ЛАБОРАТОРНАЯ РАБОТА №5

Курс «Объектно-ориентированное программирование»

Тема: Шаблоны. Библиотека STL.

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

Ход работы

Вариант 5

Задание 1.

Написать шаблонный класс DataManager<T> 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.

В функции main() продемонстрировать применение шаблонного класса DataManager для типов int, double и char. Элементы контейнера должны выводиться на консоль с помощью std::ostream_iterator.

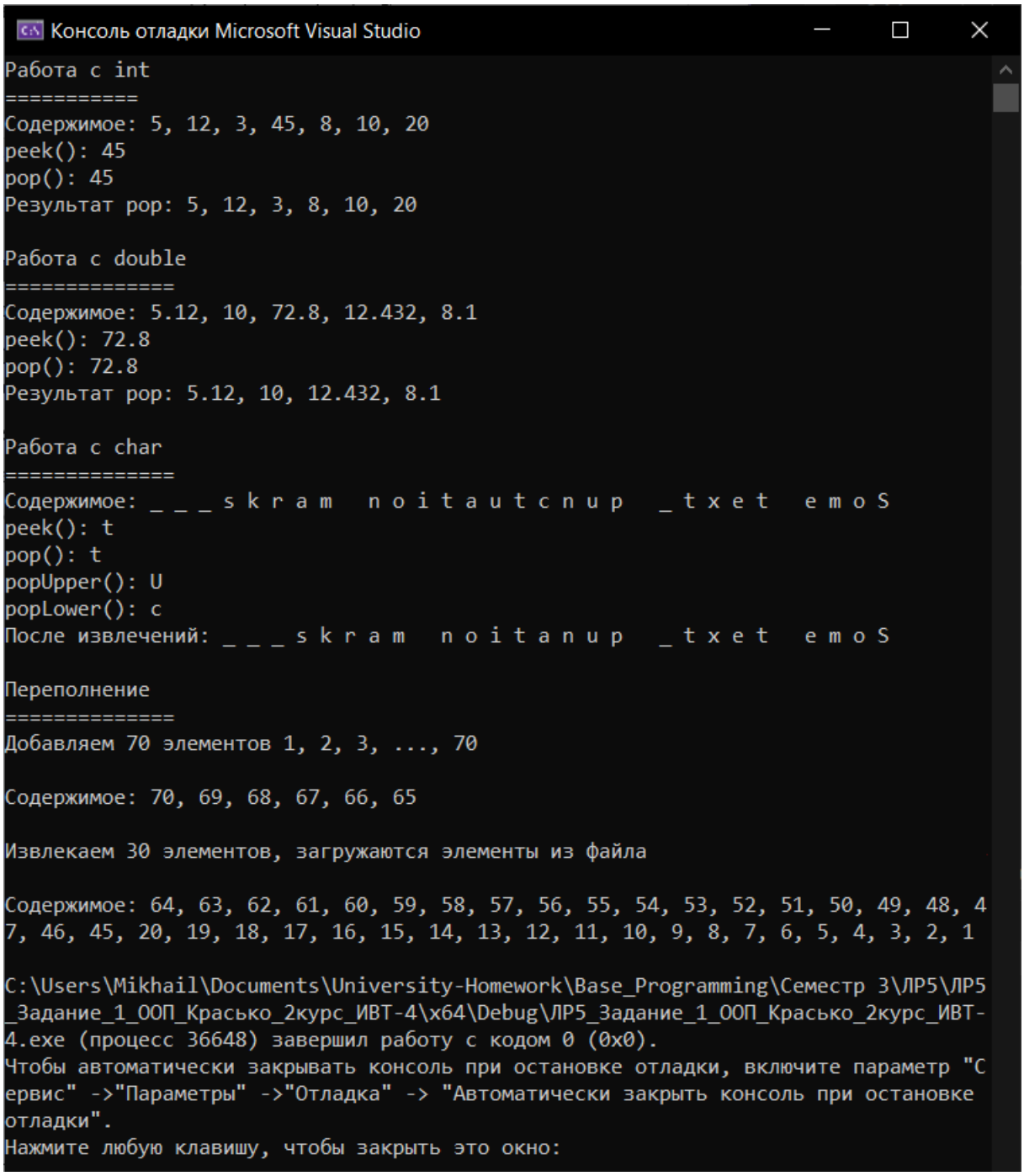
Вариант 5.

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).

