Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы» направление подготовки: 09.03.04 – «Программная инженерия»

# Лабораторная работа " Последовательные контейнеры библиотеки STL."

Выполнил студент гр. РИС-24-3б

Жиряков Леонид Антонович

Проверил:

Доцент кафедры ИТАС   
Ольга Андреевна Полякова

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (оценка) (подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

г. Пермь, 2024

Вариант 8.

Постановка задачи:

Общая:

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.

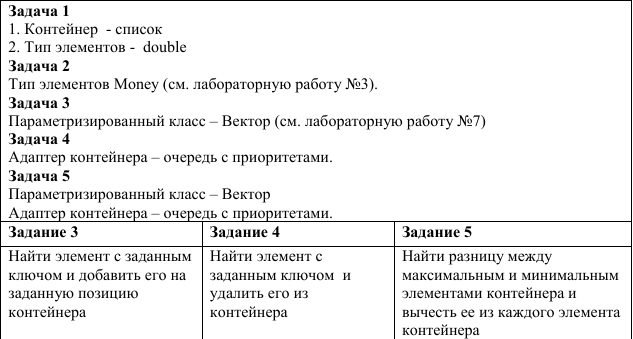
3. Добавить элементы в соответствии с заданием

4. Удалить элементы в соответствии с заданием.

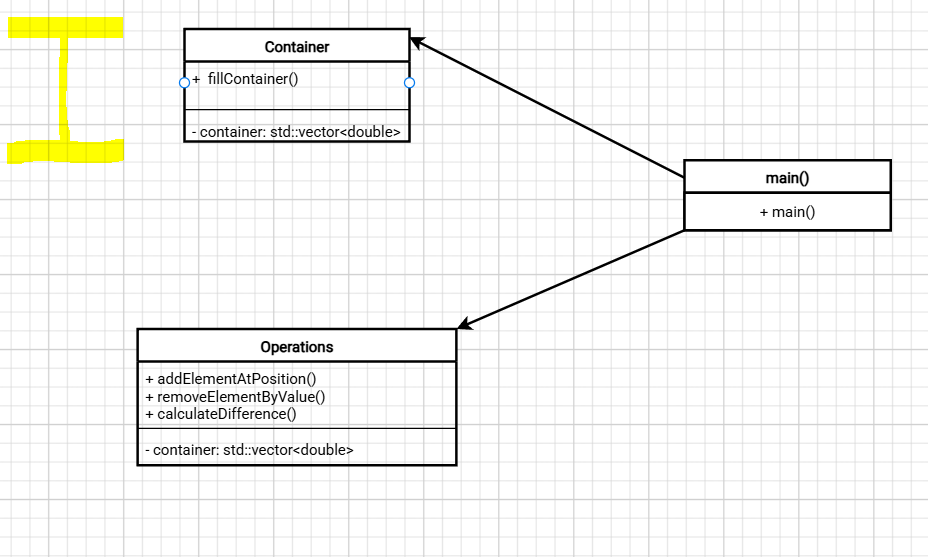
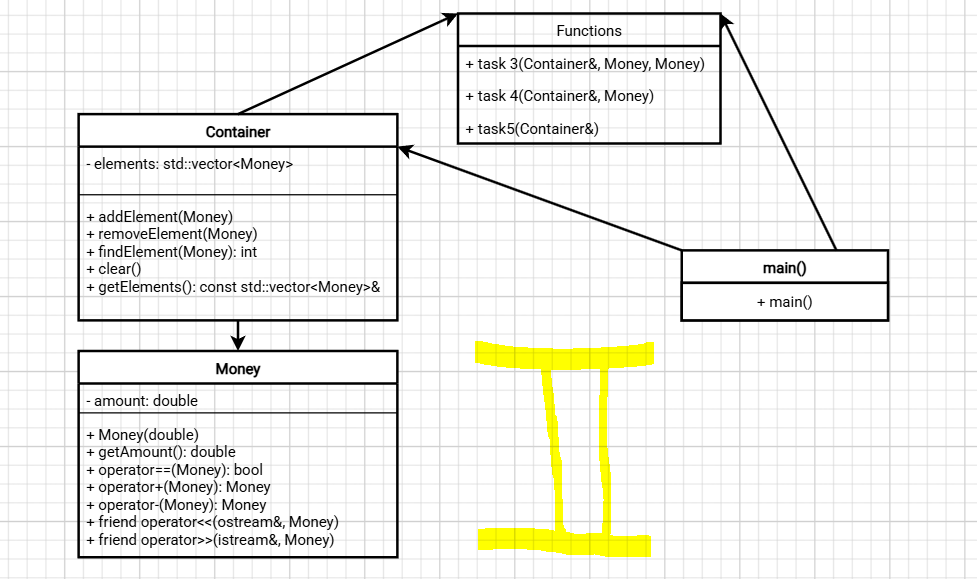
5. Выполнить задание варианта для полученного контейнера.

