ФТФ, 2-ый курс, ИВТ-Б(2), Королёв Алексей.

**Ссылка на GitHub:** https://github.com/ShEIH24/Lab5\_Cpp/

**ЛАБОРАТОРНАЯ РАБОТА №5**

**Тема:** Шаблоны. Библиотека STL.

**Цель:** Получить навыки обобщенного программирования в С++, научиться использовать библиотеку STL для решения различных практических задач.

**Индивидуальные задания:**

1. Написать шаблонный класс DataManager для специфической работы с набором однотипных данных (максимальная вместимость равна 64 элементам). В набор можно добавлять данные (метод push(T elem) для добавления одного элемента и метод push(T elems[], size\_t n) для добавления группы из n элементов), считывать без извлечения (метод T peek()) и извлекать (метод T pop()) по некоторым алгоритмам (в соответствии с вариантом, приложение А). Если набор заполнен на 100% и поступает команда добавления нового элемента(ов), то данные полностью выгружаются (дописываются) в специальный файл dump.dat, а сам массив очищается и новые данные записываются уже в обновленный набор. Если из массива удаляется последний элемент, то он заполняется ранее выгруженными в файл данными (если файл не пуст).

Необходимо также реализовать явную специализацию шаблонного класса для символьного типа. В ней надо запрограммировать следующее: при добавлении символа в набор все знаки пунктуации должны автоматически заменяться на символ подчеркивания; добавить методы char popUpper() и char popLower(), которые позволяют при извлечении символа из набора привести его к верхнему или нижнему регистру, соответственно.

*Вариант 5. push(): данные пишутся в начало набора, остальные смещаются вправо; peek(): возвращает центральный элемент в наборе или 0, если число элементов четно; pop(): извлекает средний элемент из набора (если элементов четное число, то первый элемент слева от центра раздела набора).*

1. Написать код для чтения произвольного текстового файла и вывода на экран всех слов размером более 3 букв, встречающихся в нем не менее 7 раз, в порядке убывания частоты встречаемости (приложение А). В качестве разделителей слов рассматривать пробел, точку, запятую, тире, двоеточие, восклицательный знак, точку с запятой. Использовать контейнер std::map.
2. Создать класс книги Book, в котором хранится название, автор и год издания книги. В главной функции создать коллекцию книг (приложение А). Продемонстрировать сортировку книг по автору (первичный ключ) и названию (вторичный ключ). Продемонстрировать поиск в коллекции: найти все книги, год издания которых находится в указанном диапазоне. Использовать контейнер std::vector и функторы.
3. Подсчитать и вывести на консоль количество всех книг новее 2009 года, используя только стандартные алгоритмы и функторы STL (подcказка: std::count\_if, std::greater<>, std::bind)
4. Написать шаблонный класс кэша данных Cache с методом добавления элемента в кеш put(T elem) и его аналогом – оператором +=, а также методом проверки наличия элемента в кеше bool contains(T elem). Написать явную специализацию шаблона для типа std::string с такими нюансами: метод contains() должен возвращать true по совпадению первого символа строки; метод add() должен генерировать исключение, если в кеше уже есть 100 строк. В главной функции инстанцировать Cache с типами int и std::string, добавить в каждый несколько элементов и продемонстрировать для каждого работу метода contains() (см. Приложение А).
5. Модифицировать код игры «Блек-джек» из лабораторной работы № 4: использовать библиотеку STL для работы с коллекциями объектов.

**Ход работы:**

1. Код программы:

#include <iostream>

#include <fstream>

#include <iterator>

#include <algorithm>

#include <cctype>

#include <locale>

// Шаблонный класс DataManager для работы с однотипным набором данных

template <typename T, size\_t MAX\_SIZE = 64>

class DataManager {

private:

T data[MAX\_SIZE]; // Массив для хранения данных

size\_t current\_size; // Текущее количество элементов в наборе

const std::string DUMP\_FILE = "dump.dat"; // Имя файла для выгрузки данных

// Метод для смещения всех элементов вправо

void shiftRight() {

if (current\_size > 0) {

// Перемещаем каждый элемент на одну позицию вправо

for (size\_t i = current\_size; i > 0; --i) {

data[i] = data[i - 1];

}

}

}

// Метод для определения центрального индекса

size\_t getCenterIndex() const {

if (current\_size == 0) return 0;

// Для четного количества - первый элемент слева от центра

return current\_size % 2 == 0 ? (current\_size / 2) - 1 : current\_size / 2;

}

public:

// Конструктор - инициализация пустого набора

DataManager() : current\_size(0) {}

// Добавление одного элемента в набор

void push(T elem) {

// Если набор заполнен - выгружаем данные в файл

if (current\_size >= MAX\_SIZE) {

dumpToFile();

}

// Смещаем существующие элементы вправо

shiftRight();

// Вставляем новый элемент в начало

data[0] = elem;

// Увеличиваем размер, если не достигнут максимум

if (current\_size < MAX\_SIZE) {

++current\_size;

}

}

// Добавление группы элементов

void push(T elems[], size\_t n) {

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

push(elems[i]);

}

}

// Возврат центрального элемента без извлечения

T peek() const {

if (current\_size == 0) return T();

// Получаем центральный индекс

size\_t center = getCenterIndex();

return data[center];

}

// Извлечение центрального элемента

T pop() {

if (current\_size == 0) return T();

// Находим центральный индекс

size\_t center = getCenterIndex();

T elem = data[center];

// Смещаем элементы влево, перезаписывая центральный

for (size\_t i = center; i < current\_size - 1; ++i) {

data[i] = data[i + 1];

}

// Уменьшаем размер

--current\_size;

// Если набор пуст, пробуем загрузить из файла дампа

if (current\_size == 0) {

loadFromDumpFile();

}

return elem;

}

// Выгрузка данных в файл при заполнении

void dumpToFile() {

// Открываем файл для добавления в конец в бинарном режиме

std::ofstream dump(DUMP\_FILE, std::ios::app | std::ios::binary);

dump.write(reinterpret\_cast<char\*>(data), current\_size \* sizeof(T));

dump.close();

current\_size = 0;

}

// Загрузка данных из файла дампа

void loadFromDumpFile() {

std::ifstream dump(DUMP\_FILE, std::ios::binary);

if (dump) {

// Определяем размер файла

dump.seekg(0, std::ios::end);

size\_t file\_size = dump.tellg() / sizeof(T);

if (file\_size > 0) {

dump.seekg(0, std::ios::beg);

// Считываем данные, не превышая размер массива

size\_t read\_size = std::min(file\_size, MAX\_SIZE);

dump.read(reinterpret\_cast<char\*>(data), read\_size \* sizeof(T));

current\_size = read\_size;

// Очищаем файл после чтения

dump.close();

std::ofstream truncate(DUMP\_FILE, std::ios::trunc);

truncate.close();

}

}

}

// Получение текущего размера набора

size\_t size() const { return current\_size; }

};

// Явная специализация для символьного типа

template <>

class DataManager<char, 64> {

private:

char data[64];

size\_t current\_size;

const std::string DUMP\_FILE = "dump.dat";

// Метод для смещения элементов вправо

void shiftRight() {

if (current\_size > 0) {

for (size\_t i = current\_size; i > 0; --i) {

data[i] = data[i - 1];

}

}

}

// Получение центрального индекса

size\_t getCenterIndex() const {

if (current\_size == 0) return 0;

return current\_size % 2 == 0 ? (current\_size / 2) - 1 : current\_size / 2;

}

// Замена символов пунктуации на подчеркивание

char sanitizePunctuation(char c) {

return std::ispunct(static\_cast<unsigned char>(c)) ? '\_' : c;

}

public:

// Конструктор

DataManager() : current\_size(0) {}

// Добавление одного символа с заменой пунктуации

void push(char elem) {

if (current\_size >= 64) {

dumpToFile();

}

shiftRight();

// Санация символа - замена пунктуации

data[0] = sanitizePunctuation(elem);

if (current\_size < 64) {

++current\_size;

}

}

// Добавление группы символов

void push(char elems[], size\_t n) {

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

push(elems[i]);

}

}

// Возврат центрального символа

char peek() const {

if (current\_size == 0) return '\0';

size\_t center = getCenterIndex();

return data[center];

}

// Извлечение центрального символа

char pop() {

if (current\_size == 0) return '\0';

size\_t center = getCenterIndex();

char elem = data[center];