Результат работы программы:



Код:

```
#include <iostream>
#include <fstream>
#include <algorithm>
#include <iterator>
#include <cctype>
#include <vector>

template <typename T>
class DataManager {
private:
    T data[64];
    int currentSize;
    const char* filename = "dump.dat";
    std::fstream file;

public:
    DataManager() : currentSize(0) {
        // Открываем файл для работы
        file.open(filename, std::ios::in | std::ios::out |
std::ios::binary); // открытие файла для чтения и записи в бинарном
режиме
        if (!file.is_open()) {
            // Создание файла только для записи, закрытие и
открытие уже с чтением
            file.open(filename, std::ios::out | std::ios::binary);
            file.close();
            file.open(filename, std::ios::in | std::ios::out |
std::ios::binary);
        }
    }

    ~DataManager() {
        if (file.is_open()) {
            file.close();
        }
    }

    void push(T elem) {
        if (currentSize == 64) {
            dumpToFile();
            currentSize = 0;
        }

        // Добавляем элемент в начало
        for (int i = currentSize; i > 0; i--) {
            data[i] = data[i - 1];
        }
        data[0] = elem;
        currentSize++;
    }

    void push(T elems[], size_t n) {
        size_t elementsToAdd;
        if (n > 64) { // Защита от переполнения
            elementsToAdd = 64;
        }
```

```
            else {
                elementsToAdd = n;
            }
            for (size_t i = 0; i < elementsToAdd; i++) {
                push(elems[i]);
            }
        }

        T peek() const {
            if (currentSize == 0) return T(0);

            // Для четного количества тоже возвращаем 0
            if (currentSize % 2 == 0) {
                return T(0);
            }
            else {
                return data[currentSize / 2];
            }
        }

        T pop() {
            if (currentSize == 0) {
                loadFromFile();
                // Если файл пустой
                if (currentSize == 0) return T(0);
            }

            int index;
            if (currentSize % 2 == 0) {
                index = currentSize / 2 - 1; // Первый элемент слева от
центра
            }
            else {
                index = currentSize / 2; // Центральный элемент
            }

            T result = data[index];

            // Сдвиг элементов влево от удаляемого включительно и до
конца
            for (int i = index; i < currentSize - 1; i++) {
                data[i] = data[i + 1];
            }
            currentSize--;

            return result;
        }

        void print(std::ostream& os = std::cout) const {
            // итератор вывода типа T
            std::ostream_iterator<T> out_it(os, ", "); // out_it пишет
в поток os (т.е. std::cout)
            for (int i = 0; i < currentSize; i++) {
                // Убираем запятую после последнего элемента
                if (i == currentSize - 1) {
                    os << data[i];
```

```
                }
                else {
                    *out_it = data[i];
                    ++out_it;
                }
            }
            os << std::endl;
        }

        int size() const {
            return currentSize;
        }

    private:
        void dumpToFile() {
            if (!file.is_open()) return;

            file.seekp(0, std::ios::end); // seekp тут перемещает
указатель в конец файла
            // Исправлено: записываем только текущие элементы, не весь
массив
            for (int i = 0; i < currentSize; i++) {
                file.write(reinterpret_cast<const char*>(&data[i]),
sizeof(T)); // Преобразует указатель на T в указатель на char
            }
            file.flush(); // запись данных из буфера на диск
        }

        void loadFromFile() {
            if (!file.is_open()) return;

            file.seekg(0, std::ios::beg); // Начало файла
            file.clear(); // Сброс флагов ошибок

            int elementsRead = 0;
            for (elementsRead = 0; elementsRead < 64; elementsRead++) {
                if
(!file.read(reinterpret_cast<char*>(&data[elementsRead]), sizeof(T))) {
                    break;
                }
            }
            currentSize = elementsRead;


            if (elementsRead > 0) {
                file.close();

                // Открываем файл и очищаем его
                file.open(filename, std::ios::out | std::ios::trunc);
                file.close();
                // Снова открываем для чтения и записи
                file.open(filename, std::ios::in | std::ios::out |
std::ios::binary);
            }