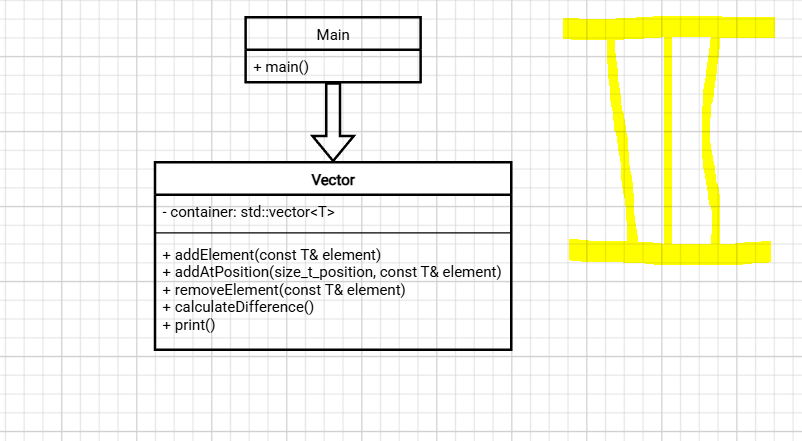
6. Выполнение всех заданий оформить в виде методов параметризированного класса.

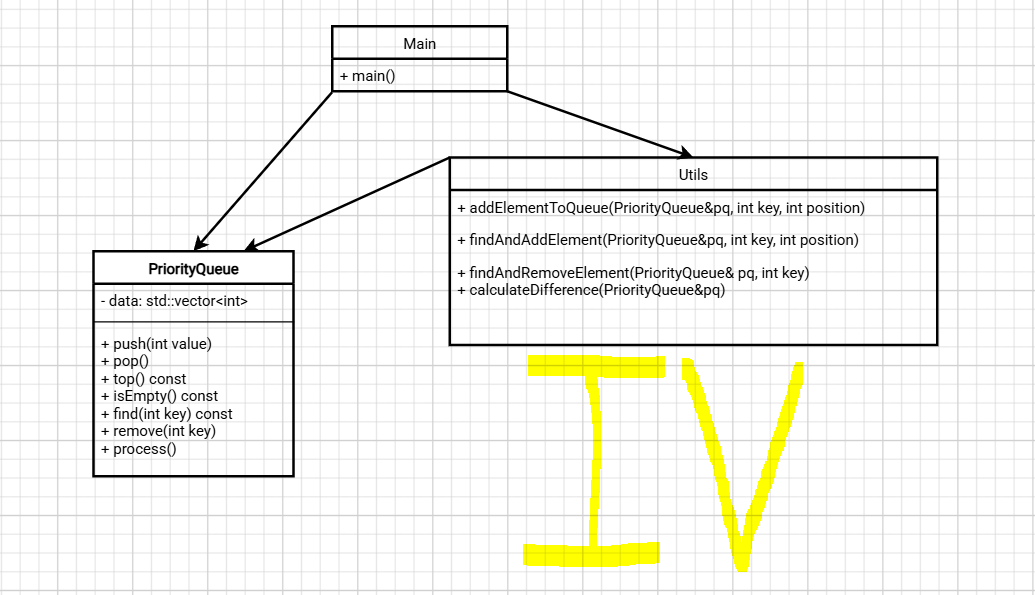
Персональная:

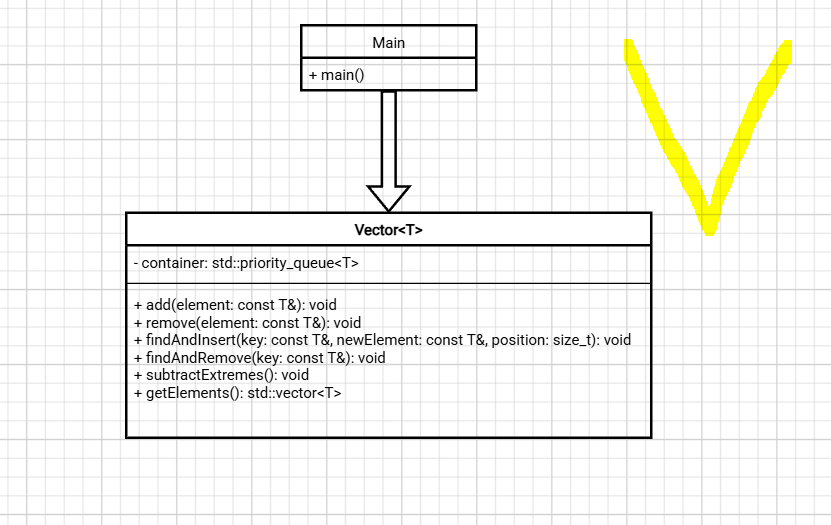


UML – диаграмма:







Программная реализация

**Заголовочный файлы**

**I**

**Container.h:**

#pragma once

#include <vector>

void fillContainer(std::vector<double>& container, int size);

operations.h:

#pragma once

#include <vector>

void addElementAtPosition(std::vector<double>& container, double value, size\_t position);

void removeElementByValue(std::vector<double>& container, double value);

void calculateDifference(std::vector<double>& container);

**II**

**Container.h:**

#pragma once

#include <vector>

#include "Money.h"

class Container

{

public:

void addElement(const Money& money);

void removeElement(const Money& money);

int findElement(const Money& money) const;

void clear();

const std::vector<Money>& getElements() const;

private:

std::vector<Money> elements;

};

**functions.h:**

#pragma once

#include "Container.h"

void task3(Container& container, const Money& moneyToFind, const Money& moneyToAdd);

void task4(Container& container, const Money& moneyToFind);

void task5(Container& container);

**Money.h:**

#pragma once

#include <iostream>

class Money

{

public:

Money(double amount = 0.0);

double getAmount() const;

bool operator==(const Money& other) const;

Money operator+(const Money& other) const;

Money operator-(const Money& other) const;

friend std::ostream& operator<<(std::ostream& os, const Money& money);

friend std::istream& operator>>(std::istream& is, Money& money);

private:

double amount;

};

**III**

**Vector.h:**

#pragma once

#include <iostream>

#include <vector>

#include <algorithm>

template<typename T>

class Vector

{

private:

std::vector<T> container;

public:

void addElement(const T& element);

void addAtPosition(size\_t position, const T& element);

void removeElement(const T& element);

void calculateDifference();

void print();

};

**IV  
Utils.h:**#pragma once

#include "PriorityQueue.h"

void addElementToQueue(PriorityQueue& pq, int key, int position);

void findAndAddElement(PriorityQueue& pq, int key, int position);

void findAndRemoveElement(PriorityQueue& pq, int key);

void calculateDifference(PriorityQueue& pq);

**PriorityQueue.h:**

#pragma once

#include <vector>

#include <stdexcept>

class PriorityQueue

{

private:

std::vector<int> data;

public:

void push(int value);

void pop();

int top() const;

bool isEmpty() const;

int find(int key) const; // Задание 3

void remove(int key); // Задание 4

void process(); // Задание 5

};

**V  
Vector.h:**#pragma once

#include <queue>

#include <vector>

#include <stdexcept>

template<typename T>

class Vector

{

private:

std::priority\_queue<T> container;

public:

// Метод для добавления элемента

void add(const T& element);

// Метод для удаления элемента

void remove(const T& element);

// Метод для поиска элемента по ключу и добавления на заданную позицию

void findAndInsert(const T& key, const T& newElement, size\_t position);

// Метод для поиска элемента по ключу и удаления его

void findAndRemove(const T& key);

// Метод для нахождения разницы между максимальным и минимальным элементами и вычитание из каждого

void subtractExtremes();

