МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Чувашский государственный университет имени И.Н. Ульянова»

(ФГБОУ ВО «ЧГУ им. И.Н. Ульянова»)

Факультет информатики и вычислительной техники

Кафедра компьютерных технологий

КУРСОВОЙ ПРОЕКТ

по дисциплине «Объектно-ориентированное программирование»

на тему

РАЗРАБОТКА И РЕАЛИЗАЦИЯ КЛАССОВ НА С++. ВАРИАНТ 5.

Курсовой проект выполнил

студент КТ-43-21

Казаков А.Ю.

Научный руководитель

Ст. преподаватель

Мытникова Е. А.

Чебоксары 2023

Содержание

[ВВЕДЕНИЕ 3](#_Toc136182037)

[1. АНАЛИТИЧЕСКАЯ ЧАСТЬ 4](#_Toc136182038)

[1.1 Описание предметной области 4](#_Toc136182039)

[1.2 Описание современных подходов к решению задачи 4](#_Toc136182040)

[2. ПРОЕКТНО – КОНСТРУКТОРСКАЯ ЧАСТЬ 6](#_Toc136182041)

[3. ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ. 8](#_Toc136182042)

[3.1 Описание задач 8](#_Toc136182043)

[3.2 Решение задачи 9](#_Toc136182044)

[ЗАКЛЮЧЕНИЕ 19](#_Toc136182045)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 20](#_Toc136182046)

[ПРИЛОЖЕНИЯ 21](#_Toc136182047)

[Приложение 1 21](#_Toc136182048)

[Приложение 2 25](#_Toc136182050)

[Приложение 3 28](#_Toc136182052)

# **ВВЕДЕНИЕ**

Данная курсовая работа направлена на изучение и закрепления навыков ООП и практического использования алгоритмов и структур данных на языке С++. Объектно-ориентированный подход в последнее десятилетие стал одним из наиболее интенсивно развивающихся направлений в программировании и наиболее популярным средством разработки программного обеспечения. Наиболее распространенным объектно-ориентированным языком программирования является язык C++. Данная работа предполагает собой написание трех программ, в каждой из которых затрагиваются основные аспекты ООП, а также работа с основными структурами данных. В первой работе данными будут встроенные типы данных, во второй и третьей – пользовательские. Все три программы предполагают изучение и применение стандартной библиотеки шаблонов С++.

# **АНАЛИТИЧЕСКАЯ ЧАСТЬ**

## 1.1 Описание предметной области

STL (Standard Template Library) — это библиотека шаблонов, которая включает в себя множество контейнеров, алгоритмов и итераторов для работы с данными в C++. Она была разработана в рамках стандарта C++ и включена в него с версии C++98.

STL предоставляет удобный интерфейс для работы с данными, позволяя программисту использовать готовые контейнеры (например, векторы, списки, множества и т.д.) и алгоритмы (например, сортировка, поиск и т.д.), не заботясь о реализации этих компонентов.

Стандартная библиотека включает в себя два контейнера-класса queue и vector.

Контейнер queue (очередь) представляет собой контейнер, который работает по принципу "первым пришел - первым ушел" (FIFO - first in, first out). Это означает, что элементы добавляются в конец очереди, а извлекаются из ее начала.

Контейнер vector (динамический массив) представляет собой контейнер, который хранит элементы в непрерывной области памяти и позволяет быстрый доступ к любому элементу по индексу.

## 1.2 Описание современных подходов к решению задачи

Первая часть

Для разработки первой программы необходимо создать контейнер типа queue и vector. Для этого воспользуемся стандартной библиотекой STL.

По заданию в первой части хранимые данные это встроенный тип данных char. Для реализации перебора элементов в структурах данных используются итераторы. Библиотекой STL не предусмотрен итератор для структуры queue. Поэтому для перебора элементов будем извлекать встроенный контейнер в queue, которым является deque, проверять его на соответствие условию там, где это необходимо и после проведения всех операция удалять все элементы из queue и повторно вносить элементы из vector в queue.

Вторая часть

Для создания второй программы будем так же использовать queue и vector. Для пользовательских данных определим свой класс, в котором опишем все поля и методы, необходимые для работы.

Так же для ввода/вывода данных будем использовать потоковый ввод/вывод. Данные будем вводить из стандартного потока ввода, а выводить в стандартный поток на консоль. Для этого переопределим соответствующие операторы ( << и >> ).

Третья часть

В третьей программе будем использовать структуры данных. Которые определены и реализованы в библиотеке STL. По заданию определим два контейнера queue и vector. Создадим итераторы, там, где это можно (очередь не позволяет создать итератор). Так же воспользуемся стандартными алгоритмами поиска, сортировки, подсчета.

Алгоритм сортировки не поддерживается контейнером queue. Для реализации сортировки будем извлекать встроенный контейнер в queue, которым является deque, сортировать deque и перемещать элементы обратно в queue. Для поиска сортировки переопределим оператор сравнения “<” для проверки условия между определенным полем объекта пользовательского класса и создадим функцию для поиска сортировки по убыванию, которая будет сравнивать поля объектов пользовательского класса с помощью оператора сравнения “>”.

# **2.** **ПРОЕКТНО – КОНСТРУКТОРСКАЯ ЧАСТЬ**

Первая часть

В первой части программы сначала создадим queue<char> c\_qu;

Для перебора всех элементов и вывода их на экран получим встроенный контейнер deque и воспользуемся циклом for.

auto buff = data.\_Get\_container();

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

cout << buff[i];

}

Остальные задания по программе выполняются с помощью цикла for. Полный листинг программы предоставлен в [Приложении 1.](#_Приложение_2)

Вторая часть

Во второй части программы будем использовать те же принципы, только будем работать с пользовательским типом данных. Для этого создадим класс, описывающий тип данных Country.

class Country {

private:

string name = "Not defined";

string form\_government = "Not defined";

float area = 1;

}

Доступ к полям класса будет происходить через get/set.