// Смещение элементов

for (size\_t i = center; i < current\_size - 1; ++i) {

data[i] = data[i + 1];

}

--current\_size;

// Загрузка из файла, если набор пуст

if (current\_size == 0) {

loadFromDumpFile();

}

return elem;

}

// Извлечение и преобразование в верхний регистр

char popUpper() {

char c = pop();

return std::toupper(static\_cast<unsigned char>(c));

}

// Извлечение и преобразование в нижний регистр

char popLower() {

char c = pop();

return std::tolower(static\_cast<unsigned char>(c));

}

// Выгрузка в файл

void dumpToFile() {

std::ofstream dump(DUMP\_FILE, std::ios::app | std::ios::binary);

dump.write(data, current\_size);

dump.close();

current\_size = 0;

}

// Загрузка из файла

void loadFromDumpFile() {

std::ifstream dump(DUMP\_FILE, std::ios::binary);

if (dump) {

dump.seekg(0, std::ios::end);

size\_t file\_size = dump.tellg();

if (file\_size > 0) {

dump.seekg(0, std::ios::beg);

size\_t read\_size = std::min(file\_size, static\_cast<size\_t>(64));

dump.read(data, read\_size);

current\_size = read\_size;

// Очистка файла после чтения

dump.close();

std::ofstream truncate(DUMP\_FILE, std::ios::trunc);

truncate.close();

}

}

}

// Получение текущего размера

size\_t size() const { return current\_size; }

};

int main() {

// Установка русской локали для корректного вывода

setlocale(LC\_ALL, "Russian");

// Демонстрация работы с целыми числами

{

std::cout << "Демонстрация DataManager для целых чисел:" << std::endl;

DataManager<int> intManager;

int intData[] = { 1, 2, 3, 4, 5, 6, 7 };

intManager.push(intData, 7);

std::cout << "Центральный элемент: " << intManager.peek() << std::endl;

std::cout << "Элементы: ";

std::copy(intData, intData + intManager.size(),

std::ostream\_iterator<int>(std::cout, " "));

std::cout << std::endl;

}

// Демонстрация работы с числами с плавающей точкой

{

std::cout << "\nДемонстрация DataManager для чисел с плавающей точкой:" << std::endl;

DataManager<double> doubleManager;

double doubleData[] = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };

doubleManager.push(doubleData, 7);

std::cout << "Центральный элемент: " << doubleManager.peek() << std::endl;

std::cout << "Элементы: ";

std::copy(doubleData, doubleData + doubleManager.size(),

std::ostream\_iterator<double>(std::cout, " "));

std::cout << std::endl;

}

// Демонстрация работы с символами

{

std::cout << "\nДемонстрация DataManager для символов:" << std::endl;

DataManager<char> charManager;

char charData[] = { 'a', 'b', 'c', 'd', 'f', '?', 'e'};

charManager.push(charData, 7);

std::cout << "Центральный элемент: " << charManager.peek() << std::endl;

std::cout << "Извлечение в верхнем регистре: " << charManager.popUpper() << std::endl;

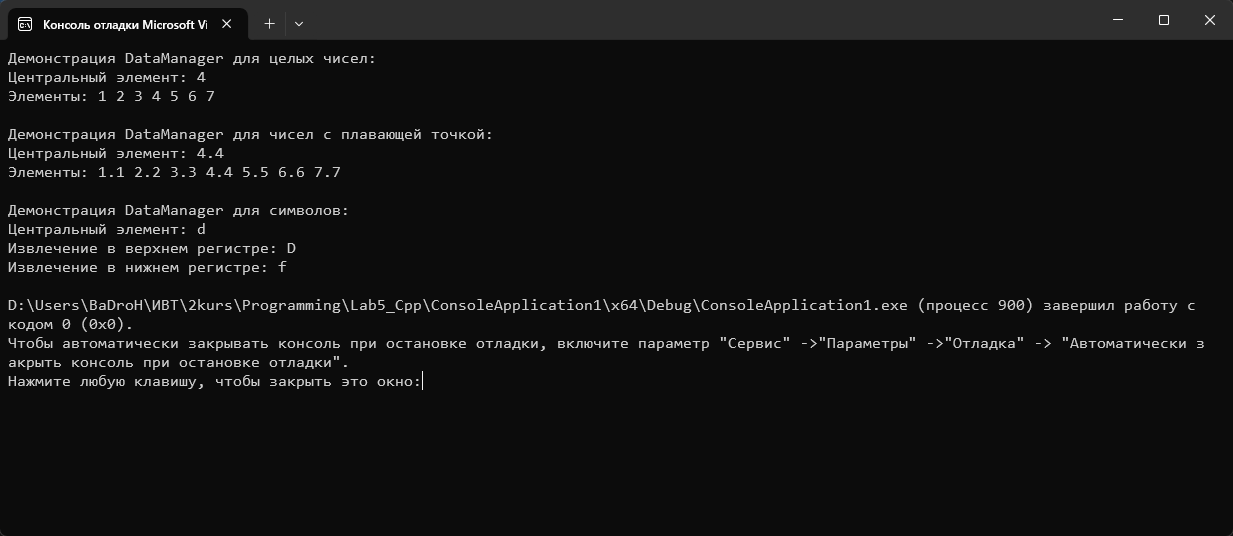
std::cout << "Извлечение в нижнем регистре: " << charManager.popLower() << std::endl;

}

return 0;

}

Результат программы:



1. Код программы:

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <map>

#include <vector>

#include <algorithm>

#include <iomanip>

class WordFrequencyCounter {

private:

// Контейнер для хранения частоты слов

// Ключ - слово, значение - количество повторений

std::map<std::string, int> wordFrequency;

// Метод для проверки, является ли символ разделителем слов

bool isSeparator(char c) {

return c == ' ' || c == '.' || c == ',' || c == '-' ||

c == ':' || c == '!' || c == ';';

}

// Метод для очистки слова от пунктуации и приведения к нижнему регистру

std::string cleanWord(const std::string& word) {

std::string cleaned;

for (char c : word) {

if (!ispunct(c)) {

cleaned += std::tolower(c);

}

}

return cleaned;

}

public:

// Метод для чтения файла и подсчета частоты слов

void processFile(const std::string& filename) {

// Открытие файла для чтения

std::ifstream file(filename);

// Проверка успешности открытия файла

if (!file.is\_open()) {

std::cerr << "Не удалось открыть файл: " << filename << std::endl;

return;

}

// Буфер для чтения строк файла

std::string line;

// Чтение файла построчно

while (std::getline(file, line)) {

// Буфер для накопления текущего слова

std::string word;

// Проход по каждому символу в строке

for (char c : line) {

// Если встретили разделитель

if (isSeparator(c)) {

// Если слово не пустое

if (!word.empty()) {

// Очистка слова от пунктуации

std::string cleanedWord = cleanWord(word);

// Учитываем только слова длиннее 3 символов

if (cleanedWord.length() > 3) {

// Увеличиваем счетчик частоты слова

wordFrequency[cleanedWord]++;

}

// Очистка буфера слова

word.clear();

}

}

else {

// Накопление символов в текущее слово

word += c;

}

}

// Обработка последнего слова в строке

if (!word.empty()) {

std::string cleanedWord = cleanWord(word);

if (cleanedWord.length() > 3) {

wordFrequency[cleanedWord]++;

}

}

}

}

// Метод для вывода слов, встречающихся не менее 7 раз

void printFrequentWords() {

// Создаем вектор пар для сортировки по частоте

std::vector<std::pair<std::string, int>> sortedWords;

// Заполнение вектора словами, встречающимися не менее 7 раз

for (const auto& pair : wordFrequency) {

if (pair.second >= 7) {

sortedWords.push\_back(pair);

}

}

// Сортировка вектора по убыванию частоты

std::sort(sortedWords.begin(), sortedWords.end(),

[](const auto& a, const auto& b) {

return a.second > b.second;

});

// Вывод отсортированных слов и их частоты в требуемом формате

for (const auto& pair : sortedWords) {

// Выравнивание по левому краю для слова и по правому для числа

std::cout << std::left << std::setw(10) << pair.first

<< std::right << std::setw(5) << pair.second

<< std::endl;

}

}

};

// Главная функция - точка входа в программу

int main() {

setlocale(LC\_ALL, "Russian");

// Создание экземпляра класса для подсчета частоты слов

WordFrequencyCounter counter;

// Обработка текстового файла

counter.processFile("input.txt");

