}

// Возвращаем элементы обратно в стек в обратном порядке

for (int i = tempVector.size() - 1; i >= 0; --i) {

myStack.push(tempVector[i]);

}

// Вывод содержимого стека

std::stack<int> tempStack = myStack; // Копия для вывода

std::cout << "Содержимое стека после удаления: ";

while (!tempStack.empty()) {

std::cout << tempStack.top() << " ";

tempStack.pop();

}

std::cout << std::endl; // Вывод: 1 2 4 5

return 0;

}

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

• Аналогично стеку, в очереди нельзя напрямую удалить элемент по номеру. Нужно временно перенести элементы из очереди во вспомогательный контейнер, удалить нужный элемент и вернуть элементы обратно.

#include <iostream>

#include <queue>

#include <vector>

int main() {

std::queue<int> myQueue;

myQueue.push(1);

myQueue.push(2);

myQueue.push(3);

myQueue.push(4);

myQueue.push(5);

int indexToRemove = 2; // Номер элемента для удаления (считая с начала, с 0)

std::vector<int> tempVector;

int currentIndex = 0;

while (!myQueue.empty()) {

if (currentIndex != indexToRemove) {

tempVector.push\_back(myQueue.front());

}

myQueue.pop();

currentIndex++;

}

// Возвращаем элементы обратно в очередь

for (int element : tempVector) {

myQueue.push(element);

}

// Вывод содержимого очереди

std::queue<int> tempQueue = myQueue; // Копия для вывода

std::cout << "Содержимое очереди после удаления: ";

while (!tempQueue.empty()) {

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

tempQueue.pop();

}

std::cout << std::endl; // Вывод: 1 2 4 5

return 0;

}

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

• У контейнера stack нет прямых итераторов. Чтобы напечатать стек с использованием итератора, нужно сначала перенести элементы во вспомогательный контейнер (например, vector), а затем использовать итератор для печати элементов этого вспомогательного контейнера.

#include <iostream>

#include <stack>

#include <vector>

template <typename T>

void printStackWithIterator(std::stack<T> s) {

std::vector<T> tempVector;

while (!s.empty()) {

tempVector.push\_back(s.top());

s.pop();

}

// Выводим элементы в обратном порядке, чтобы соответствовать порядку стека

std::cout << "Содержимое стека: ";

for (auto it = tempVector.rbegin(); it != tempVector.rend(); ++it) {

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

}

std::cout << std::endl;

}

int main() {

std::stack<int> myStack;

myStack.push(1);

myStack.push(2);

myStack.push(3);

printStackWithIterator(myStack); // Вывод: Содержимое стека: 3 2 1

return 0;

}

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

• Аналогично стеку, у очереди нет прямых итераторов.

#include <iostream>

#include <queue>

#include <vector>

template <typename T>

void printQueueWithIterator(std::queue<T> q) {

std::vector<T> tempVector;

while (!q.empty()) {

tempVector.push\_back(q.front());

q.pop();

}

// Выводим элементы в порядке очереди

std::cout << "Содержимое очереди: ";

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

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

}

std::cout << std::endl;

}

int main() {

std::queue<int> myQueue;

myQueue.push(1);

myQueue.push(2);

myQueue.push(3);

printQueueWithIterator(myQueue); // Вывод: Содержимое очереди: 1 2 3

return 0;

}

**Ссылка на GitHub:**