Так же переопределим методы для потокового ввода/вывода.

friend ostream& operator << (ostream& os, const Country& obj)

friend istream& operator >> (istream& in, Country& obj)

Будем вводить данные построчно для каждого поля класса Country.

Так как в классе переопределены операторы ввода/вывода, то мы можем непосредственно считывать и записывать данные о Стране в поток, командами cin >> County и cout << County.

Все необходимые изменения показаны в листинге [Приложения 2](#_Приложение_2_-).

Третья часть

В третьей части в классе Country переопределим метод меньше:

friend bool operator < (const Country& lhs, const Country& rhs)

И создадим функцию compare:

bool compare(const Country& lhs, const Country& rhs) {

return lhs.get\_name() > rhs.get\_name();

}

Они пригодятся для создания функциональных объектов для алгоритмов сортировки, поиска, подсчета STL. Таким образом класс Country примет следующий вид

class Country {

public:

friend ostream& operator << (ostream& os, const Country& obj);

friend istream& operator >> (istream& in, Country& obj);

friend bool operator < (const Country& lhs, const Country& rhs);

}

Так же для queue библиотекой STL не поддерживаются алгоритмы сортировки и поиска. Для выхода из этой ситуации будем выполнять следующие действия:

1. Получим встроенный контейнер deque из queue

2. Отсортируем deque

3. Переместим все данные из deque в queue

Все необходимые изменения показаны в листинге [Приложения 3.](#_Приложение_2_-)

# **3.** **ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ.**

## 3.1 Описание задач

Описание задачи №1:

1. Создать объект-контейнер в соответствии с вариантом задания и заполнить его пользовательскими 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.

Описание задачи №2

Для второй части используем тот же алгоритм, только выводится будут пользовательские данные, а исходные данные будут взяты из стандартного потока ввода.

Описание задачи №3

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.

## 3.2 Решение задачи

Порядок решения задачи будем описывать на примере задачи №3.

Для решения задачи и написания кода будем использовать IDE Visual Studio, которая поддерживает самые современные стандарты языка С++ и STL.

В соответствии с вариантом задания требуется создать пользовательский тип данных Country с полями:

* имя - string
* форма правления – string
* площадь – float

Создание контейнера, содержащего объекты класса Country.

Создадим переменную size типа int, которая будет отвечать за размер первого контейнера, содержащего пользовательский тип данных, далее будет происходить ввод имени, формы правления и площади по каждому объекту и созданный объект будет добавляться в очередь с помощью метода push().

int size = 1;

cout << "Количество объектов: "; cin >> size;

queue<Country> c\_queue;

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

Country obj;

cin >> obj;

c\_queue.push(obj);

}

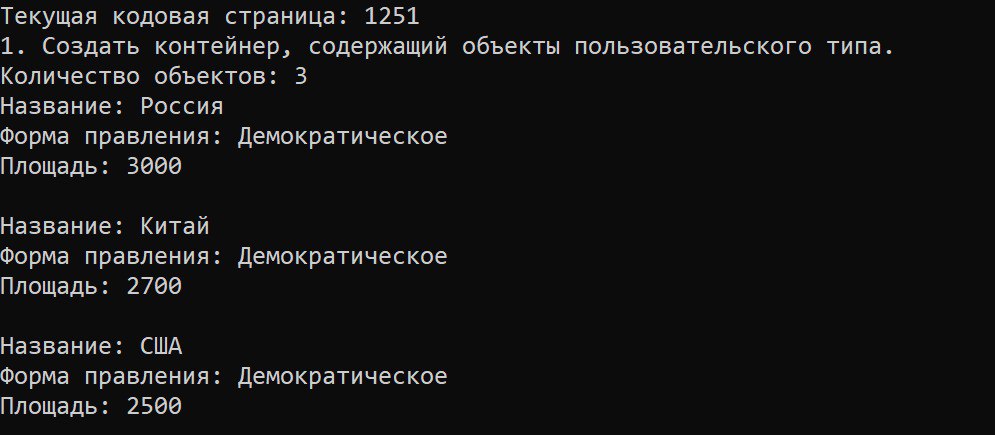


Рисунок 1 – Ввод размера контейнера и заполнение данными

Сортировка первого контейнера по убыванию и вывод содержимого контейнера в консоль.

Для операции сортировки контейнера создадим функцию sort\_containers\_desc() в которую будем передавать ссылку на контейнер. Сортировка первого контейнера по убыванию будет происходить с помощью метода sort, для этого извлечём встроенный контейнер deque и передадим методу sort итераторы на начало и конец очереди. Сортировка будет происходить с помощью функции compare, в которой будут сравниваться два входящих объекта Country по полю name через оператор “>”.

После проведения операции сортировки в извлеченном контейнере deque все объекты будут отсортированы по полю name в порядке убывания и далее требуется очистить контейнер queue и занести элементы контейнера deque в queue.

Код функции sort\_containers\_desc и сompare:

bool compare(const Country& lhs, const Country& rhs) {

return lhs.get\_name() > rhs.get\_name();

}

void sort\_containers\_desc(T\* data) {

auto buff = data->\_Get\_container();

sort(buff.begin(), buff.end(), compare);

while (data->size() != 0) data->pop();

for (auto s : buff) data->push(s);

}

Просмотр содержимого контейнера будет происходит с помощью функции print\_container(T data), куда в качестве параметра функции будет передаваться копия контейнера и будет происходить итерация и вывод информации в консоль с помощью переопределенного оператора “<<” в классе Country.