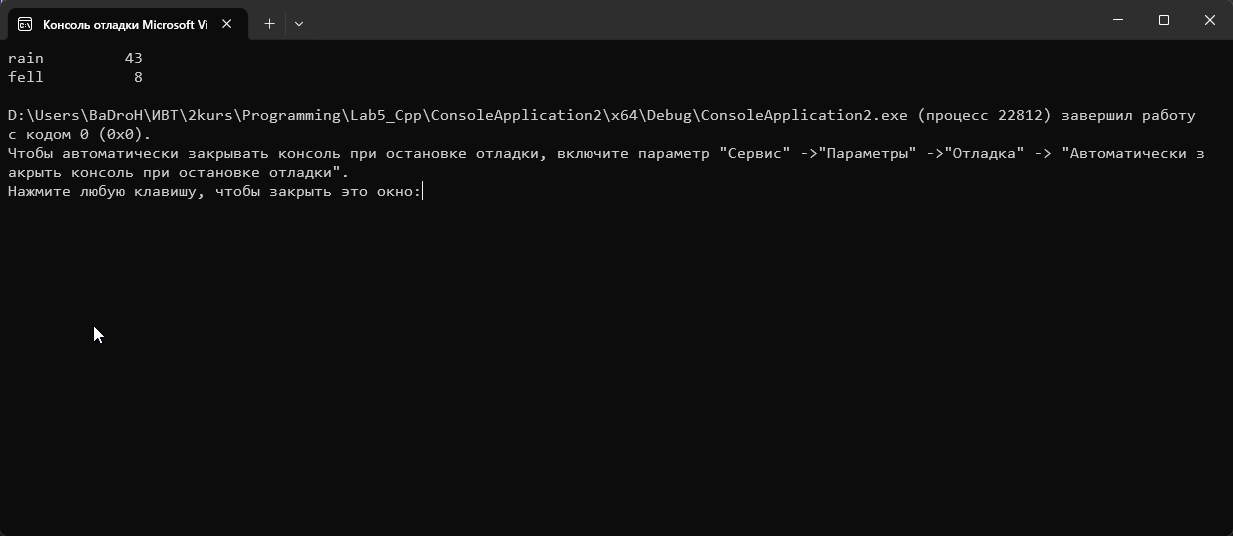
// Вывод слов, встречающихся не менее 7 раз

counter.printFrequentWords();

return 0;

}

Результат программы:



1. Код программы:

#include <iostream>

#include <vector>

#include <algorithm>

#include <string>

// Класс книги с приватными полями и публичными методами доступа

class Book {

private:

std::string name; // Название книги

std::string author; // Автор книги

int year; // Год издания

public:

// Конструктор класса Book

Book(const std::string& name, const std::string& author, int year)

: name(name), author(author), year(year) {}

// Геттер для получения названия книги

std::string getName() const { return name; }

// Геттер для получения автора книги

std::string getAuthor() const { return author; }

// Геттер для получения года издания

int getYear() const { return year; }

};

// Функтор для сортировки книг по автору (первичный ключ) и названию (вторичный ключ)

struct BookSorter {

bool operator()(const Book\* a, const Book\* b) const {

// Сначала сравниваем авторов

if (a->getAuthor() != b->getAuthor()) {

return a->getAuthor() < b->getAuthor();

}

// Если авторы одинаковы, сравниваем названия книг

return a->getName() < b->getName();

}

};

// Функтор для поиска книг в указанном диапазоне лет

struct BookFinder {

int minYear; // Минимальный год

int maxYear; // Максимальный год

// Конструктор функтора с заданием диапазона лет

BookFinder(int min, int max) : minYear(min), maxYear(max) {}

// Перегрузка оператора() для проверки попадания в диапазон

bool operator()(const Book\* book) const {

return book->getYear() >= minYear && book->getYear() <= maxYear;

}

};

int main() {

setlocale(LC\_ALL, "RUSSIAN");

// Создание вектора указателей на книги

std::vector<Book\*> books;

books.push\_back(new Book("Война и мир", "Толстой Л.Н.", 2010));

books.push\_back(new Book("Подросток", "Достоевский Ф.М.", 2004));

books.push\_back(new Book("Обрыв", "Гончаров И.А.", 2010));

books.push\_back(new Book("Анна Каренина", "Толстой Л.Н.", 1999));

books.push\_back(new Book("Обыкновенная история", "Гончаров И.А.", 2011));

books.push\_back(new Book("Утраченные иллюзии", "Бальзак О.", 2009));

books.push\_back(new Book("Оливер Твист", "Диккенс Ч.", 2001));

books.push\_back(new Book("Фауст", "Гёте И.В.", 2010));

books.push\_back(new Book("Лилия долины", "Бальзак О.", 1998));

std::cout << "\nКниги в алфавитном порядке:\n\n";

// Создание функтора для сортировки

BookSorter book\_sorter;

// Сортировка книг с помощью функтора

std::sort(books.begin(), books.end(), book\_sorter);

// Вывод отсортированных книг

for (std::vector<Book\*>::iterator i = books.begin(); i != books.end(); ++i) {

std::cout << (\*i)->getAuthor() << " \"" << (\*i)->getName() << "\"" << std::endl;

}

// Создание функтора для поиска книг в диапазоне 2005-2014 годов

BookFinder book\_finder(2005, 2014);

std::cout << "\nКниги в диапазоне года издания 2005 - 2014:\n\n";

// Поиск книг с помощью find\_if и функтора

std::vector<Book\*>::iterator finder = std::find\_if(books.begin(), books.end(), book\_finder);

// Вывод найденных книг

while (finder != books.end()) {

std::cout << (\*finder)->getAuthor() << " \"" << (\*finder)->getName() << "\"" << std::endl;

finder = std::find\_if(++finder, books.end(), book\_finder);

}

// Освобождение динамически выделенной памяти

for (std::vector<Book\*>::iterator i = books.begin(); i != books.end(); ++i) {

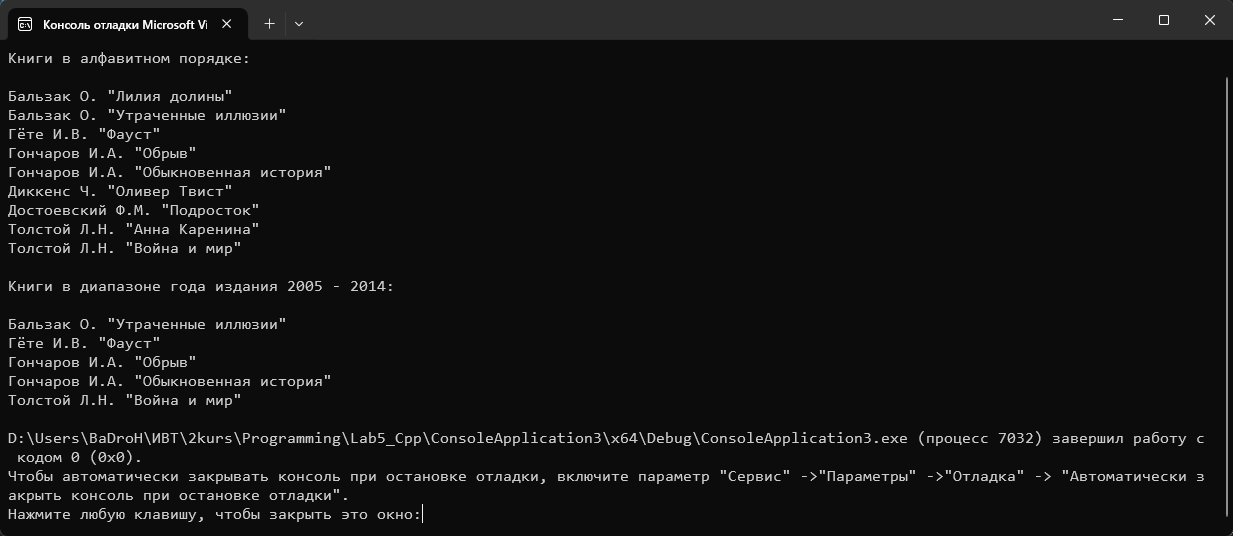
delete (\*i);

}

return 0;

}

Результат программы:



1. Код программы:

#include <iostream>

#include <vector>

#include <algorithm>

#include <string>

#include <functional> // для std::greater и std::bind2nd

class Book {

private:

std::string name; // Название книги

std::string author; // Автор книги

int year; // Год издания

public:

Book(const std::string& name, const std::string& author, int year)

: name(name), author(author), year(year) {}

std::string getName() const { return name; }

std::string getAuthor() const { return author; }

int getYear() const { return year; }

};

// Функтор для поиска книг новее указанного года

struct BookYearChecker {

bool operator()(const Book\* book) const {

return book->getYear() > 2009;

}

};

int main() {

setlocale(LC\_ALL, "RUSSIAN");

std::vector<Book\*> books;

books.push\_back(new Book("Война и мир", "Толстой Л.Н.", 2010));

books.push\_back(new Book("Подросток", "Достоевский Ф.М.", 2004));

books.push\_back(new Book("Обрыв", "Гончаров И.А.", 2010));

books.push\_back(new Book("Анна Каренина", "Толстой Л.Н.", 1999));

books.push\_back(new Book("Обыкновенная история", "Гончаров И.А.", 2011));

books.push\_back(new Book("Утраченные иллюзии", "Бальзак О.", 2009));

books.push\_back(new Book("Оливер Твист", "Диккенс Ч.", 2001));

books.push\_back(new Book("Фауст", "Гёте И.В.", 2010));

books.push\_back(new Book("Лилия долины", "Бальзак О.", 1998));

// Подсчет книг новее 2009 года с использованием std::count\_if

int newBooksCount = std::count\_if(books.begin(), books.end(), BookYearChecker());

std::cout << "Количество книг новее 2009 года: " << newBooksCount << std::endl;

// Дополнительный вариант с использованием std::greater и std::bind

int newBooksCountAlt = std::count\_if(

books.begin(),

books.end(),

std::bind(std::greater<int>(),

std::bind(&Book::getYear, std::placeholders::\_1),

2009)

);

std::cout << "Количество книг новее 2009 года (альтернативный метод): " << newBooksCountAlt << std::endl;

// Вывод списка книг новее 2009 года

std::cout << "\nСписок книг новее 2009 года:\n";

for (const auto& book : books) {

if (book->getYear() > 2009) {

std::cout << book->getAuthor() << " \"" << book->getName() << "\" ("

<< book->getYear() << " год)" << std::endl;

}

}

// Освобождение памяти

for (auto book : books) {

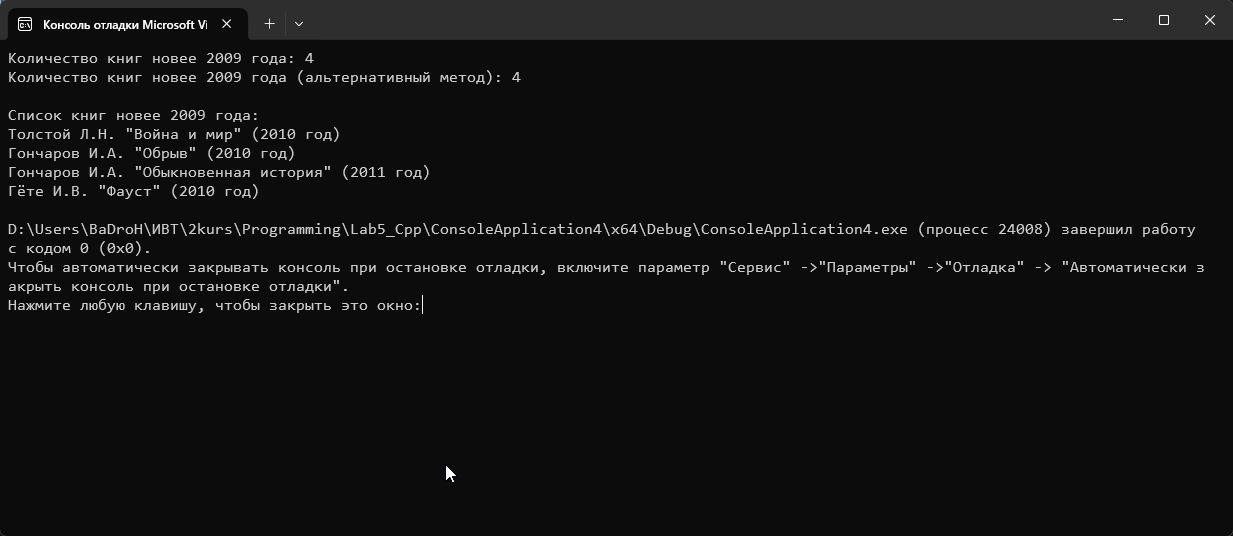
delete book;

}

return 0;

}

Результат программы:



1. Код программы:

#include <iostream>

#include <vector>

#include <string>

#include <stdexcept>

#include <algorithm>

// Основной шаблонный класс кэша

template <typename T>

class Cache {

private:

std::vector<T> data; // Контейнер для хранения элементов

public:

// Метод добавления элемента в кэш

void put(T elem) {

// Добавляем элемент, если его еще нет в кэше

if (!contains(elem)) {

data.push\_back(elem);

}

}

// Перегрузка оператора += для добавления элемента

Cache& operator+=(const T& elem) {

put(elem);

return \*this;

}

// Метод проверки наличия элемента в кэше

bool contains(T elem) const {

// Используем std::find для поиска элемента

return std::find(data.begin(), data.end(), elem) != data.end();

}

};

// Явная специализация для std::string

template <>

class Cache<std::string> {

private:

std::vector<std::string> data; // Контейнер для строк

public:

// Специализированный метод добавления строки

void put(const std::string& elem) {

// Если в кэше уже 100 строк, генерируем исключение

if (data.size() >= 100) {

throw std::runtime\_error("Максимальное количество строк в кэше достигнуто");

}

data.push\_back(elem);

}

// Перегрузка оператора +=

Cache& operator+=(const std::string& elem) {

put(elem);

return \*this;

}

// Специализированный метод проверки наличия строки

bool contains(const std::string& elem) const {

// Проверяем только первый символ строки

for (const auto& str : data) {

if (!str.empty() && !elem.empty() && str[0] == elem[0]) {

return true;

}

}

return false;

}

};

int main() {

setlocale(LC\_ALL, "Russian");

// Создаем кэш для целых чисел

Cache<int> cache;

cache.put(1); // Добавление элемента через put()

cache.put(2);

cache += 5; // Добавление элемента через оператор +=

// Создаем кэш для строк

Cache<std::string> voc;

voc.put("OK");

// Демонстрация работы contains() для разных типов

std::cout << "Проверка наличия для cache: " << std::endl;

std::cout << "Contains(5): " << cache.contains(5) << std::endl; // Должно вывести 1

std::cout << "Contains(10): " << cache.contains(10) << std::endl; // Должно вывести 0

std::cout << "\nПроверка наличия для voc: " << std::endl;

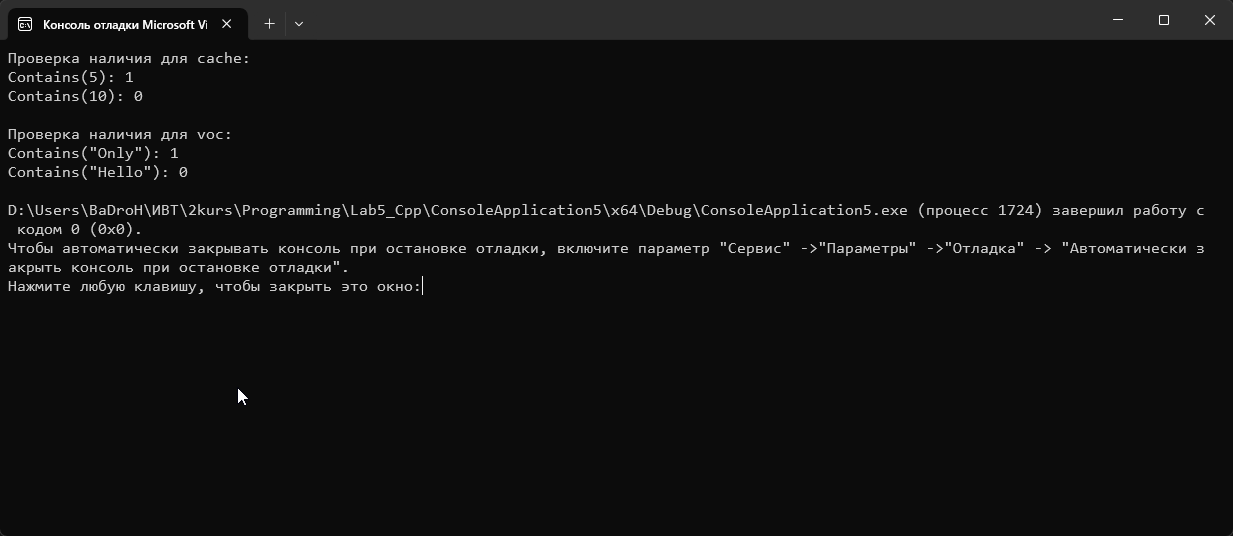
std::cout << "Contains(\"Only\"): " << voc.contains("Only") << std::endl; // Должно вывести 1

std::cout << "Contains(\"Hello\"): " << voc.contains("Hello") << std::endl; // Должно вывести 0

return 0;