// Метод для получения всех элементов (для вывода)

std::vector<T> getElements() const;

};

**Реализация методов**

**I**

#include "container.h"

#include <cstdlib> // для rand()

void fillContainer(std::vector<double>& container, int size)

{

for (int i = 0; i < size; ++i)

{

container.push\_back(static\_cast<double>(rand() % 100)); // Заполнение случайными числами

}

}

#include "operations.h"

#include <algorithm>

#include <iostream>

void addElementAtPosition(std::vector<double>& container, double value, size\_t position)

{

if (position <= container.size())

{

container.insert(container.begin() + position, value);

}

else

{

std::cout << "Ошибка: позиция выходит за границы контейнера." << std::endl;

}

}

void removeElementByValue(std::vector<double>& container, double value)

{

auto it = std::remove(container.begin(), container.end(), value);

if (it != container.end())

{

container.erase(it, container.end());

}

else

{

std::cout << "Элемент не найден для удаления." << std::endl;

}

}

void calculateDifference(std::vector<double>& container)

{

if (container.empty())

{

std::cout << "Контейнер пуст." << std::endl;

return;

}

double maxElement = \*std::max\_element(container.begin(), container.end());

double minElement = \*std::min\_element(container.begin(), container.end());

double difference = maxElement - minElement;

for (auto& elem : container)

{

elem -= difference; // вычитание разности из каждого элемента

}

std::cout << "Разница (max - min): " << difference << std::endl;

}

**II**

#include "Money.h"

Money::Money(double amount) : amount(amount) {}

double Money::getAmount() const

{

return amount;

}

bool Money::operator==(const Money& other) const

{

return amount == other.amount;

}

Money Money::operator+(const Money& other) const

{

return Money(amount + other.amount);

}

Money Money::operator-(const Money& other) const

{

return Money(amount - other.amount);

}

std::ostream& operator<<(std::ostream& os, const Money& money)

{

os << money.amount;

return os;

}

std::istream& operator>>(std::istream& is, Money& money)

{

is >> money.amount;

return is;

}

#include "functions.h"

#include <iostream>

#include <algorithm>

void task3(Container& container, const Money& moneyToFind, const Money& moneyToAdd)

{

int position = container.findElement(moneyToFind);

if (position != -1)

{

container.addElement(moneyToAdd);

std::cout << "Добавлен элемент: " << moneyToAdd << std::endl;

}

else

{

std::cout << "Элемент не найден." << std::endl;

}

}

void task4(Container& container, const Money& moneyToFind)

{

container.removeElement(moneyToFind);

std::cout << "Элемент " << moneyToFind << " удален." << std::endl;

}

void task5(Container& container)

{

const std::vector<Money>& elements = container.getElements();

if (elements.empty()) return;

Money maxElement = elements[0];

Money minElement = elements[0];

for (const auto& elem : elements)

{

if (elem.getAmount() > maxElement.getAmount()) maxElement = elem;

if (elem.getAmount() < minElement.getAmount()) minElement = elem;

}

double difference = maxElement.getAmount() - minElement.getAmount();

for (auto& elem : const\_cast<std::vector<Money>&>(elements))

{

elem = Money(elem.getAmount() - difference);

}

}

#include "Container.h"

void Container::addElement(const Money& money)

{

elements.push\_back(money);

}

void Container::removeElement(const Money& money)

{

auto it = std::remove(elements.begin(), elements.end(), money);

elements.erase(it, elements.end());

}

int Container::findElement(const Money& money) const

{

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

{

if (elements[i] == money)

{

return i;

}

}

return -1; // Не найдено

}

void Container::clear()

{

elements.clear();

}

const std::vector<Money>& Container::getElements() const

{

return elements;

}

**III**

**Vector.cpp:**

#include "Vector.h"

// Добавление элемента в контейнер

template<typename T>

void Vector<T>::addElement(const T& element)

{

container.push\_back(element);

}

// Добавление элемента на заданную позицию

template<typename T>

void Vector<T>::addAtPosition(size\_t position, const T& element)

{

if (position <= container.size())

{

container.insert(container.begin() + position, element);

}

else

{

std::cout << "Ошибка: позиция выходит за границы контейнера!" << std::endl;

