МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

КАФЕДРА ИИТ

ОТЧЁТ

по лабораторной работе №6

**«СТАНДАРТНАЯ БИБЛИОТЕКА ШАБЛОНОВ»**

Выполнил:

студент 2 курса

группы ПО-9

Мисиюк Алексей Сергеевич

Проверил:

Козик И. Д.

**Порядок выполнения работы.**

Написать и отладить три программы. Первая программа демонстрирует использование контейнерных классов для хранения встроенных типов данных.

Вторая программа демонстрирует использование контейнерных классов для хранения пользовательских типов данных.

Третья программа демонстрирует использование алгоритмов STL.

В программе № 1 выполнить следующее:

1. Создать объект-контейнер в соответствии с вариантом задания и заполнить его данными, тип которых определяется вариантом задания.

2. Просмотреть контейнер.

3. Изменить контейнер, удалив из него одни элементы и заменив другие.

4. Просмотреть контейнер, используя для доступа к его элементам итераторы.

5. Создать второй контейнер этого же класса и заполнить его данными того же типа, что и первый контейнер.

6. Изменить первый контейнер, удалив из него n элементов после заданного и добавив затем в него все элементы из второго контейнера.

7. Просмотреть первый и второй контейнеры.

В программе № 2 выполнить то же самое, но для данных пользовательского типа.

В программе № 3 выполнить следующее:

1. Создать контейнер, содержащий объекты пользовательского типа. Тип контейнера выбирается в соответствии с вариантом задания.

2. Отсортировать его по убыванию элементов.

3. Просмотреть контейнер.

4. Используя подходящий алгоритм, найти в контейнере элемент, удовлетворяющий заданному условию.

5. Переместить элементы, удовлетворяющие заданному условию в другой (предварительно пустой) контейнер. Тип второго контейнера определяется вариантом задания.

6. Просмотреть второй контейнер.

7. Отсортировать первый и второй контейнеры по возрастанию элементов.

8. Просмотреть их.

9. Получить третий контейнер путем слияния первых двух.

10. Просмотреть третий контейнер.

11 .Подсчитать, сколько элементов, удовлетворяющих заданному условию, содержит третий контейнер.

12.Определить, есть ли в третьем контейнере элемент, удовлетворяющий заданному условию

**Вариант 15**

|  |  |  |  |
| --- | --- | --- | --- |
| 15 | queue | map | int |

**Код программы**

#include <iostream>

#include <queue>

#include <map>

#include <string>

#include <algorithm>

#include <vector>

struct Person {

std::string name;

int age;

Person(const std::string& \_name, int \_age) : name(\_name), age(\_age) {}

Person() {}

};

int main()

{

std::cout << std::endl << "Программа #1" << std::endl << std::endl;

// Создание и заполнение первого контейнера типа queue<int>

std::queue<int> firstQueue;

firstQueue.push(10);

firstQueue.push(20);

firstQueue.push(30);

// Просмотр первого контейнера

std::cout << "Первый контейнер (queue): ";

std::queue<int> tempQueue = firstQueue;

while (!tempQueue.empty()) {

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

tempQueue.pop();

}

std::cout << std::endl;

// Изменение первого контейнера, удаляя один элемент и добавляя другой

firstQueue.push(40); // Добавляем элемент 40

firstQueue.pop(); // Удаляем первый элемент

// Просмотр первого контейнера с использованием итератора

std::cout << "Первый контейнер после изменения: ";

tempQueue = firstQueue; // Создаем временную копию первого контейнера

while (!tempQueue.empty()) {

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

tempQueue.pop();

}

std::cout << std::endl;

// Создание и заполнение второго контейнера типа map<int, int>

std::map<int, int> secondMap;

secondMap[1] = 100;

secondMap[2] = 200;

secondMap[3] = 300;

// Изменение первого контейнера, удаляя n элементов и добавляя все элементы из второго контейнера

int n = 2; // Заданный элемент

while (!firstQueue.empty() && n > 0) {

firstQueue.pop();

n--;

}

for (const auto& pair : secondMap) {

firstQueue.push(pair.second);

}

// Просмотр первого контейнера после изменения

std::cout << "Первый контейнер после изменения и добавления элементов из второго контейнера: ";

tempQueue = firstQueue; // Обновляем временную копию первого контейнера

while (!tempQueue.empty()) {

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

tempQueue.pop();

}

std::cout << std::endl;

// Просмотр второго контейнера типа map

std::cout << "Второй контейнер (map): ";

for (const auto& pair : secondMap) {

std::cout << pair.second << " ";

}

std::cout << std::endl;

std::cout << std::endl << "Программа #2" << std::endl << std::endl;

std::queue<Person> firstQueue2;

firstQueue2.push(Person("Alice", 25));

firstQueue2.push(Person("Bob", 30));

firstQueue2.push(Person("Charlie", 35));

// Просмотр первого контейнера

std::cout << "Первый контейнер (queue): \n";

std::queue<Person> tempQueue2 = firstQueue2;

while (!tempQueue2.empty()) {

Person person = tempQueue2.front();

std::cout << "Name: " << person.name << ", Age: " << person.age << std::endl;

tempQueue2.pop();