        }
    };
```

```
    // Специализация для char
    template <>
    class DataManager<char> {
    private:
        char data[64];
        int currentSize;
        const char* filename = "dump.dat";
        std::fstream file;

    public:
        // Открываем файл для работы
        DataManager() : currentSize(0) {
            file.open(filename, std::ios::in | std::ios::out |
std::ios::binary); // открытие файла для чтения и записи в бинарном
режиме
            if (!file.is_open()) {
                // Создание файла только для записи, закрытие и
открытие уже с чтением
                file.open(filename, std::ios::out | std::ios::binary);
                file.close();
                file.open(filename, std::ios::in | std::ios::out |
std::ios::binary);
            }
        }

        ~DataManager() {
            if (file.is_open()) {
                file.close();
            }
        }

        void push(char elem) {
            // Заменяем пунктуацию на _
            if (std::ispunct(elem)) {
                elem = '_';
            }

            if (currentSize == 64) {
                dumpToFile();
                currentSize = 0;
            }

            for (int i = currentSize; i > 0; i--) {
                data[i] = data[i - 1];
            }
            data[0] = elem;
            currentSize++;
        }

        void push(char elems[], size_t n) {
            size_t elementsToAdd;
            if (n > 64) { // Защита от переполнения
                elementsToAdd = 64;
            }
            else {
                elementsToAdd = n;
```

```
        }
        for (size_t i = 0; i < elementsToAdd; i++) {
            push(elems[i]);
        }
    }

    char peek() const {
        if (currentSize == 0) return 0;

        // Для четного количества тоже возвращаем 0
        if (currentSize % 2 == 0) {
            return 0;
        }
        else {
            return data[currentSize / 2];
        }
    }

    char pop() {
        if (currentSize == 0) {
            loadFromFile();
            // Если файл пустой
            if (currentSize == 0) return 0;
        }

        int index;
        if (currentSize % 2 == 0) {
            index = currentSize / 2 - 1; // Первый элемент слева от
центра
        }
        else {
            index = currentSize / 2; // Центральный элемент
        }

        char result = data[index];

        // Сдвиг элементов влево от удаляемого включительно и до
конца
        for (int i = index; i < currentSize - 1; i++) {
            data[i] = data[i + 1];
        }
        currentSize--;

        return result;
    }

    char popUpper() {
        char c = pop(); // Получаем символ из массива
        return static_cast<char>(
            std::toupper(static_cast<unsigned char>(c))); //
Преобразование char в unsigned char и обратно
    }

    char popLower() {
        char c = pop();
        return static_cast<char>(
```

```
                std::tolower(static_cast<unsigned char>(c)));
        }

        // Метод для вывода с использованием ostream_iterator
        void print(std::ostream& os = std::cout) const {
            std::ostream_iterator<char> out_it(os, " ");
            for (int i = 0; i < currentSize; i++) {
                *out_it = data[i];
                ++out_it;
            }
            os << std::endl;
        }

        int size() const {
            return currentSize;
        }

    private:
        void dumpToFile() {
            if (!file.is_open()) return;

            file.seekp(0, std::ios::end); // seekp тут перемещает
указатель в конец файла
            for (int i = 0; i < currentSize; i++) {
                file.write(&data[i], sizeof(char));
            }
            file.flush(); // запись данных из буфера на диск
        }

        void loadFromFile() {
            if (!file.is_open()) return;

            file.seekg(0, std::ios::beg); // Начало файла
            file.clear(); // Сброс флагов ошибок

            int elementsRead = 0;
            for (elementsRead = 0; elementsRead < 64; elementsRead++) {
                if (!file.read(&data[elementsRead], sizeof(char))) {
                    break;
                }
            }
            currentSize = elementsRead;


            if (elementsRead > 0) {
                file.close();

                // Открываем файл и очищаем его
                file.open(filename, std::ios::out | std::ios::trunc);
                file.close();
                // Снова открываем для чтения и записи
                file.open(filename, std::ios::in | std::ios::out |
std::ios::binary);
            }
        }
    };
```