}

Результат программы:



1. Код программы:

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include <random>

#include <stdexcept>

#include <iomanip>

#include <memory>

#include <map>

#include <windows.h>

#include <fcntl.h>

#include <corecrt\_io.h>

#include <unordered\_map>

#include <numeric>

// Пользовательское исключение для ситуации превышения максимальной суммы очков

// Наследуется от стандартного runtime\_error для более информативной обработки ошибок

class OverflowException : public std::runtime\_error {

public:

// Конструктор передает сообщение об ошибке в базовый класс исключений

OverflowException(const std::string& message) : std::runtime\_error(message) {}

};

// Перечисление для масти карт с Unicode-символами

// Используется для типизации и красивого отображения карт

enum class Suit {

Hearts, // ♥ Красные сердца

Diamonds, // ♦ Красные бубны

Clubs, // ♣ Черные трефы

Spades // ♠ Черные пики

};

// Перечисление для номинала карт

// Позволяет преобразовывать номинал в числовое значение

enum class Rank {

Two = 2, // Двойка

Three, // Тройка

Four, // Четверка

Five, // Пятерка

Six, // Шестерка

Seven, // Семерка

Eight, // Восьмерка

Nine, // Девятка

Ten, // Десятка

Jack, // Валет

Queen, // Дама

King, // Король

Ace // Туз

};

// Класс карты - основной элемент игры

class Card {

private:

Rank rank; // Номинал карты

Suit suit; // Масть карты

bool isFaceUp; // Состояние карты (лицом вверх/вниз)

public:

// Конструктор для создания карты с заданным номиналом и мастью

Card(Rank r, Suit s) : rank(r), suit(s), isFaceUp(true) {}

// Определение очков для каждой карты по правилам BlackJack

int getValue() const {

switch (rank) {

case Rank::Jack: // Картинки стоят 10 очков

case Rank::Queen:

case Rank::King:

return 10;

case Rank::Ace: // Туз может стоить 11 или 1

return 11;

case Rank::Ten: // Десятка стоит 10

return 10;

default:

// Для остальных карт - их номинальное значение

return static\_cast<int>(rank);

}

}

// Дружественная функция для вывода карты в консоль

friend std::ostream& operator<<(std::ostream& os, const Card& card);

// Геттеры для получения информации о карте

Rank getRank() const { return rank; }

Suit getSuit() const { return suit; }

bool getFaceUp() const { return isFaceUp; }

// Метод для изменения видимости карты

void setFaceUp(bool up) { isFaceUp = up; }

};

// Перегрузка оператора вывода для красивой печати карты

std::ostream& operator<<(std::ostream& os, const Card& card) {

if (!card.getFaceUp()) {

os << "??";

return os;

}

static const std::unordered\_map<Rank, std::string> rankNames = {

{Rank::Two, "2"}, {Rank::Three, "3"}, {Rank::Four, "4"},

{Rank::Five, "5"}, {Rank::Six, "6"}, {Rank::Seven, "7"},

{Rank::Eight, "8"}, {Rank::Nine, "9"}, {Rank::Ten, "10"},

{Rank::Jack, "J"}, {Rank::Queen, "Q"}, {Rank::King, "K"},

{Rank::Ace, "A"}

};

static const std::unordered\_map<Suit, std::string> suitSymbols = {

{Suit::Hearts, "\u2665"}, {Suit::Diamonds, "\u2666"},

{Suit::Clubs, "\u2663"}, {Suit::Spades, "\u2660"}

};

os << rankNames.at(card.getRank()) << suitSymbols.at(card.getSuit());

return os;

}

// Класс колоды для управления картами

class Deck {

protected:

std::vector<Card> cards; // Вектор для хранения карт

int numDecks; // Количество используемых колод

// Метод для создания и инициализации колоды

void createDecks() {

cards.clear(); // Очистка колоды перед заполнением

for (int d = 0; d < numDecks; ++d) {

for (const auto& suit : { Suit::Hearts, Suit::Diamonds, Suit::Clubs, Suit::Spades }) {

for (int r = static\_cast<int>(Rank::Two); r <= static\_cast<int>(Rank::Ace); ++r) {

cards.emplace\_back(static\_cast<Rank>(r), suit);

}

}

}

shuffle(); // Перемешивание после создания

}

public:

// Конструктор колоды с возможностью указать число колод

Deck(int decks = 4, bool is36 = false) : numDecks(decks) {

createDecks();

}

// Геттеры для получения информации о колоде

int getNumDecks() const { return numDecks; }

const std::vector<Card>& getAllCards() const { return cards; }

// Метод для случайного перемешивания карт

void shuffle() {

std::random\_device rd; // Генератор случайных чисел

std::mt19937 g(rd()); // Вихрь Мерсенна для качественной случайности

std::shuffle(cards.begin(), cards.end(), g); // Перемешивание

}

// Метод для взятия карты из колоды

Card drawCard() {

if (cards.empty()) {

createDecks(); // Если карты кончились, пересоздаем колоду

}

Card drawn = cards.back(); // Берем последнюю карту

cards.pop\_back(); // Удаляем ее из колоды

return drawn;

}

// Метод для подсчета оставшихся карт

int remainingCards() const { return cards.size(); }

};

// Абстрактный базовый класс для игроков

class Player {

protected:

std::vector<Card> hand; // Рука игрока (набор карт)

int score; // Текущее количество очков

bool isBust; // Признак превышения 21 очка

int balance; // Баланс игрока

int currentBet; // Текущая ставка

public:

// Конструктор с начальным балансом

Player(int initialBalance = 10000) :

score(0), isBust(false), balance(initialBalance), currentBet(0) {}

// Метод для установки ставки

void placeBet(int bet) {

if (bet > balance) {

throw std::runtime\_error("Недостаточно средств");

}

currentBet = bet;

balance -= bet;

}

// Метод для начисления выигрыша (удвоение ставки)

void win() {

balance += currentBet \* 2;

}

// Метод возврата ставки в случае ничьей

void push() {

balance += currentBet;

}

// Геттеры для баланса и ставки

int getBalance() const { return balance; }

int getCurrentBet() const { return currentBet; }

// Добавление карты в руку с пересчетом очков

void addCard(Card card) {

hand.push\_back(std::move(card)); // Использование std::move для перемещения

updateScore();

}

virtual void updateScore() {

score = std::accumulate(hand.begin(), hand.end(), 0, [](int total, const Card& card) {

return total + card.getValue();

});

int aceCount = std::count\_if(hand.begin(), hand.end(), [](const Card& card) {

return card.getRank() == Rank::Ace;

});

while (score > 21 && aceCount > 0) {

score -= 10; // Туз как 1 вместо 11

aceCount--;

}

if (score > 21) {

isBust = true;

throw OverflowException("Перебор!");

}

}

// Геттеры для получения информации о руке

int getScore() const { return score; }

bool getBust() const { return isBust; }

const std::vector<Card>& getHand() const { return hand; }

// Чисто виртуальный метод для логики взятия карт

virtual bool shouldTakeCard() = 0;

virtual ~Player() = default;

};

// Класс дилера с особой стратегией взятия карт

class Dealer : public Player {

public:

// Дилер берет карты, пока сумма меньше 17

bool shouldTakeCard() override {

return score < 17;

}

// Метод для сокрытия первой карты дилера

void showInitialHand() {

if (!hand.empty()) {

hand[0].setFaceUp(false); // Первая карта закрыта

}

}

};

// Класс игрока-человека с интерактивным выбором действий

class HumanPlayer : public Player {

public:

HumanPlayer(int initialBalance = 10000) : Player(initialBalance) {}

// Интерактивный метод выбора действия

bool shouldTakeCard() override {

std::cout << "\n1. Хватит\n";

std::cout << "2. Еще\n";

// Добавляем опцию сплита, если позволяет структура руки

if (canSplit()) {

std::cout << "3. Сплит?\n";

}

int choice;

std::cin >> choice;

switch (choice) {

case 1: return false; // Не брать карту

case 2: return true; // Взять карту

case 3:

if (canSplit()) {

// Логика сплита (пока не реализована)

std::cout << "Сплит пока не реализован.\n";

return false;

}

return false;

default: return false;

}

}

// Проверка возможности сплита

bool canSplit() const {

// Сплит возможен, если в руке 2 карты одного номинала

return (hand.size() == 2 &&

hand[0].getRank() == hand[1].getRank());

}

};

// Основной класс игры BlackJack

class BlackJackGame {

private:

Deck deck; // Колода карт

std::unique\_ptr<Dealer> dealer; // Указатель на дилера

std::unique\_ptr<HumanPlayer> player; // Указатель на игрока

// Метод вывода руки дилера с возможностью скрытия первой карты

void printDealerHand(bool hideFirst = true) {

std::cout << "Дилер: ";

const auto& dealerHand = dealer->getHand();

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

if (i == 0 && hideFirst) {

std::cout << "?? "; // Скрытая карта

}

else {

std::cout << dealerHand[i] << " ";

}

}

std::cout << std::endl;

}

// Метод вывода руки игрока

void printPlayerHand() {

std::cout << "Вы: ";

for (const auto& card : player->getHand()) {

std::cout << card << " ";

}

std::cout << std::endl;

}

public:

// Конструктор игры с инициализацией колоды и игроков

BlackJackGame() : deck(4) {

dealer = std::make\_unique<Dealer>();

player = std::make\_unique<HumanPlayer>();

}

// Основной метод игры

void play() {

try {

// Запрос и установка ставки

int bet;

std::cout << "Ваш баланс: " << player->getBalance() << std::endl;

std::cout << "Ваша ставка? ";

std::cin >> bet;

player->placeBet(bet);

// Начальная раздача карт

initialDeal();

printDealerHand();

printPlayerHand();

// Ход игрока

playerTurn();

// Ход дилера, если игрок не перебрал

if (!player->getBust()) {

dealerTurn();

determineWinner();

}

}

catch (const std::exception& e) {

std::cerr << "Ошибка: " << e.what() << std::endl;

}

}

void initialDeal() {

player->addCard(deck.drawCard());

dealer->addCard(deck.drawCard());

player->addCard(deck.drawCard());

dealer->addCard(deck.drawCard());

}

void playerTurn() {

try {

while (player->shouldTakeCard()) {

Card newCard = deck.drawCard();

player->addCard(newCard);

printDealerHand();

printPlayerHand();

}

}

catch (const OverflowException& e) {

std::cout << "Перебор! Ваши карты: ";

for (const auto& card : player->getHand()) {

std::cout << card << " ";

}

std::cout << "\nОбщая сумма: " << player->getScore() << std::endl;

}

}

void dealerTurn() {

try {

while (dealer->shouldTakeCard()) {

dealer->addCard(deck.drawCard());

}

}

catch (const OverflowException& e) {

std::cout << "Дилер перебрал!\n";

}

}

void determineWinner() {

int playerScore = player->getScore();

int dealerScore = dealer->getScore();

std::cout << "\n--- Результаты ---\n";

std::cout << "Ваши очки: " << playerScore << std::endl;

std::cout << "Очки дилера: " << dealerScore << std::endl;

if (player->getBust()) {

std::cout << "Вы проиграли (перебор).\n";

}

else if (dealer->getBust()) {

std::cout << "Вы победили (дилер перебрал)!\n";

player->win();

}

else if (playerScore > dealerScore) {

std::cout << "Вы победили!\n";

player->win();

}

else if (playerScore < dealerScore) {

std::cout << "Вы проиграли.\n";

}

else {

std::cout << "Ничья.\n";

player->push();

}

std::cout << "Ваш выигрыш: " << player->getCurrentBet()

<< ". Всего: " << player->getBalance() << std::endl;

}

};

// Главная функция программы

int main() {

// Установка локали для корректного отображения русских символов

setlocale(LC\_ALL, "");

SetConsoleOutputCP(CP\_UTF8);

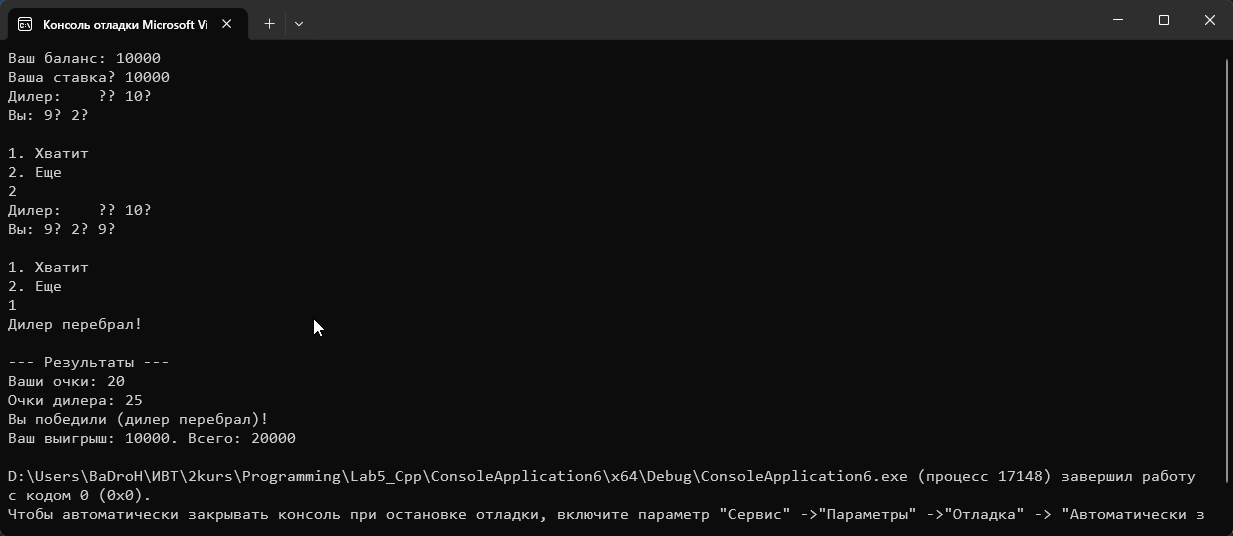
// Создание и запуск игры

BlackJackGame game;

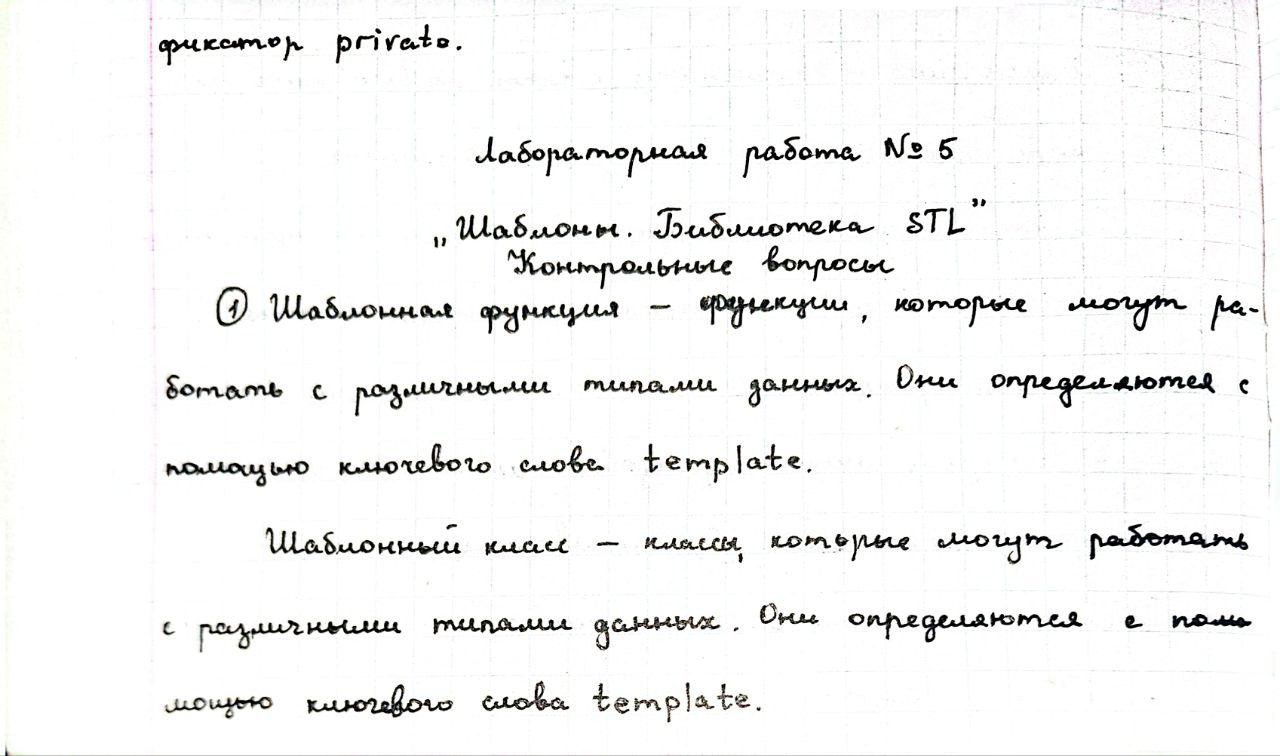
game.play();