}

}

// Удаление элемента из контейнера

template<typename T>

void Vector<T>::removeElement(const T& element)

{

auto it = std::find(container.begin(), container.end(), element);

if (it != container.end())

{

container.erase(it);

}

else

{

std::cout << "Элемент не найден!" << std::endl;

}

}

// Вычисление разницы между максимальным и минимальным элементами

template<typename T>

void Vector<T>::calculateDifference()

{

if (container.empty())

{

std::cout << "Контейнер пуст!" << std::endl;

return;

}

T minElement = \*std::min\_element(container.begin(), container.end());

T maxElement = \*std::max\_element(container.begin(), container.end());

T difference = maxElement - minElement;

for (auto& elem : container)

{

elem -= difference;

}

}

// Печать элементов контейнера

template<typename T>

void Vector<T>::print()

{

for (const T& elem : container)

{

std::cout << elem << " ";

}

std::cout << std::endl;

}

template class Vector<int>; // Пример явной специализации для int

**IV  
Utils.cpp:**#include "Utils.h"

void addElementToQueue(PriorityQueue& pq, int key, int position)

{

pq.push(key);

}

void findAndAddElement(PriorityQueue& pq, int key, int position)

{

if (pq.find(key) != -1)

{

addElementToQueue(pq, key, position);

}

}

void findAndRemoveElement(PriorityQueue& pq, int key)

{

pq.remove(key);

}

void calculateDifference(PriorityQueue& pq)

{

pq.process();

} **PriorityQueue.cpp:**

#include "PriorityQueue.h"

#include <algorithm>

void PriorityQueue::push(int value)

{

data.push\_back(value);

std::push\_heap(data.begin(), data.end());

}

void PriorityQueue::pop()

{

std::pop\_heap(data.begin(), data.end());

data.pop\_back();

}

int PriorityQueue::top() const

{

if (isEmpty())

{

throw std::runtime\_error("Queue is empty");

}

return data.front();

}

bool PriorityQueue::isEmpty() const

{

return data.empty();

}

int PriorityQueue::find(int key) const

{

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

return (it != data.end()) ? std::distance(data.begin(), it) : -1; // Возвращает индекс или -1

}

void PriorityQueue::remove(int key)

{

auto it = std::remove(data.begin(), data.end(), key);

if (it != data.end())

{

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

std::make\_heap(data.begin(), data.end());

}

}

void PriorityQueue::process()

{

if (data.empty()) return;

int max = \*std::max\_element(data.begin(), data.end());

int min = \*std::min\_element(data.begin(), data.end());

int diff = max - min;

for (auto& elem : data)

{

elem -= diff;

}

}

**V  
Vector.cpp:**

#include "Vector.h"

template<typename T>

void Vector<T>::add(const T& element)

{

container.push(element);

}

template<typename T>

void Vector<T>::remove(const T& element)

{

// Создание временной очереди

std::priority\_queue<T> temp;

while (!container.empty())

{

T current = container.top();

container.pop();

if (current != element)

{

temp.push(current);

}

}

container = std::move(temp);

}

template<typename T>

void Vector<T>::findAndInsert(const T& key, const T& newElement, size\_t position)

{

std::priority\_queue<T> temp;

size\_t currentPosition = 0;

bool inserted = false;

while (!container.empty())

{

T current = container.top();

container.pop();

if (current == key && !inserted && currentPosition == position)

{

temp.push(newElement);

inserted = true;

}

temp.push(current);

currentPosition++;

}

// Если элемент не найден, добавим новый элемент в конец

if (!inserted)

{

temp.push(newElement);

}

container = std::move(temp);

}

template<typename T>

void Vector<T>::findAndRemove(const T& key)

{

remove(key);

}

template<typename T>

void Vector<T>::subtractExtremes()

{

if (container.empty()) return;

T maxElement = container.top();

container.pop(); // Удаляем максимальный элемент

T minElement = maxElement; //Предполагаем, что одновременно максимальный и минимальный - одинаковые

std::priority\_queue<T> temp;

while (!container.empty())

{

T current = container.top();

container.pop();

if (current < minElement)

{

minElement = current;

}

temp.push(current);

}

// Вычисляем разницу

T difference = maxElement - minElement;