Код переопределенного оператора и функции print\_container:

Class Country { ...

friend ostream& operator << (ostream& os, const Country& obj) {

return os << "Название: " << obj.get\_name() << "\nФорма правления: " << obj.get\_form\_government() << "\nПлощадь: " << obj.get\_area() << endl;

… }}

void print\_container(T data) {

auto buff = data.\_Get\_container();

for (int i = 0; i < buff.size(); i++) {

cout << "Страна №" << i + 1 << "\n-----------" << endl;

cout << buff[i] << endl;

}}

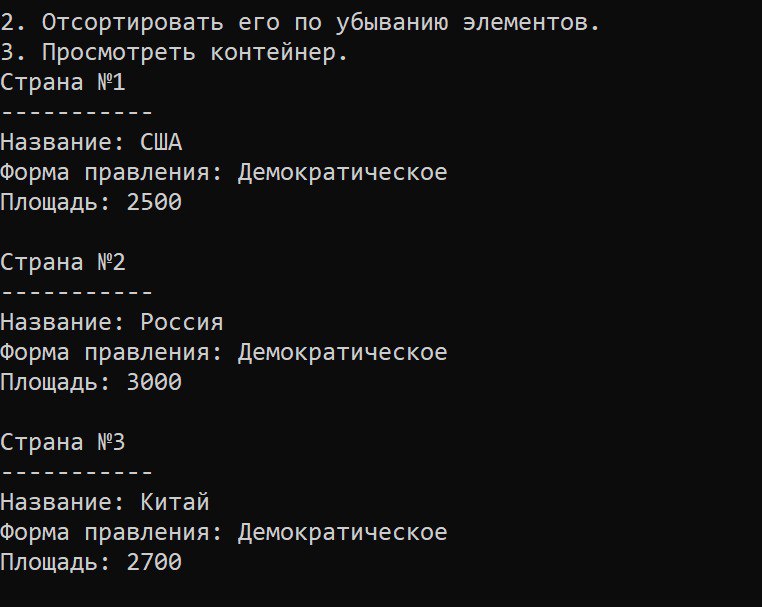


Рисунок 2 – Сортировка контейнера по убыванию

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

Поиск элемента, удовлетворяющий заданному условию, будет происходит с помощью функции find\_obj(Т data). В фунцию find\_obj будет передаваться в качестве параметра копия контейнера queue. Условие для поиска будет осуществляться по полю name каждого объекта в цикле for. Искомые данные будут вводиться с помощью стандартного потока ввода через консоль. Если искомое название будет соответствовать названию одного из объектов контейнера queue, то найденный объект будет отображен в консоли с помощью стандартного потока вывода.

Код функции find\_obj(Т data):

void find\_obj(T data) {

auto buff = data.\_Get\_container();

string search\_name = "";

cout << "Название Станы для поиска: "; cin >> search\_name;

for (auto el : buff) {

if (el.get\_name() == search\_name)

cout << el << endl;

}

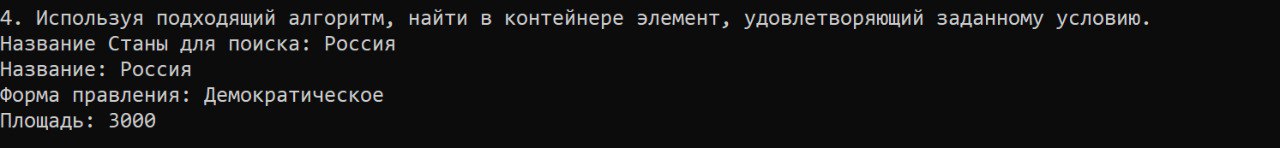


Рисунок 3 – Результат поиск объекта в соответствии заданному условию

Переместить элементы, удовлетворяющие заданному условию в другой пустой контейнер и вывод содержимого контейнера в консоль.

Для создания второго контейнера воспользуемся контейнером vector<Country> в соответствии с заданием, который будет содержать список объектов типа Country.

Для перемещения будем использовать функцию move\_obj(T1\* c1, T2\* c2). В функцию move\_obj будут передаваться в качестве параметров ссылки на контейнер queue и ссылка на второй пустой контейнер vector. Поиск нужного объекта для перемещения будет происходит по условию соответствия введенного названия перемещаемой страны через стандартный поток ввода и далее требуется проитерировать первый контейнер на соответствие искомого названия и если объект будет найден, то найденный объект будет удален из первого контейнера queue с помощью функции erase() и занесен в контейнер vector через метод push\_back().

Просмотр содержимого контейнера vector будет происходит с помощью цикла for.