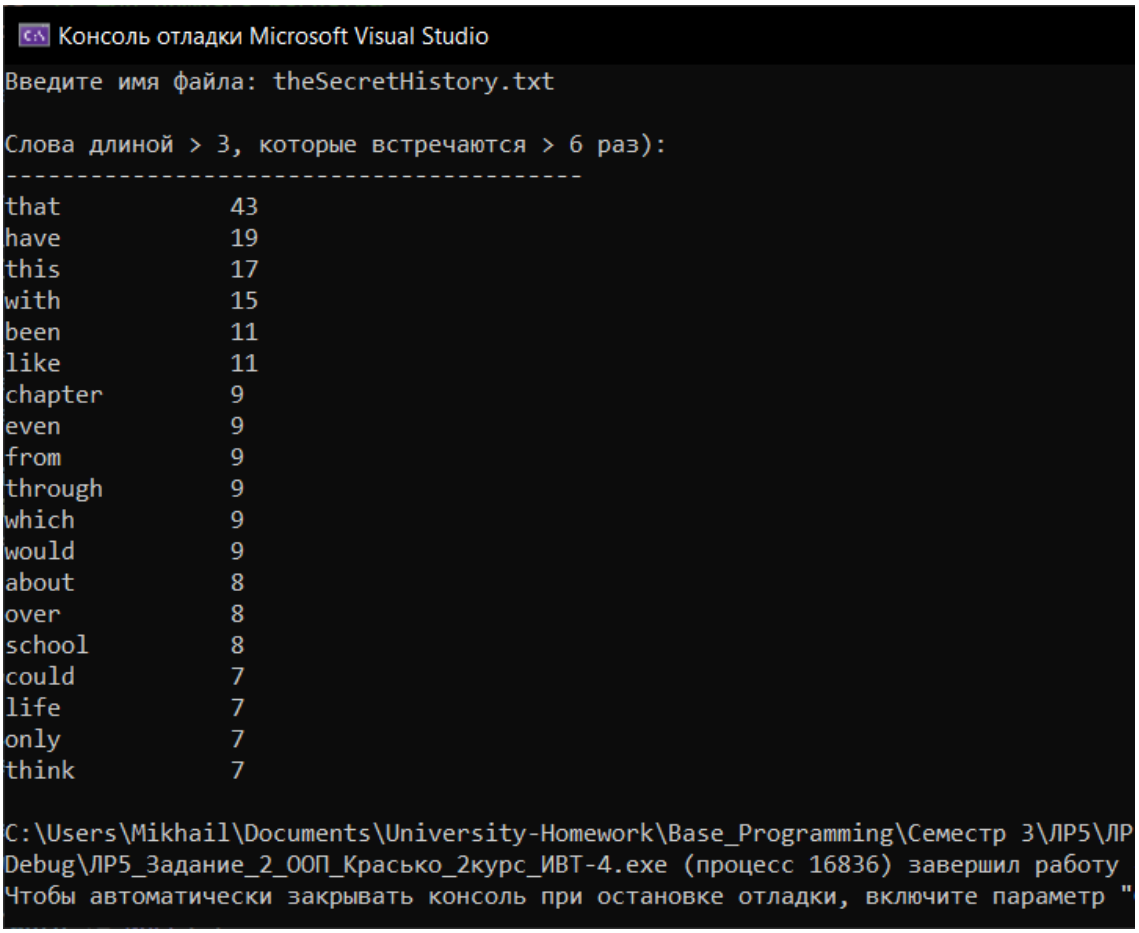
```cpp
int main() {
    setlocale(LC_ALL, "Russian");

    std::cout << "Работа с int\n==========\n";
    DataManager<int> intManager;

    int intArr[] = { 20, 10, 8, 45, 3, 12, 5 };
    intManager.push(intArr, 7);

    std::cout << "Содержимое: ";
    intManager.print();

    std::cout << "peek(): " << intManager.peek() << "\n";
    std::cout << "pop(): " << intManager.pop() << "\n";
    std::cout << "Результат pop: ";
    intManager.print();

    std::cout << "\nРабота с double\n==========\n";
    DataManager<double> doubleManager;

    double doubleArr[] = { 8.1, 12.432, 72.8, 10, 5.12 };
    doubleManager.push(doubleArr, 5);

    std::cout << "Содержимое: ";
    doubleManager.print();

    std::cout << "peek(): " << doubleManager.peek() << "\n";
    std::cout << "pop(): " << doubleManager.pop() << "\n";
    std::cout << "Результат pop: ";
    doubleManager.print();

    std::cout << "\nРабота с char\n==========\n";
    DataManager<char> charManager;

    char charArr[] = "Some text, punctuation marks:,.";

    charManager.push(charArr, strlen(charArr));

    std::cout << "Содержимое: ";
    charManager.print();

    std::cout << "peek(): " << charManager.peek() << "\n";
    std::cout << "pop(): " << charManager.pop() << "\n";
    std::cout << "popUpper(): " << charManager.popUpper() << "\n";
    std::cout << "popLower(): " << charManager.popLower() << "\n";
    std::cout << "После извлечений: ";
    charManager.print();

    std::cout << "\nПереполнение\n==========\n";
    DataManager<int> testManager;

    for (int i = 1; i <= 70; i++) {
        testManager.push(i);
    }
```

```
        std::cout << "Добавляем 70 элементов 1, 2, 3, ..., 70\n";
        std::cout << "\nСодержимое: ";
        testManager.print();

        std::cout << "\nИзвлекаем 30 элементов, загружаются элементы из
файла\n";
        for (int i = 1; i <= 30; i++) {
            testManager.pop();
        }

        std::cout << "\nСодержимое: ";
        testManager.print();

        return 0;
    }
```