}

// Изменение первого контейнера, удаляя один элемент и добавляя другой

firstQueue2.push(Person("Dave", 40)); // Добавляем новый элемент

firstQueue2.pop(); // Удаляем первый элемент

// Просмотр первого контейнера с использованием итератора

std::cout << "Первый контейнер после изменения: \n";

tempQueue2 = firstQueue2; // Создаем временную копию первого контейнера

while (!tempQueue2.empty()) {

Person person = tempQueue2.front();

std::cout << "Name: " << person.name << ", Age: " << person.age << std::endl;

tempQueue2.pop();

}

// Создание и заполнение второго контейнера типа map<int, Person>

std::map<int, Person> secondMap2;

secondMap2[1] = Person("Eve", 45);

secondMap2[2] = Person("Frank", 50);

secondMap2[3] = Person("Grace", 55);

// Изменение первого контейнера, удаляя n элементов и добавляя все элементы из второго контейнера

n = 1;

while (!firstQueue2.empty() && n > 0) {

firstQueue2.pop();

n--;

}

for (const auto& pair : secondMap2) {

firstQueue2.push(pair.second);

}

// Просмотр первого контейнера после изменения

std::cout << "Первый контейнер после изменения и добавления элементов из второго контейнера: \n";

tempQueue2 = firstQueue2; // Обновляем временную копию первого контейнера

while (!tempQueue2.empty()) {

Person person = tempQueue2.front();

std::cout << "Name: " << person.name << ", Age: " << person.age << std::endl;

tempQueue2.pop();

}

// Просмотр второго контейнера типа map

std::cout << "Второй контейнер (map): \n";

for (const auto& pair : secondMap2) {

Person person = pair.second;

std::cout << "Name: " << person.name << ", Age: " << person.age << std::endl;

}

std::cout << std::endl << "Программа #3" << std::endl << std::endl;

// Шаг 1: Создание контейнера

std::queue<int> container;

// Заполнение контейнера элементами

container.push(5);

container.push(10);

container.push(2);

container.push(8);

container.push(3);

// Шаг 2: Сортировка по убыванию

std::vector<int> sorted\_container;

while (!container.empty()) {

sorted\_container.push\_back(container.front());

container.pop();

}

std::sort(sorted\_container.rbegin(), sorted\_container.rend());

// Шаг 3: Просмотр контейнера

std::cout << "Контейнер: ";

for (const auto& element : sorted\_container) {

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

}

std::cout << std::endl;

// Шаг 4: Поиск элемента, удовлетворяющего условию

int target\_element = 8;

auto found\_element = std::find(sorted\_container.begin(), sorted\_container.end(), target\_element);

if (found\_element != sorted\_container.end()) {

std::cout << "Найден элемент: " << \*found\_element << std::endl;

}

else {

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

}

// Шаг 5: Перемещение элементов, удовлетворяющих условию, во второй контейнер

std::vector<int> second\_container;

std::copy\_if(sorted\_container.begin(), sorted\_container.end(), std::back\_inserter(second\_container),

[target\_element](int element) { return element == target\_element; });

// Шаг 6: Просмотр второго контейнера

std::cout << "Второй контейнер: ";

for (const auto& element : second\_container) {

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

}

std::cout << std::endl;

// Шаг 7: Отсортировать первый и второй контейнеры по возрастанию элементов

std::sort(sorted\_container.begin(), sorted\_container.end());

std::sort(second\_container.begin(), second\_container.end());

// Шаг 8: Просмотр отсортированных контейнеров

std::cout << "Первый контейнер (по возрастанию): ";

for (const auto& element : sorted\_container) {

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

}

std::cout << std::endl;

std::cout << "Второй контейнер (по возрастанию): ";

for (const auto& element : second\_container) {

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

}

std::cout << std::endl;

// Шаг 9: Получение третьего контейнера путем объединения первых двух

std::vector<int> third\_container;

std::merge(sorted\_container.begin(), sorted\_container.end(),

second\_container.begin(), second\_container.end(),

std::back\_inserter(third\_container));

// Шаг 10: Просмотр третьего контейнера

std::cout << "Третий контейнер: ";

for (const auto& element : third\_container) {

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

}

std::cout << std::endl;

// Шаг 11: Подсчет элементов, удовлетворяющих условию, в третьем контейнере

int count = std::count\_if(third\_container.begin(), third\_container.end(),

[target\_element](int element) { return element == target\_element; });

std::cout << "Количество элементов, удовлетворяющих условию, в третьем контейнере: " << count << std::endl;

bool contains\_element = std::find(third\_container.begin(), third\_container.end(), target\_element) != third\_container.end();

if (contains\_element) {

std::cout << "В третьем контейнере есть элемент, удовлетворяющий условию" << std::endl;

}

else {

std::cout << "В третьем контейнере нет элемента, удовлетворяющего условию" << std::endl;

}

return 0;

}

**Результат**