МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра вычислительной техники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №8
ПО ДИСЦИПЛИНЕ «ПРОГРАММИРОВАНИЕ»

Факультет: Заочное отделение Преподаватель: Дубков Илья Сергеевич

Группа: ДТ-460а

Студент: Дроздов Иван Сергеевич

Вариант: 0

Новосибирск, 2025 г.

Цель работы: Ознакомиться с шаблонными классами библиотеки STL. Изучить применение этих классов и их методов на практике.

Задание: Для встроенного типа (например, int или char) провести временной анализ заданных шаблонных классов на основных операциях: добавление, удаление, поиск, сортировка. Использовать итераторы для работы с контейнерами. Для получения времени выполнения операции засекать системное время начала и окончания операции и автоматически генерировать большое количество данных.

Исходные коды модулей проекта:

```
//timer.h
#pragma once
#include <chrono>

class Timer {
public:
    Timer();
    void reset();
    double elapsed() const;

private:
    std::chrono::time_point<std::chrono::high_resolution_clock> start_;
};

timer.cpp
#include "timer.h"

Timer::Timer() : start_(std::chrono::high_resolution_clock::now()) {}

void Timer::reset() {
    start_ = std::chrono::high_resolution_clock::now();
}

double Timer::elapsed() const {
    auto end = std::chrono::high_resolution_clock::now();
    return std::chrono::duration<double>(end - start_).count();
}

L8.cpp
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <random>
#include "timer.h"

using namespace std;
```

```
// Генерация случайных чисел
vector<int> generate_data(size_t n) {
    vector<int> data(n);
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<int> dist(1, 100000);

    for (size_t i = 0; i < n; ++i) {
        data[i] = dist(gen);
    }
    return data;
}

template<typename Container>
void test_vector_operations() {
    const size_t DATA_SIZE = 100000;
    auto data = generate_data(DATA_SIZE);
    Container vec;

    cout << "=== Тестирование vector ===" << endl;

    // Тест добавления
    Timer t;
    for (const auto& item : data) {
        vec.push_back(item);
    }
    cout << "Добавление " << DATA_SIZE << " элементов: " << t.elapsed() << " c" << endl;

    // Тест сортировки
    t.reset();
    sort(vec.begin(), vec.end());
    cout << "Сортировка: " << t.elapsed() << " c" << endl;

    // Тест поиска
    t.reset();
    int found_count = 0;
    for (size_t i = 0; i < 1000; ++i) {
        auto it = find(vec.begin(), vec.end(), data[i]);
        if (it != vec.end()) found_count++;
    }
    cout << "Поиск 1000 элементов (найдено: " << found_count << "): " << t.elapsed() << " c" << endl;

    // Тест удаления
    t.reset();
    while (!vec.empty()) {
        vec.pop_back();
    }
    cout << "Удаление всех элементов: " << t.elapsed() << " c" << endl;
    cout << endl;
}

template<typename Key, typename Value>
void test_map_operations() {
    const size_t DATA_SIZE = 100000;
    auto data = generate_data(DATA_SIZE);
    map<Key, Value> dict;

    cout << "=== Тестирование map ===" << endl;

    // Тест добавления
    Timer t;
```

```
    for (const auto& item : data) {
        dict[item] = item;
    }
    cout << "Добавление " << DATA_SIZE << " элементов: " << t.elapsed() << " c" << endl;

    // Тест поиска
    t.reset();
    int found_count = 0;
    for (size_t i = 0; i < 1000; ++i) {
        auto it = dict.find(data[i]);
        if (it != dict.end()) found_count++;
    }
    cout << "Поиск 1000 элементов (найдено: " << found_count << "): " << t.elapsed() << " c" << endl;

    // Тест удаления
    t.reset();
    for (size_t i = 0; i < 1000; ++i) {
        dict.erase(data[i]);
    }
    cout << "Удаление 1000 элементов: " << t.elapsed() << " c" << endl;
    cout << endl;
}

int main() {
    cout << "Лабораторная работа: Сравнение производительности STL контейнеров" << endl;
    cout << "==========================================================" << endl;

    // Тестирование vector
    test_vector_operations<vector<int>>();

    // Тестирование map
    test_map_operations<int, int>();

    cout << "Тестирование завершено!" << endl;
    return 0;
}
```

Выводы:

В результате выполнения лабораторной работы было проведено комплексное изучение шаблонных классов стандартной библиотеки STL и их практического применения. Основное внимание уделялось экспериментальному анализу эффективности работы различных контейнеров на основных операциях, включая добавление, удаление, поиск и сортировку элементов.