Код функции move\_obj() и цикла:

void move\_obj(T1\* c1, T2\* c2) {

auto buff\_c1 = c1->\_Get\_container();

string name = "";

cout << "Название Станы для перемещения: "; cin >> name;

for (int i = 0; i < buff\_c1.size(); ++i) {

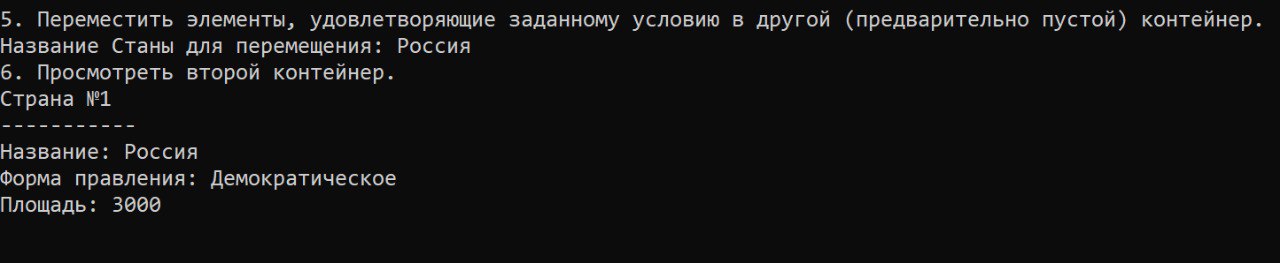
if (buff\_c1[i].get\_name() == name) {

c2->push\_back(buff\_c1[i]);

buff\_c1.erase(buff\_c1.begin() + i); }}

while (c1->size() != 0) c1->pop();

for (auto s : buff\_c1) c1->push(s);}

Рисунок 4 – Результат поиск и перемещения объекта во второй контейнер

Отсортировать первый и второй контейнеры по возрастанию элементов и вывод содержимого контейнеров в консоль.

Для операции сортировки контейнера создадим функцию sort\_container(T\* data). В функцию в качестве параметра будет передаваться ссылка на контейнер. Сортировка контейнера будет происходить с помощью метода sort, для этого извлечём встроенный контейнер deque и передадим методу sort итераторы на начало и конец очереди. Сортировка будет происходить с помощью оператора сравнения “<”, который мы переопределили в классе Country. Метод sotr() будет сортировать объекты по полю name.

Для сортировки контейнера vector не требуется создавать отдельную функцию, достаточно вызвать мето sort() передав в качестве первого параметра итератор vector.begin() и в качестве второго параметра итератор на конец очереди vector. end().

Код фукции sort\_containers и переопределение оператора сравнения в классе Country:

class Country {

friend bool operator < (const Country& lhs, const Country& rhs) {

return lhs.get\_name() < rhs.get\_name();

}}

void sort\_containers(T\* data) {

auto buff = data->\_Get\_container();

sort(buff.begin(), buff.end());

while (data->size() != 0) data->pop();

for (auto s : buff) data->push(s);

}

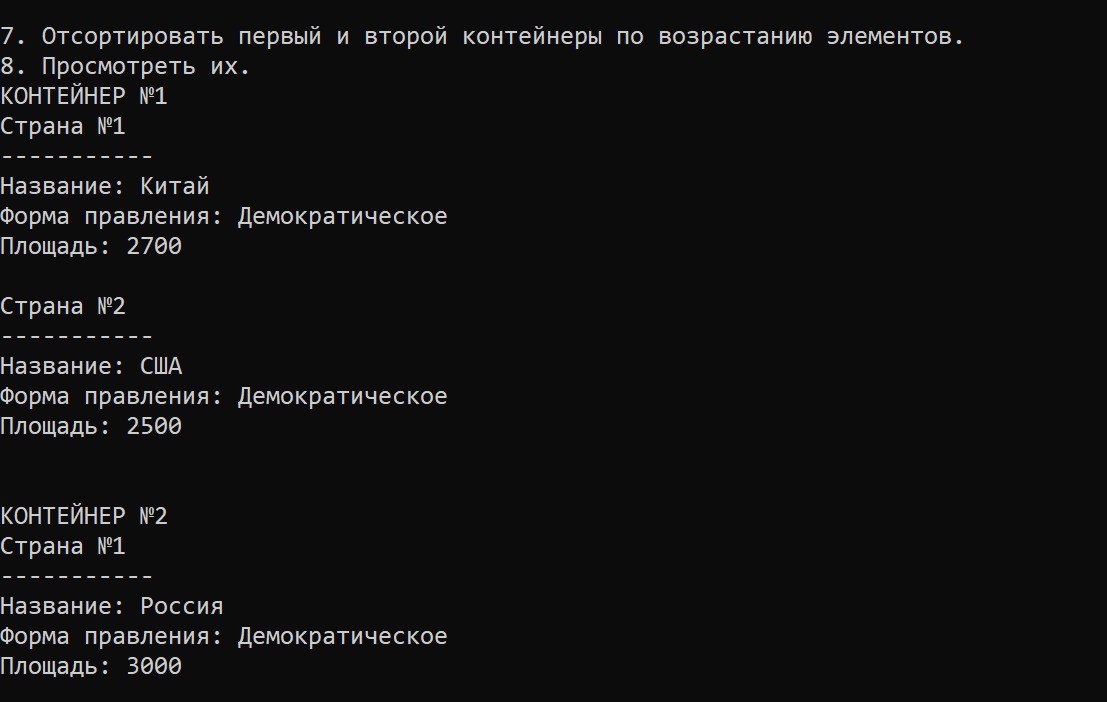


Рисунок 5 – Результат сортировки двух контейнеров

Получить третий контейнер путем слияния контейнера queue и vector

Для создания третьего контейнера воспользуемся контейнером vector, т.к. данный контейнер позволит легко добавлять и оперировать элементами в будущем.

Создадим функцию move\_obj() куда передадим первым параметром ссылку на третий контейнер vector, вторым параметром передадим ссылку на первый контейнер queue и третьим параметром ссылку на второй контейнер vector. В функции merge\_containers() получим встроенный контейнер deque из второго контейнера queue и с помощью цикла for пройдемся по контейнеру deque и во время итерации будем добавлять каждый итерируемый объект в результирующий контейнер vector с помощью метода push\_back(). Подобную операцию проведем и для второго контейнера vector.