Задание 2.

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.

Результат работы программы:



Код:

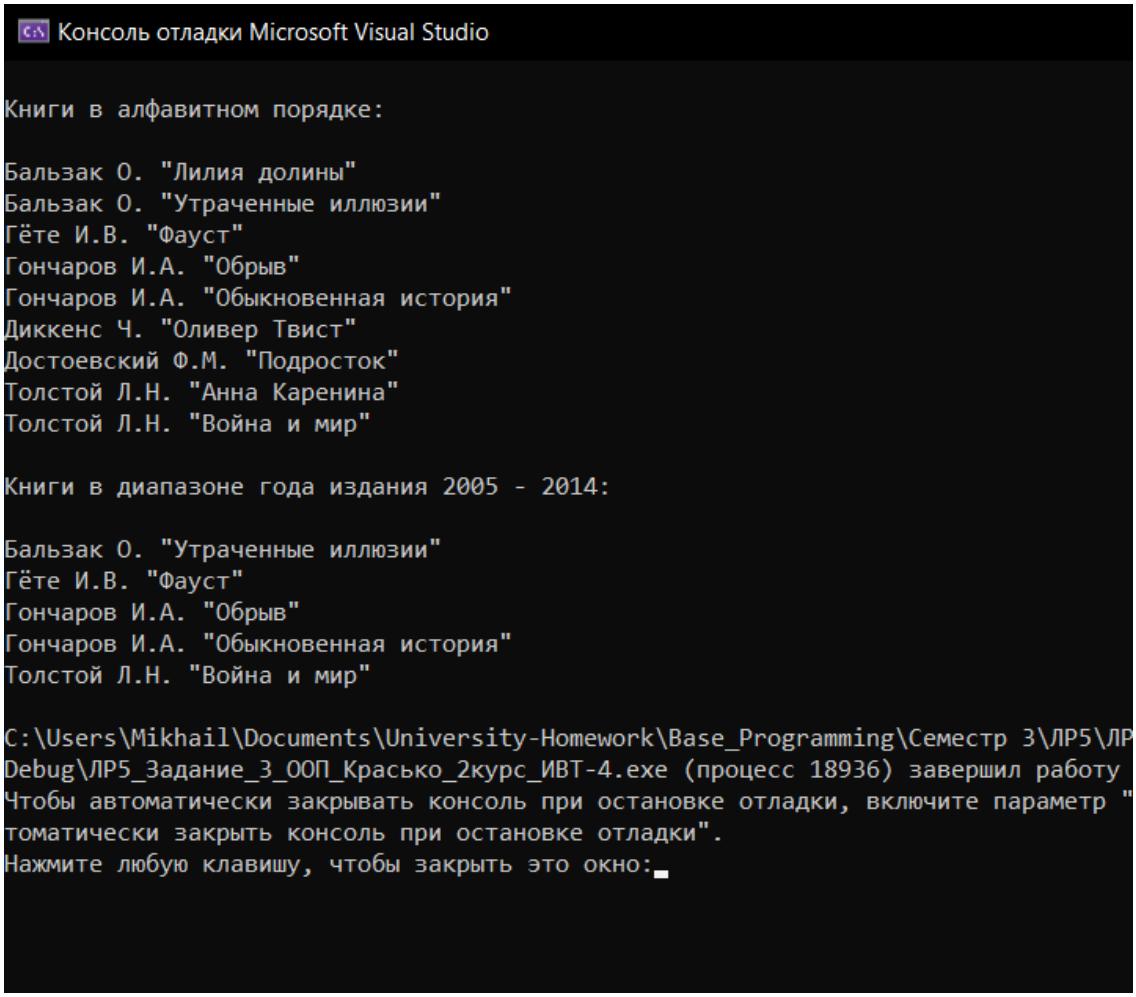
```
#define _CRT_SECURE_NO_WARNINGS

#include <iostream>
#include <fstream>
#include <map>
#include <vector>
#include <cstring>
#include <cctype> // для нижнего регистра
#include <windows.h>  // Для смены кодировки консоли

using namespace std;

int main() {
    setlocale(LC_ALL, "Russian");

    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    cout << "Введите имя файла: ";
    string filename;
    cin >> filename;

    ifstream file(filename);
    if (!file.is_open()) {
        cout << "Ошибка! Файл не найден." << endl;
        return 1;
    }

    map<string, int> wordsMap;

    const size_t stringLen = 250;
    char line[stringLen];

    while (file.getline(line, stringLen)) {
        char* word = strtok(line, " .,-:!;");

        while (word != NULL) {
            string currentWord = word;

            // Всё слова должны быть в нижнем регистре
            for (size_t i = 0; i < currentWord.length(); i++) {
                currentWord[i] = tolower((unsigned
char)currentWord[i]);
            }

            // Проверяем длину слова
            if (currentWord.length() > 3) {
                wordsMap[currentWord]++;
            }
            // следующее слово для следующей иттерации
            word = strtok(NULL, " .,-:!;");
        }
    }
```

```
        file.close();

        // Переносим в вектор пар сторок с количеством вхождений
        vector<pair<string, int>> wordsVector;

        for (auto it = wordsMap.begin(); it != wordsMap.end(); ++it) {
            if (it->second >= 7) {
                wordsVector.push_back(make_pair(it->first, it-
>second));
            }
        }

        // пузырьковая сортировка
        if (!wordsVector.empty()) {
            for (size_t i = 0; i < wordsVector.size(); i++) {
                for (size_t j = 0; j < wordsVector.size() - 1; j++) {
                    if (wordsVector[j].second < wordsVector[j +
1].second) {
                        auto temp = wordsVector[j];
                        wordsVector[j] = wordsVector[j + 1];
                        wordsVector[j + 1] = temp;
                    }
                }
            }
        }

        // Выводим результаты
        cout << "\nСлова длиной > 3, которые встречаются > 6 раз):\n---
-----------------------------------\n";

        if (wordsVector.empty()) {
            cout << "Нет слов, удовлетворяющих условию" << endl;
        }
        else {
            for (size_t i = 0; i < wordsVector.size(); i++) {
                cout << wordsVector[i].first << "\t\t" <<
wordsVector[i].second << endl;
            }
        }

        return 0;
    }
```