// Восстанавливаем очередь и вычитаем разницу из каждого элемента

container = std::move(temp);

temp = std::priority\_queue<T>();

while (!container.empty())

{

temp.push(container.top() - difference);

container.pop();

}

container = std::move(temp);

}

template<typename T>

std::vector<T> Vector<T>::getElements() const

{

std::priority\_queue<T> temp = container;

std::vector<T> elements;

while (!temp.empty())

{

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

temp.pop();

}

return elements;

}

// Не забываем явным образом инстанцировать шаблон

template class Vector<int>; // Или другой желаемый тип

**Функция main**

**I**

#include <iostream>

#include <vector>

#include "operations.h"

#include "container.h"

int main()

{

setlocale(LC\_ALL, "Russian");

std::vector<double> container;

// Заполнение контейнера

fillContainer(container, 10); // Заполнение 10 элементами

std::cout << "Элементы контейнера: ";

for (const auto& elem : container)

{

std::cout << elem << " ";

}

std::cout << std::endl;

// Задание 3

addElementAtPosition(container, 5.0, 2); // Добавление 5.0 на позицию 2

// Задание 4

removeElementByValue(container, 5.0); // Удаление элемента 5.0

// Задание 5

calculateDifference(container); // Вычисление разности

// Вывод конечного состояния контейнера

std::cout << "Элементы контейнера: ";

for (const auto& elem : container)

{

std::cout << elem << " ";

}

std::cout << std::endl;

return 0;

}

**II**

#include <iostream>

#include "Container.h"

#include "functions.h"

int main()

{

Container container;

setlocale(LC\_ALL, "Russian");

// Пример заполнения контейнера

container.addElement(Money(10.0));

container.addElement(Money(20.0));

container.addElement(Money(30.0));

// Вывод элементов в контейнере

for (const auto& elem : container.getElements())

{

std::cout << elem << " ";

}

std::cout << std::endl;

// Выполнение заданий

task3(container, Money(20.0), Money(25.0)); // Задание 3

task4(container, Money(20.0)); // Задание 4

task5(container); // Задание 5

// Вывод элементов в контейнере

for (const auto& elem : container.getElements())

{

std::cout << elem << " ";

}

std::cout << std::endl;

return 0;

}

**III**

#include <iostream>

#include "Vector.h"

int main()

{

Vector<int> myVector;

// Заполнение контейнера элементами

myVector.addElement(10);

myVector.addElement(20);

myVector.addElement(30);

myVector.print();

// Задание 3: Найти элемент с заданным ключом и добавить его на заданную позицию

myVector.addAtPosition(1, 20); // Добавление 20 на позицию 1

myVector.print();

// Задание 4: Найти элемент с заданным ключом и удалить его из контейнера

myVector.removeElement(20); // Удаление 20

myVector.print();

// Задание 5: Найти разницу между максимальным и минимальным элементами

myVector.calculateDifference();

myVector.print();

return 0;

}

**IV**#include <iostream>

#include "PriorityQueue.h"

#include "Utils.h"

int main()

{

PriorityQueue pq;

setlocale(LC\_ALL, "Russian");

// Добавление элементов

pq.push(10);

pq.push(20);

pq.push(30);

std::cout << "Верхний элемент: " << pq.top() << std::endl;

// Задание 3: Найти элемент и добавить его

findAndAddElement(pq, 20, 1);

// Задание 4: Найти элемент и удалить его

findAndRemoveElement(pq, 20);

// Задание 5: Процесс для разницы

calculateDifference(pq);

// Печать элементов:

while (!pq.isEmpty())

{

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

pq.pop();

}

std::cout << std::endl;

return 0;

}

**V**#include <iostream>

#include "Vector.h"

int main()

{

setlocale(LC\_ALL, "Russian");

Vector<int> v;

v.add(5);

v.add(15);

v.add(10);

std::cout << "Элементы после добавления: " << std::endl;

for (const auto& elem : v.getElements())

{

std::cout << elem << " ";

}

std::cout << std::endl;

v.findAndInsert(10, 7, 1);

std::cout << "После установки 7 перед 10: " << std::endl;

for (const auto& elem : v.getElements())

{

std::cout << elem << " ";

}

std::cout << std::endl;

v.findAndRemove(15);

std::cout << "После удаления 15: " << std::endl;

for (const auto& elem : v.getElements())