Просотр третьего контейнера будет происходить с помощью функции print\_container().

Код функции merge\_containers:

void merge\_containers(T1\* gen, T2 c1, T3 c2)

{

auto buff\_1 = c1.\_Get\_container();

for (auto el : buff\_1) {

gen->push\_back(el);

}

for (auto el : c2) {

gen->push\_back(el);

}

}

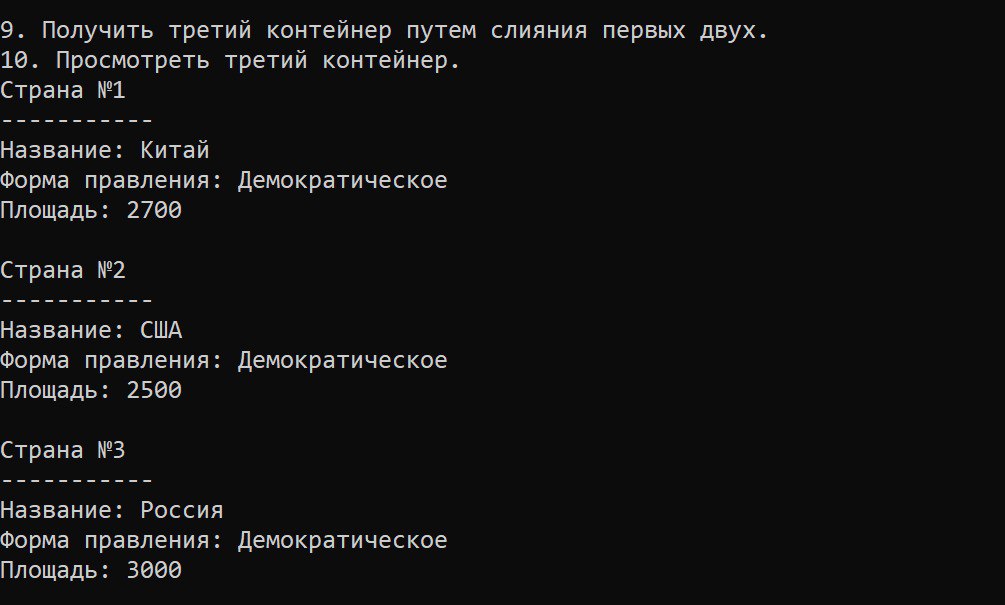


Рисунок 6 – Результат объединения двух контейнеров в третий

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

Для подсчёта элементов заданному условию создадим функцию search\_occurrences() куда передадим в качестве параметра копию третьего контейнера vector. Поиск объекта будет происходить по полю name и так же требуется создать переменную счётчик типа int, которая будет отображать количество найденных элементов заданному условию и передаваться в стандартный поток вывода в консоль.

Код функции search\_occurrences:

void search\_occurrences(vector<T> gen) {

string search\_name = "";

cout << "Название Станы для поиска: "; cin >> search\_name;

int count = 0;

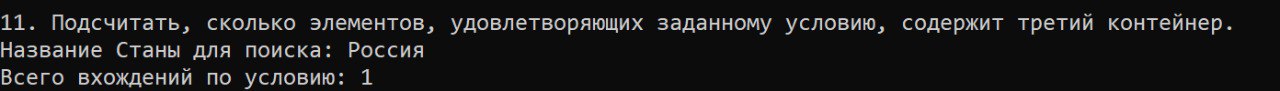
for (auto el : gen) {

if (el.get\_name() == search\_name)

count++;

cout << "Всего вхождений по условию: " << count << endl;

}

Рисунок 7 – Результат подсчёта найденных объектов заданному условию

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

Для проверки наличия искомого объекта в контейнере vector создадим функцию search\_obj() куда передадим в качестве параметра копию третьего контейнера vector. Поиск объекта будет происходить по полю namе и если объект будет найден, то функция передаст сообщение “Объект найден” в стандартный поток вывода в консоль, иначе сообщение “Вхождений не найдено”.

Код функции search\_obj:

void search\_obj(vector<T> gen) {

string search\_name = "";

cout << "Название Станы для поиска: "; cin >> search\_name;

for (auto el : gen) {

if (el.get\_name() == search\_name) {

cout << "Объект найден." << endl;

return;

}

}

cout << "Вхождений не найдено" << endl;

}

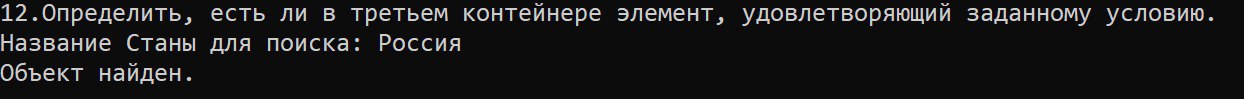


Рисунок 8 – Результат поиска наличия объекта в контейнере

# **ЗАКЛЮЧЕНИЕ**

В ходе проектной работы были созданы и отлажены три программы с помощью STL библиотеки.

В первой программе были использованы контейнеры queue и vector. В качестве содержимого контейнеров был использован встроенный тип данных char. Так же были добавлены и реализованы методы для вывода queue. Итераторы очень удобный инструмент, когда необходим перебор всех элементов структуры. Далее были протестированы методы контейнеров, результат – программа работает без ошибок и полученные данные соответствуют ожидаемым!