Задание 3.

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.

Результат работы программы:



Код:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>

using namespace std;


class Book {
private:
    string name;
    string author;
    int year;

public:
    Book(string n, string a, int y) {
        name = n;
        author = a;
        year = y;
    }

    // Геттеры
```

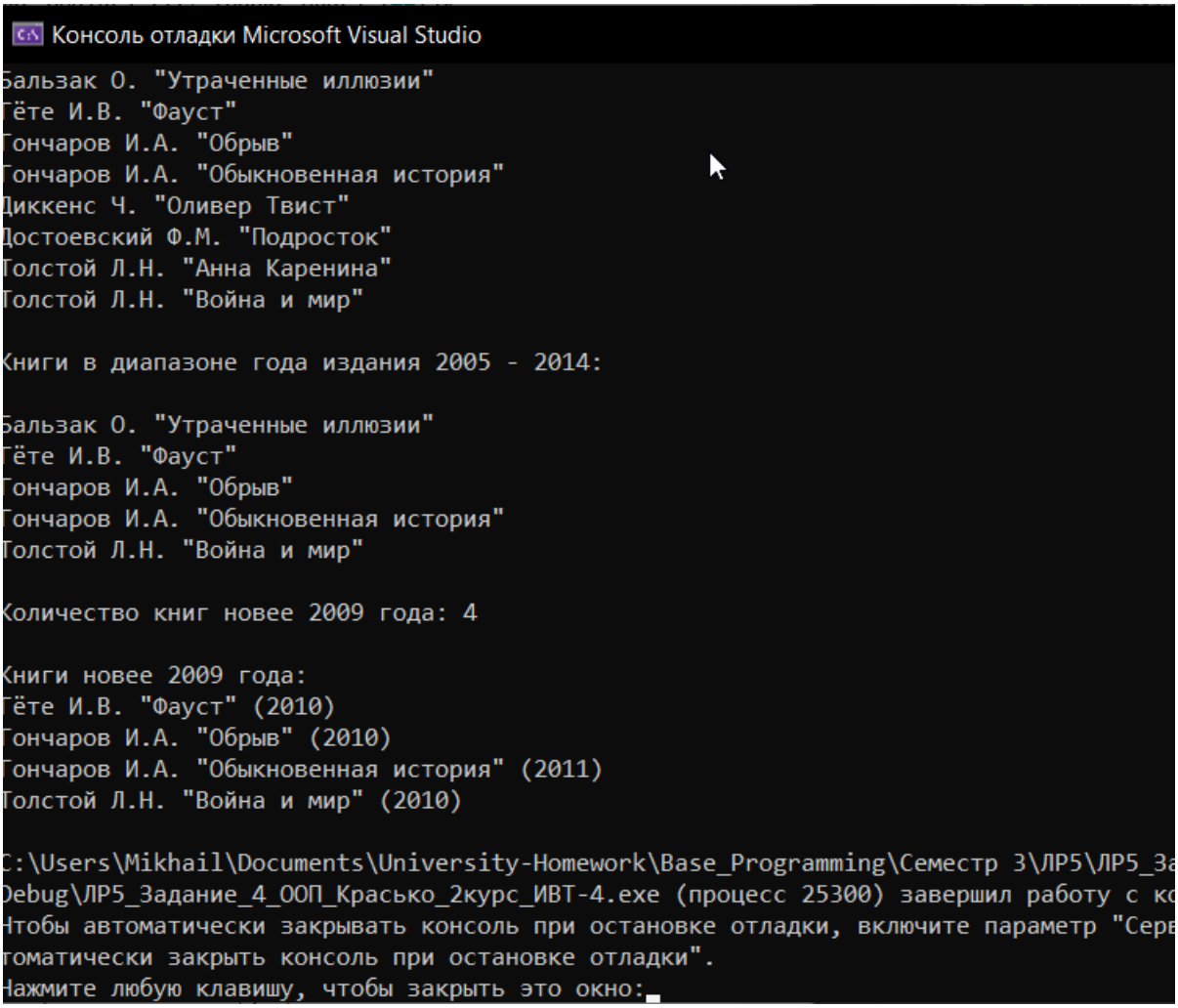
```
        string getName() const {
            return name;
        }