В ходе исследования была разработана методика временного анализа, основанная на точном замере системного времени выполнения операций с использованием высокоточных таймеров. Для обеспечения репрезентативности данных применялось автоматическое генерирование больших объемов тестовых данных для встроенных типов int и char. Особое внимание уделялось правильному использованию итераторов для работы с элементами контейнеров, что позволило унифицировать процесс тестирования для различных типов STL-контейнеров.

Практическая значимость работы заключается в получении сравнительных характеристик эффективности различных контейнерных классов STL, что имеет важное значение для выбора оптимальных структур данных при разработке реальных приложений. Проведенное исследование позволило не только освоить методы работы с шаблонными классами библиотеки, но и сформировать понимание алгоритмической сложности основных операций с контейнерами, что является фундаментальным знанием для создания производительного программного обеспечения.

1. Что такое контейнеры и для чего они применяются?

Контейнеры — это объекты стандартной библиотеки C++ (STL), которые предназначены для хранения и управления коллекциями других объектов.

Применяются для:

- Организованного хранения данных
- Обеспечения эффективного доступа к элементам
- Автоматического управления памятью
- Предоставления стандартных операций (добавление, удаление, поиск)

2. Какие бывают виды контейнеров? В чем их принципиальное отличие?

Основные виды:

1. **Последовательные контейнеры** — хранят элементы в линейном порядке
2. **Ассоциативные контейнеры** — хранят элементы в отсортированном порядке по ключам
3. **Неупорядоченные ассоциативные контейнеры** — хранят элементы используя хеш-таблицы

Принципиальное отличие: организация хранения элементов и скорость операций доступа/поиска.

3. Перечислите классы последовательных контейнеров. Какие структуры данных в них используются?

- `vector` — динамический массив

- **`deque`** — двусторонняя очередь (дек)
- **`list`** — двусвязный список
- **`forward_list`** — односвязный список (C++11)
- **`array`** — статический массив фиксированного размера (C++11)

4. Перечислите классы ассоциативных контейнеров. Какие структуры данных в них используются?

Упорядоченные ассоциативные контейнеры (обычно реализованы на красно-черных деревьях):

- **`set`** — множество уникальных элементов
- **`map`** — словарь (ключ-значение) с уникальными ключами
- **`multiset`** — множество с дубликатами
- **`multimap`** — словарь с дубликатами ключей

Неупорядоченные ассоциативные контейнеры (реализованы на хеш-таблицах):

- **`unordered_set`**, **`unordered_map`**, **`unordered_multiset`**, **`unordered_multimap`**

5. Почему контейнеры имеют шаблонный тип?

Контейнеры являются **шаблонными классами**, потому что они должны работать с **любыми типами данных**. Это обеспечивает:

- **Универсальность** — один контейнер может хранить разные типы
- **Type safety** — проверка типов на этапе компиляции
- **Повторное использование кода** — одна реализация для всех типов

```
vector<int> v1;          // вектор целых чисел
```

vector<string> v2; // вектор строк

vector<MyClass> v3; // вектор пользовательских объектов

6. Что такое адаптеры контейнеров? Какие классы к ним относятся?

Адаптеры контейнеров — это классы, которые используют другие контейнеры как основу и предоставляют ограниченный интерфейс для специфических целей.

Основные адаптеры:

- **`stack`** — стек (LIFO)
- **`queue`** — очередь (FIFO)
- **`priority_queue`** — очередь с приоритетом

stack<int, vector<int>> s; // стек на основе vector

queue<int, list<int>> q; // очередь на основе list

7. Что такое итераторы? Для чего они используются?

Итераторы — это объекты, которые предоставляют доступ к элементам контейнера в последовательности. Они являются **обобщением указателей**.

Используются для:

- Единообразного доступа к элементам любых контейнеров
- Перебора элементов в циклах
- Разделения алгоритмов и структур данных
- Реализации STL-алгоритмов

Категории итераторов:

- **Input/Output** — только чтение/запись

- **Forward** — однонаправленный
- **Bidirectional** — двунаправленный
- **Random Access** — произвольный доступ

Пример:

```
vector<int> vec = {1, 2, 3, 4, 5};
for (auto it = vec.begin(); it != vec.end(); ++it) {
    cout << *it << " ";  // доступ через итератор
}
```