В ходе второй части работы был создан пользовательский класс страна и реализованы методы для работы с этим классом. В пользовательском классе Country были переопределены оператор “<<” для стандартного потока вывода и оператор “>>” стандартного потока ввода для корректного отображения результатов и ввода данных по каждому объекту класса Страна.

В третьей части были использованы итераторы, и алгоритмы для сортировки, поиска, подсчета значении в структуре данных и поиска на наличие объекта в контейнере (при определенных условиях).

# **СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

1. Джосьютис Н. М. C++. Стандартная библиотека. Для профессионалов: Пер. с англ. – СПб.: Питер, 2019
2. Страуструп Б. Язык программирования С++. 4-е изд. — М.: Бином, 2022.
3. Эффективное использование STL / Скотт Мейерс / Питер, 2020
4. C/С++. Программирование на языке высокого уровня / Т.А. Павловская — СПб.: Питер, 2020 .
5. C++ и STL: справочное руководство (Программирование на C++ с применением стандартной библиотеки шаблонов) / Дэвид Р. Мюссер, Жилмер Дж. Дердж, Атул Сейни / Вильямс, 2020
6. C++ 17 STL Стандартная библиотека шаблонов / Яцек Галовиц, СПБ. Питер 2018
7. STL – стандартная библиотека шаблонов C++: Пер. с англ. / П. Плаугер, А. Степанов, М. Ли, Д. Массер. – СПб.: БХВ-Петербург, 2018
8. STL для программистов на C++ / Леен Аммерааль /ДМК, 2020

# **ПРИЛОЖЕНИЯ**

Приложение 1

Листинг программы первой части

#include <iostream>

#include <string>

#include <queue>

#include <vector>

#include <stdlib.h>

#include <windows.h>

#include <iomanip>

#include <sstream>

using namespace std;

const string TEXT = "У лукоморья дуб зелёный; Златая цепь на дубе том: И днём и ночью кот учёный Всё ходит по цепи кругом; Идёт направо — песнь заводит, Налево — сказку говорит. Там чудеса: там леший бродит, Русалка на ветвях сидит; Там на неведомых дорожках Следы невиданных зверей; Избушка там на курьих ножках Стоит без окон, без дверей; ";

void print\_header();

void get\_cons\_buff(int&, int&);

void center\_string(string&, const size\_t&);

template<typename T>

void view\_container(T data);

template<typename T, typename T2>

void replace\_index(T new\_value, int pos, T2\* data);

int main()

{

setlocale(LC\_ALL, "Russian");

print\_header();

int size = 1;

queue<char> c\_qu;

vector<char> c\_vec;

do {

cout << "Длина контейнера: "; cin >> size;

if (size > TEXT.length()) {

cout << "Длина контейнера не должна превышать " << TEXT.length() << " символов" << endl;

}

} while (size > TEXT.length());

cout << "1. Создать объект-контейнер \"queue <char>\" и заполнить его данными";

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

c\_qu.push(TEXT[i]);

c\_vec.push\_back(TEXT[size+i]);

}

cout << "2. Просмотреть контейнер." << endl;

view\_container(c\_qu); cout << "\n" << endl;

cout << "3. Изменить контейнер, удалив из него одни элементы и заменив другие." << endl;

int pos = 0;

string new\_value = "";

cout << "Новая строка: "; cin >> new\_value;

cout << "Позиция для вставки: "; cin >> pos;

replace\_index(new\_value, pos, &c\_qu);

cout << endl;

cout << "4. Просмотреть контейнер, используя для доступа к его элементам итераторы." << endl;

view\_container(c\_qu);

cout << endl;

cout << "5. Создать второй контейнер этого же класса и заполнить его данными того же типа, что и первый контейнер." << endl;

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

c\_vec.push\_back(TEXT[size + i]);

}

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

int pos\_del = 0;

cout << "Позиция для вставки контейнера №2: "; cin >> pos\_del;

replace\_index(c\_vec, pos\_del, &c\_qu);

cout << endl;

cout << "7. Просмотреть первый и второй контейнеры." << endl;

view\_container(c\_qu);

for (auto c : c\_vec) cout << c;

return 0;

}

void print\_header() {

system("title KAZAKOV A.Y. (KT-43-21)");

string title = "Курсовая работа - КАЗАКОВ А.Ю. (KT-43-21)";

string variant = "Вариант #5. Задание №1";

center\_string(title, 42u);

center\_string(variant, 30u);

cout << endl;

}

template<typename T>

void view\_container(T data) {

auto container = data.\_Get\_container();

for (auto c : container) {

cout << c;

}

cout << endl;

}

template<typename T, typename T2>

void replace\_index(T new\_value, int pos, T2\* data) {

auto buff = data->\_Get\_container();

deque<char>::iterator it = buff.begin() + pos;

buff.erase(it);

for (int i = 0; i < new\_value.size(); i++) {

buff.insert(buff.begin() + pos + i, new\_value[i]);

}

while (data->size() != 0) data->pop();

for (auto s: buff) data->push(s);

}

void get\_cons\_buff(int& x, int& y)

{

HANDLE hWndConsole;

if (hWndConsole = GetStdHandle(-12))

{

CONSOLE\_SCREEN\_BUFFER\_INFO consoleInfo;

if (GetConsoleScreenBufferInfo(hWndConsole, &consoleInfo))

{

x = consoleInfo.srWindow.Right - consoleInfo.srWindow.Left + 1;

y = consoleInfo.srWindow.Bottom - consoleInfo.srWindow.Top + 1;

}

else

printf("Error: %d\n", GetLastError());

}

else

printf("Error: %d\n", GetLastError());

}

void center\_string(string& s, const size\_t& width)