        string getAuthor() const {
            return author;
        }

        int getYear() const {
            return year;
        }
    };

    // Функтор для сортировки
    class BookSorter {
    public:
        bool operator()(Book* b1, Book* b2) {
            // В первую очередь сравниваются авторы
            if (b1->getAuthor() != b2->getAuthor()) {
                return b1->getAuthor() < b2->getAuthor();
            }
            // Если авторы одинаковые, сравниваются названия
            return b1->getName() < b2->getName();
        }
    };

    // Функтор для фильтрации книг по годам
    class BookFinder {
    private:
        int startYear;
        int endYear;

    public:
        BookFinder(int start, int end) {
            startYear = start;
            endYear = end;
        }

        bool operator()(Book* book) {
            return book->getYear() >= startYear && book->getYear() <=
endYear;
        }
    };

    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));

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

        BookSorter bookSorterObject;
        std::sort(books.begin(), books.end(), bookSorterObject); //
bookSorterObject – признак сортировки

        std::vector<Book*>::iterator i; // умный указатель, ходящий по
списку книг
        for (i = books.begin(); i != books.end(); ++i)
        {
            std::cout << (*i)->getAuthor() << " \""
                << (*i)->getName() << "\"" << std::endl;
        }

        BookFinder bookFinderObject(2005, 2014);
        std::vector<Book*>::iterator finder =
std::find_if(books.begin(), books.end(), bookFinderObject);
        // find_if ищет превый элемент, для которого функтор возвращает
true

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

        while (finder != books.end())
        {
            std::cout << (*finder)->getAuthor() << " \""
                << (*finder)->getName() << "\"" << std::endl;
            finder = std::find_if(++finder, books.end(),
bookFinderObject);
        }

        for (i = books.begin(); i != books.end(); ++i)
        {
            delete (*i);
        }

        return 0;
    }
```

Задание 4.

Подсчитать и вывести на консоль количество всех книг новее 2009 года, используя только стандартные алгоритмы и функторы STL (подсказка: std::count_if, std::greater<>, std::bind2nd).

Результат работы программы:



Код:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <functional>  // Необходимо для greater и bind2nd

using namespace std;


class Book {
private:
    string name;
    string author;
    int year;

public:
    Book(string n, string a, int y) {
        name = n;
        author = a;
        year = y;
    }

    // Геттеры
```

```
        string getName() const {
            return name;
        }

        string getAuthor() const {
            return author;
        }

        int getYear() const {
            return year;
        }
    };

    // Функтор для сортировки
    class BookSorter {
    public:
        bool operator()(Book* b1, Book* b2) {
            // В первую очередь сравниваются авторы
            if (b1->getAuthor() != b2->getAuthor()) {
                return b1->getAuthor() < b2->getAuthor();
            }
            // Если авторы одинаковые, сравниваются названия
            return b1->getName() < b2->getName();
        }
    };

    // Функтор для фильтрации книг по годам
    class BookFinder {
    private:
        int startYear;
        int endYear;

    public:
        BookFinder(int start, int end) {
            startYear = start;
            endYear = end;
        }

        bool operator()(Book* book) {
            return book->getYear() >= startYear && book->getYear() <=
endYear;
        }
    };

    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));
```