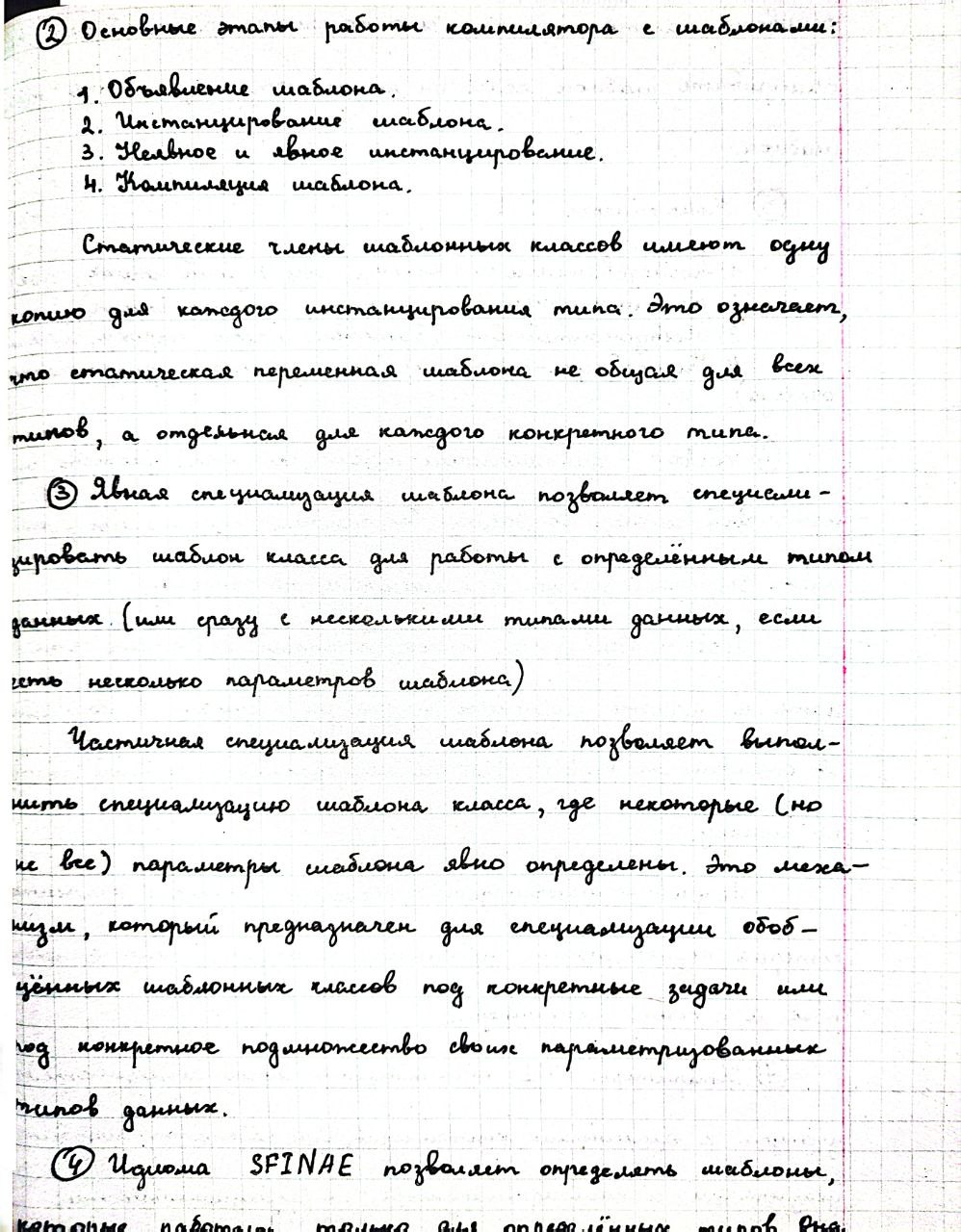
}

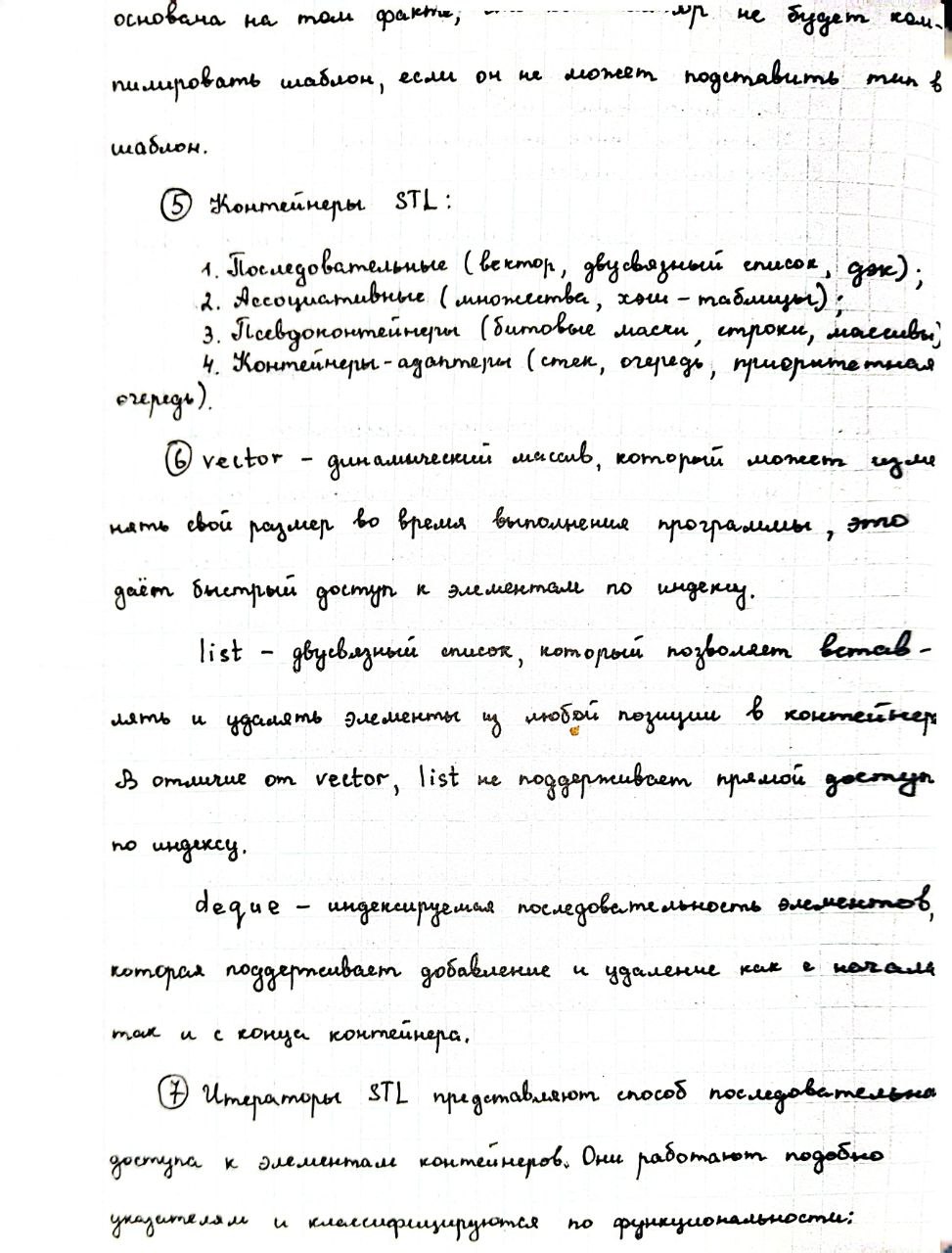
Результат программы:

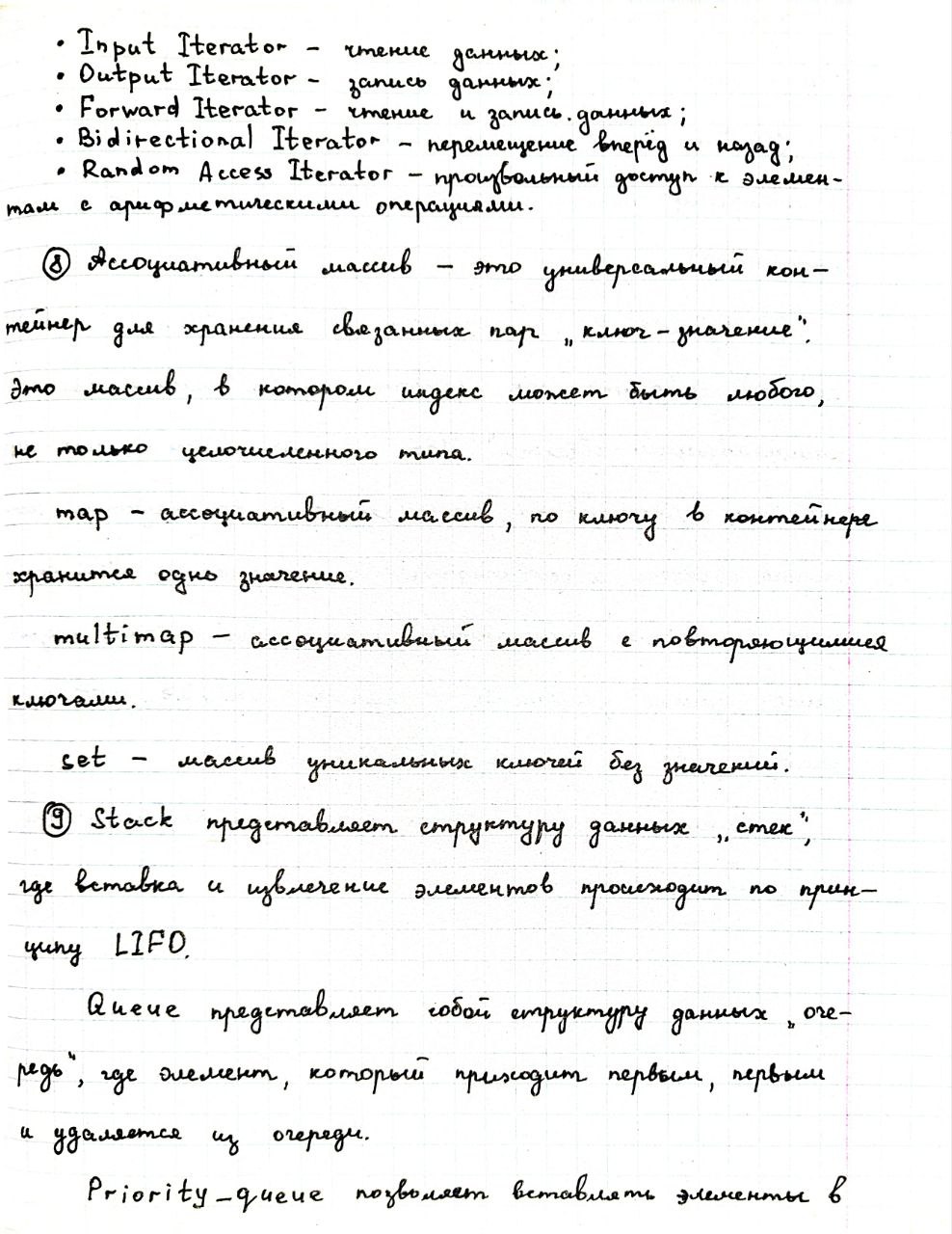


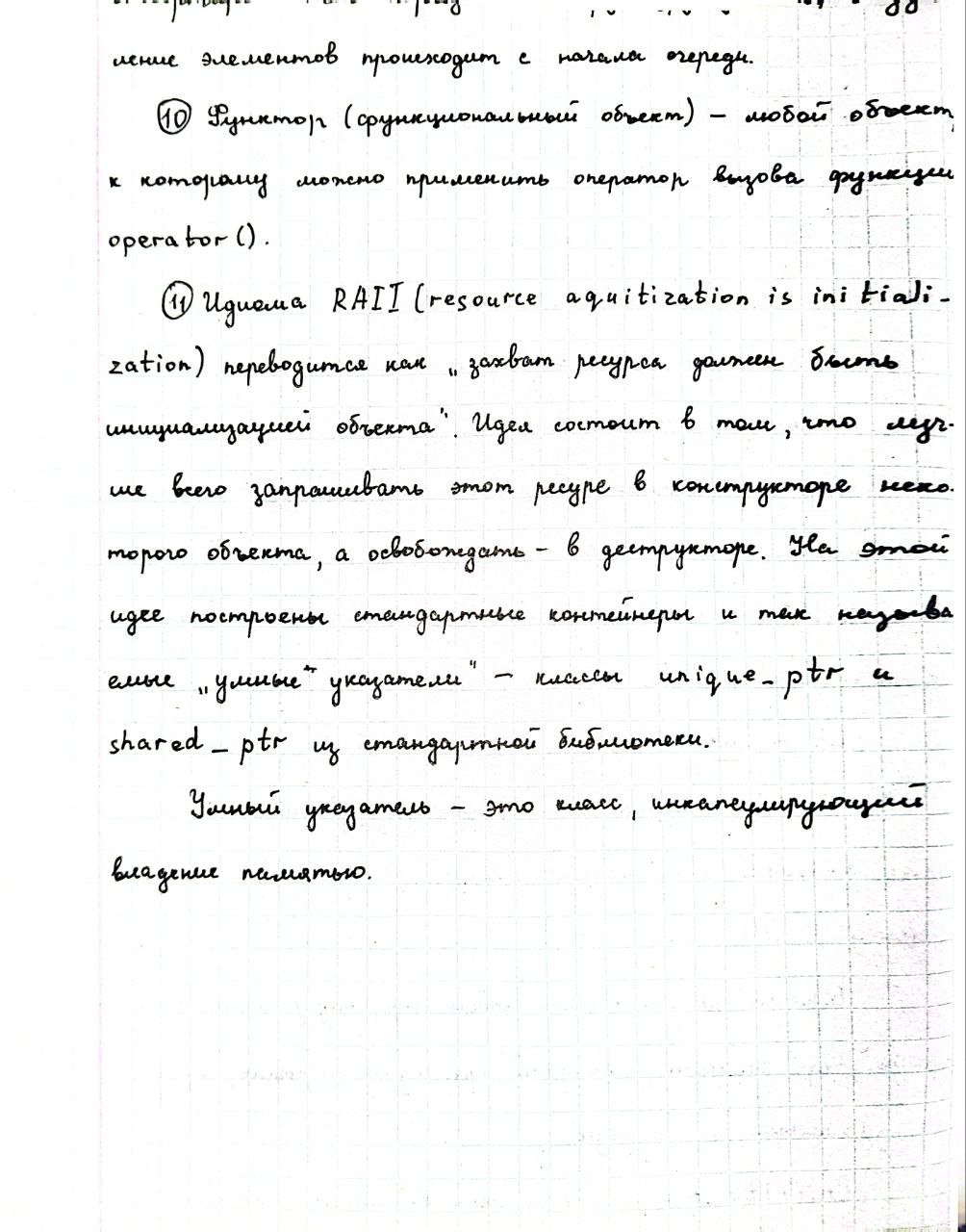
Контрольные вопросы:











Вывод: Получил навыки обобщенного программирования в С++, научился использовать библиотеку STL для решения различных практических задач.