{

int buffer\_x\_size, buffer\_y\_size;

get\_cons\_buff(buffer\_x\_size, buffer\_y\_size);

size\_t left\_bord = (buffer\_x\_size - width) / 2.;

for (;;)

{

cout << setw(left\_bord) << "";

cout << s.substr(0u, width) << endl;

s.erase(s.begin(), s.begin() + (s.size() < width ? s.size() : width));

if (s.empty())

{

break;

}

}

}

## Приложение 2

## Листинг программы для второй части

#include <iostream>

#include <string>

#include <queue>

#include <vector>

#include <stdlib.h>

#include <windows.h>

#include <iomanip>

#include <sstream>

using namespace std;

void print\_header();

void get\_cons\_buff(int&, int&);

void center\_string(string&, const size\_t&);

template<typename T>

void view\_container(T data);

class Country {

public:

Country(){}

Country(string in\_name, string in\_form\_government, float in\_area) {

this->set\_name(in\_name);

this->set\_area(in\_area);

this->set\_form\_government(in\_form\_government);

}

float get\_area() const { return area; }

string get\_name() const { return name; }

string get\_form\_government() const { return form\_government; }

void set\_name(string in\_name) { name = in\_name; }

void set\_area(float in\_area) { area = in\_area; }

void set\_form\_government(string in\_form\_government) { form\_government = in\_form\_government; }

friend ostream& operator << (ostream& os, const Country& obj) {

return os << "Название: " << obj.get\_name() << "\nФорма правления: " << obj.get\_form\_government() << "\nПлощадь: " << obj.get\_area() << endl;

}

friend istream& operator >> (istream& in, Country& obj) {

float in\_area;

string in\_name;

string in\_form\_government;

cout << "Название: "; in >> in\_name;

cout << "Форма правления: "; in >> in\_form\_government;

cout << "Площадь: "; in >> in\_area;

cout << endl;

if (in) {

obj.set\_name(in\_name);

obj.set\_area(in\_area);

obj.set\_form\_government(in\_form\_government);

}

return in;

}

private:

string name = "Not defined";

string form\_government = "Not defined";

float area = 1;

};

int main()

{

setlocale(LC\_ALL, "Russian");

print\_header();

int size = 1;

cout << "1. Создать объект-контейнер и заполнить его данными." << endl;

cout << "Количество объектов: "; cin >> size;

queue<Country> que;

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

Country obj;

cin >> obj;

que.push(obj);

}

view\_container(que);

return 0;

}

template<typename T>

void view\_container(T data) {

auto container = data.\_Get\_container();

for (auto c : container) {

cout << c;

}

cout << endl;

}

void print\_header() {

system("title KAZAKOV A.Y. (KT-43-21)");

string title = "Курсовая работа - КАЗАКОВ А.Ю. (KT-43-21)";

string variant = "Вариант #5. Задание №2";

center\_string(title, 42u);

center\_string(variant, 30u);

cout << endl;

}

void get\_cons\_buff(int& x, int& y)

{

HANDLE hWndConsole;

if (hWndConsole = GetStdHandle(-12))

{

CONSOLE\_SCREEN\_BUFFER\_INFO consoleInfo;

if (GetConsoleScreenBufferInfo(hWndConsole, &consoleInfo))

{

x = consoleInfo.srWindow.Right - consoleInfo.srWindow.Left + 1;

y = consoleInfo.srWindow.Bottom - consoleInfo.srWindow.Top + 1;

}

else

printf("Error: %d\n", GetLastError());

}

else

printf("Error: %d\n", GetLastError());

}

void center\_string(string& s, const size\_t& width)

{

int buffer\_x\_size, buffer\_y\_size;

get\_cons\_buff(buffer\_x\_size, buffer\_y\_size);

size\_t left\_bord = (buffer\_x\_size - width) / 2.;

for (;;)

{

cout << setw(left\_bord) << "";

cout << s.substr(0u, width) << endl;

s.erase(s.begin(), s.begin() + (s.size() < width ? s.size() : width));

if (s.empty())

{break;}}}

Приложение 3

Программа для третьей части

#include <iostream>

#include <string>

#include <queue>

#include <vector>

#include <deque>

using namespace std;

class Country {

public:

Country() {}

Country(string in\_name, string in\_form\_government, float in\_area) {

this->set\_name(in\_name);

this->set\_area(in\_area);

this->set\_form\_government(in\_form\_government);

}

float get\_area() const { return area; }

string get\_name() const { return name; }

string get\_form\_government() const { return form\_government; }

void set\_name(string in\_name) { name = in\_name; }

void set\_area(float in\_area) { area = in\_area; }

void set\_form\_government(string in\_form\_government) { form\_government = in\_form\_government; }

friend ostream& operator << (ostream& os, const Country& obj) {

return os << "Название: " << obj.get\_name() << "\nФорма правления: " << obj.get\_form\_government() << "\nПлощадь: " << obj.get\_area() << endl;

}

friend istream& operator >> (istream& in, Country& obj) {

float in\_area;

string in\_name;

string in\_form\_government;

cout << "Название: "; in >> in\_name;

cout << "Форма правления: "; in >> in\_form\_government;

cout << "Площадь: "; in >> in\_area;

cout << endl;

if (in) {

obj.set\_name(in\_name);

obj.set\_area(in\_area);

obj.set\_form\_government(in\_form\_government);

}

return in;

}

friend bool operator < (const Country& lhs, const Country& rhs) {

return lhs.get\_name() < rhs.get\_name();

}

private:

string name = "Not defined";

string form\_government = "Not defined";

float area = 1;