{

std::cout << elem << " ";

}

std::cout << std::endl;

v.subtractExtremes();

std::cout << "После вычитания экстремумов: " << std::endl;

for (const auto& elem : v.getElements())

{

std::cout << elem << " ";

}

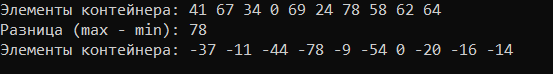
std::cout << std::endl;

return 0;

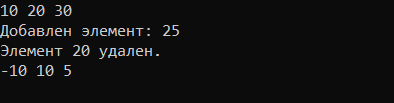
}

Результат работы:

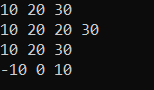
I



II



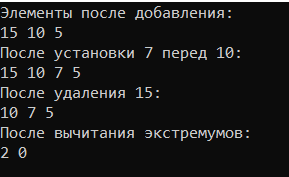
III



IV



V



Контрольные вопросы:

1. Из каких частей состоит библиотека STL?

2. Какие типы контейнеров существуют в STL?

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

4. Что представляет собой итератор?

5. Какие операции можно выполнять над итераторами?

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

7. Какие типы итераторов существуют?

8. Перечислить операции и методы общие для всех контейнеров.

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

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

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

12. Перечислить методы, которые поддерживает последовательный контейнер vector.

13. Перечислить методы, которые поддерживает последовательный контейнер list.

14. Перечислить методы, которые поддерживает последовательный контейнер deque.

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

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

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

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

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

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

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

22. Что представляют собой адаптеры контейнеров?

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

24. Перечислить методы, которые поддерживает контейнер stack.

25. Перечислить методы, которые поддерживает контейнер queue.

26. Чем отличаются друг от другa контейнеры queue и priority queue?

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

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

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

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

Ответы:

1. **Из каких частей состоит библиотека STL?**  
   STL состоит из двух основных частей: набора контейнерных классов и набора обобщенных алгоритмов.
2. **Какие типы контейнеров существуют в STL?**  
   Контейнеры STL делятся на последовательные и ассоциативные. Базовые последовательные контейнеры: векторы (vector), списки (list) и двусторонние очереди (deque).
3. **Что нужно сделать для использования контейнера STL в своей программе?**  
   Необходимо включить соответствующий заголовочный файл, например, #include <vector> для векторов.
4. **Что представляет собой итератор?**  
   Итератор — это обобщение концепции указателей, ссылающееся на элементы контейнера. Итераторы можно инкрементировать и разыменовывать.
5. **Какие операции можно выполнять над итераторами?**  
   Можно выполнять разыменование, присваивание, сравнение на равенство и неравенство, а также инкрементирование.
6. **Каким образом можно организовать цикл для перебора контейнера с использованием итератора?**  
   Используется следующая форма: for (i = first; i != last; ++i), где first и last — итераторы, указывающие на начало и конец контейнера.
7. **Какие типы итераторов существуют?**  
   Существуют входные, выходные, прямые, двунаправленные и итераторы произвольного доступа.
8. **Перечислить операции и методы общие для всех контейнеров.**  
   Общие методы включают begin(), end(), insert(), erase(), size(), empty().
9. **Какие операции являются эффективными для контейнера vector? Почему?**  
   Эффективны операции индексации [] и at(), так как они обеспечивают произвольный доступ к элементам. Неэффективны вставка и удаление элементов, кроме последнего.
10. **Какие операции являются эффективными для контейнера list? Почему?**  
    Эффективны вставка и удаление элементов в любой позиции, так как список организован как двусвязный.
11. **Какие операции являются эффективными для контейнера deque? Почему?**  
    Эффективны вставка и удаление как с начала, так и с конца, так как элементы хранятся в непрерывной области памяти.
12. **Перечислить методы, которые поддерживает последовательный контейнер vector.**  
    Методы включают push\_back(), insert(), erase(), size(), empty(), operator[], at().
13. **Перечислить методы, которые поддерживает последовательный контейнер list.**  
    Методы включают push\_back(), push\_front(), insert(), erase(), size(), empty().
14. **Перечислить методы, которые поддерживает последовательный контейнер deque.**  
    Методы включают push\_back(), push\_front(), insert(), erase(), size(), empty().
15. **Задан контейнер vector. Как удалить из него элементы со 2 по 5?**  
    Используйте v.erase(v.begin() + 1, v.begin() + 5);.
16. **Задан контейнер vector. Как удалить из него последний элемент?**  
    Используйте v.pop\_back();.
17. **Задан контейнер list. Как удалить из него элементы со 2 по 5?**  
    Используйте list.erase(++list.begin(), ++(++list.begin(), ++list.begin(), ++list.begin())); (или аналогичный подход).
18. **Задан контейнер list. Как удалить из него последний элемент?**  
    Используйте list.pop\_back();.
19. **Задан контейнер deque. Как удалить из него элементы со 2 по 5?**  
    Используйте d.erase(d.begin() + 1, d.begin() + 5);.
20. **Задан контейнер deque. Как удалить из него последний элемент?**  
    Используйте d.pop\_back();.
21. **Написать функцию для печати последовательного контейнера с использованием итератора.**

template<class T>

void print(const char\* string, T& C)

{

typename T::iterator p = C.begin();

cout << string << endl;

if (C.empty())

cout << "\nEmpty!!!\n";

else

for (; p != C.end(); p++)

cout << \*p << " ";

cout << "\n";

}

1. **Что представляют собой адаптеры контейнеров?**  
   Адаптеры контейнеров — это специализированные последовательные контейнеры (например, стек, очередь и очередь с приоритетами), которые реализованы на основе базовых контейнеров, таких как deque или list. Они не являются самостоятельными классами, а переопределяют интерфейс класса-прототипа, ограничивая доступные методы.
2. **Чем отличаются друг от друга объявления stack<int> s и stack<int, list<int>> s?**  
   Объявление stack<int> s создает стек на базе двусторонней очереди (по умолчанию), а stack<int, list<int>> s создает стек на базе списка. Это позволяет выбирать, на каком базовом контейнере будет реализован стек.
3. **Перечислить методы, которые поддерживает контейнер stack.**

* push() — добавление элемента в стек.
* pop() — удаление элемента из стека.
* top() — получение текущего элемента стека.
* empty() — проверка, пуст ли стек.
* size() — получение размера стека.

1. **Перечислить методы, которые поддерживает контейнер queue.**

* push() — добавление элемента в очередь.
* pop() — удаление элемента из начала очереди.
* front() — получение первого элемента очереди.
* back() — получение последнего элемента очереди.
* empty() — проверка, пуста ли очередь.
* size() — получение размера очереди.

1. **Чем отличаются друг от друга контейнеры queue и priority queue?**  
   Контейнер queue реализует стандартную очередь, где элементы извлекаются в порядке их добавления (FIFO), тогда как priority queue извлекает элементы в порядке их приоритета, выбирая максимальный элемент из хранимых в контейнере.
2. **Задан контейнер stack. Как удалить из него элемент с заданным номером?**  
   В стандартном стеке нет прямой возможности удалить элемент по индексу, так как он не поддерживает произвольный доступ. Для удаления элемента с заданным номером нужно будет создать временный стек, переместить элементы, пропуская нужный, и затем вернуть оставшиеся элементы обратно в оригинальный стек.
3. **Задан контейнер queue. Как удалить из него элемент с заданным номером?**  
   Как и в случае со стеком, стандартная очередь не поддерживает произвольный доступ. Для удаления элемента с заданным номером нужно будет создать временную очередь, переместить элементы, пропуская нужный, и вернуть оставшиеся элементы обратно в оригинальную очередь.
4. **Написать функцию для печати контейнера stack с использованием итератора.**  
   Функция для печати стека может быть аналогична функции для печати других контейнеров, но с учетом, что стек не поддерживает итераторы. Придется использовать временный контейнер для вывода:

void printStack(stack<int> s)

{

while (!s.empty())

{

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

s.pop();

}

cout << "\n";

}

1. **Написать функцию для печати контейнера queue с использованием итератора.**  
   Аналогично, для очереди можно использовать временную очередь:

void printQueue(queue<int> q)

{

while (!q.empty())

{

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

q.pop();

}

cout << "\n";

}

Ссылка:

**https://github.com/LeonidZhir/-**