};

bool compare(const Country& lhs, const Country& rhs) {

return lhs.get\_name() > rhs.get\_name();

}

template<typename T>

void print\_container(T data);

template<typename T>

void sort\_containers(T\* data);

template<typename T>

void sort\_containers\_desc(T\* data);

template<typename T>

void find\_obj(T data);

template<typename T1, typename T2>

void move\_obj(T1\* c1, T2\* c2);

template<typename T1, typename T2, typename T3>

void merge\_containers(T1\* gen, T2 c1, T3 c2);

template<typename T>

void search\_occurrences(vector<T> gen);

template<typename T>

void search\_obj(vector<T> gen);

int main() {

system("chcp 1251");

cout << "1. Создать контейнер, содержащий объекты пользовательского типа." << endl;

int size = 1;

cout << "Количество объектов: "; cin >> size;

queue<Country> c\_queue;

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

Country obj;

cin >> obj;

c\_queue.push(obj);

}

cout << "2. Отсортировать его по убыванию элементов." << endl;

sort\_containers\_desc(&c\_queue);

cout << "3. Просмотреть контейнер." << endl;

print\_container(c\_queue);

cout << "4. Используя подходящий алгоритм, найти в контейнере элемент, удовлетворяющий заданному условию." << endl;

find\_obj(c\_queue);

cout << "5. Переместить элементы, удовлетворяющие заданному условию в другой (предварительно пустой) контейнер." << endl;

vector<Country> c\_vector;

move\_obj(&c\_queue, &c\_vector);

cout << "6. Просмотреть второй контейнер." << endl;

for (int i = 0; i < c\_vector.size(); i++) {

cout << "Страна №" << i + 1 << "\n-----------" << endl;

cout << c\_vector[i] << endl;

}

cout << endl;

cout << "7. Отсортировать первый и второй контейнеры по возрастанию элементов." << endl;

sort\_containers(&c\_queue);

sort(c\_vector.begin(), c\_vector.end());

cout << "8. Просмотреть их." << endl;

cout << "КОНТЕЙНЕР №1" << endl;

print\_container(c\_queue);

cout << "КОНТЕЙНЕР №2" << endl;

for (int i = 0; i < c\_vector.size(); i++) {

cout << "Страна №" << i + 1 << "\n-----------" << endl;

cout << c\_vector[i] << endl;

}

cout << endl;

cout << "9. Получить третий контейнер путем слияния первых двух." << endl;

vector<Country> c\_general;

merge\_containers(&c\_general, c\_queue, c\_vector);

cout << "10. Просмотреть третий контейнер." << endl;

for (int i = 0; i < c\_general.size(); i++) {

cout << "Страна №" << i + 1 << "\n-----------" << endl;

cout << c\_general[i] << endl;

}

cout << "11. Подсчитать, сколько элементов, удовлетворяющих заданному условию, содержит третий контейнер." << endl;

search\_occurrences(c\_general);

cout << "12.Определить, есть ли в третьем контейнере элемент, удовлетворяющий заданному условию." << endl;

search\_obj(c\_general);

return 0;

}

template<typename T>

void print\_container(T data) {

auto buff = data.\_Get\_container();

for (int i = 0; i < buff.size(); i++) {

cout << "Страна №" << i + 1 << "\n-----------" << endl;

cout << buff[i] << endl;

}

cout << endl;

}

template<typename T>

void sort\_containers(T\* data) {

auto buff = data->\_Get\_container();

sort(buff.begin(), buff.end());

while (data->size() != 0) data->pop();

for (auto s : buff) data->push(s);

}

template<typename T>

void sort\_containers\_desc(T\* data) {

auto buff = data->\_Get\_container();

sort(buff.begin(), buff.end(), compare);

while (data->size() != 0) data->pop();

for (auto s : buff) data->push(s);

}

template<typename T>

void find\_obj(T data) {

auto buff = data.\_Get\_container();

string search\_name = "";

cout << "Название Станы для поиска: "; cin >> search\_name;

for (auto el : buff) {

if (el.get\_name() == search\_name) {

cout << el << endl;

}}}

template<typename T1, typename T2>

void move\_obj(T1\* c1, T2\* c2) {

auto buff\_c1 = c1->\_Get\_container();

string name = "";

cout << "Название Станы для перемещения: "; cin >> name;

for (int i = 0; i < buff\_c1.size(); ++i) {

if (buff\_c1[i].get\_name() == name) {

c2->push\_back(buff\_c1[i]);

buff\_c1.erase(buff\_c1.begin() + i);

}}

while (c1->size() != 0) c1->pop();

for (auto s : buff\_c1) c1->push(s);

}

template<typename T1, typename T2, typename T3>

void merge\_containers(T1\* gen, T2 c1, T3 c2) {

auto buff\_1 = c1.\_Get\_container();

for (auto el : buff\_1) gen->push\_back(el);

for (auto el : c2) {

gen->push\_back(el);

}}

template<typename T>

void search\_occurrences(vector<T> gen) {

string search\_name = "";

cout << "Название Станы для поиска: "; cin >> search\_name;

int count = 0;

for (auto el : gen) {

if (el.get\_name() == search\_name) {

count++;

}}

cout << "Всего вхождений по условию: " << endl;

}

template<typename T>

void search\_obj(vector<T> gen) {

string search\_name = "";

cout << "Название Станы для поиска: "; cin >> search\_name;

for (auto el : gen) {

if (el.get\_name() == search\_name) {

cout << "Объект найден." << endl;

return;

}}

cout << "Вхождений не найденою" << endl;}