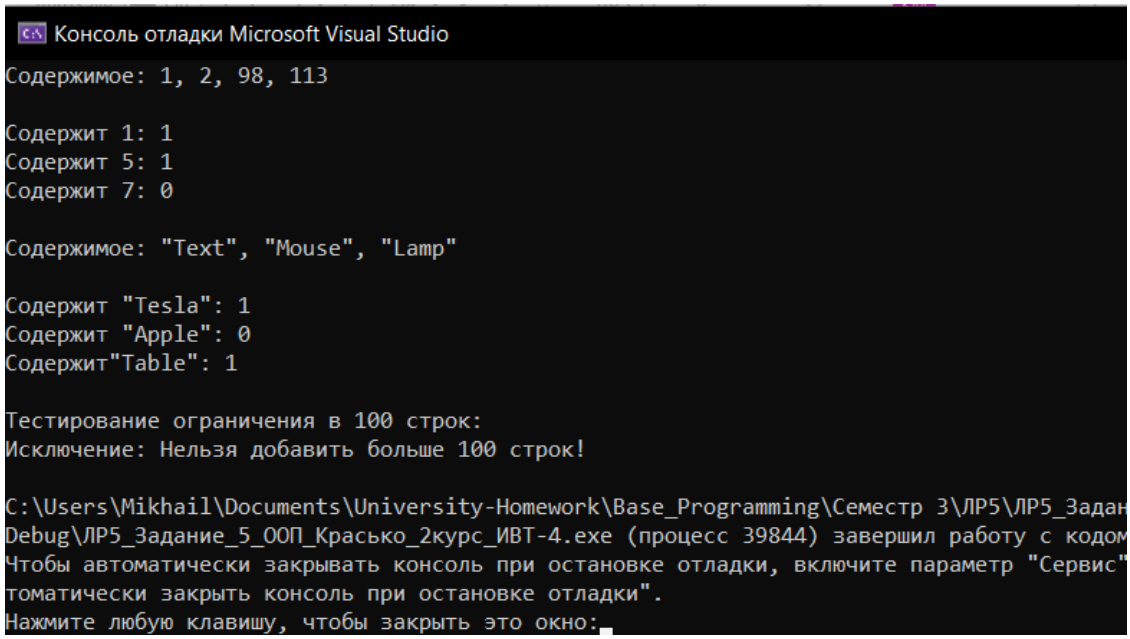
```
        books.push_back(new Book("Утраченные иллюзии", "Бальзак О.",
2009));
        books.push_back(new Book("Оливер Твист", "Диккенс Ч.", 2001));
        books.push_back(new Book("Фауст", "Гёте И.В.", 2010));
        books.push_back(new Book("Лилия долины", "Бальзак О.", 1998));

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

        BookSorter bookSorterObject;
        std::sort(books.begin(), books.end(), bookSorterObject); //
bookSorterObject - признак сортировки

        std::vector<Book*>::iterator i; // умный указатель, ходящий по
списку книг
        for (i = books.begin(); i != books.end(); ++i)
        {
            std::cout << (*i)->getAuthor() << " \""
                << (*i)->getName() << "\"" << std::endl;
        }

        BookFinder bookFinderObject(2005, 2014);
        std::vector<Book*>::iterator finder =
std::find_if(books.begin(), books.end(), bookFinderObject);
        // find_if ищет первый элемент, для которого функтор возвращает
true

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

        while (finder != books.end())
        {
            std::cout << (*finder)->getAuthor() << " \""
                << (*finder)->getName() << "\"\n";;
            finder = std::find_if(++finder, books.end(),
bookFinderObject);
        }


        std::cout << "\nКоличество книг новее 2009 года: ";

        // Получаем годы в отдельный вектор
        std::vector<int> years;
        for (i = books.begin(); i != books.end(); ++i) {
            years.push_back((*i)->getYear());
        }

        int count = std::count_if(years.begin(), years.end(),
            std::bind2nd(std::greater<int>(), 2009));

        // std::greater<int> - функтор для целых чисел, возвращает,
больше ли 1-е число 2-го
        // std::bind2nd() - функция-адаптер, создающий новый функтор

        std::cout << count << std::endl;

        std::cout << "\nКниги новее 2009 года:\n";
```

```
        for (i = books.begin(); i != books.end(); ++i) {
            if ((*i)->getYear() > 2009) {  // Проверяем, новее ли 2009
года
                std::cout << (*i)->getAuthor() << " \""
                    << (*i)->getName() << "\" ("
                    << (*i)->getYear() << ")\n";
            }
        }

        for (i = books.begin(); i != books.end(); ++i)
        {
            delete (*i);
        }

        return 0;
    }
```

Задание 5.

Написать шаблонный класс кэша данных Cache<T> 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).

Результат работы программы:



Код:

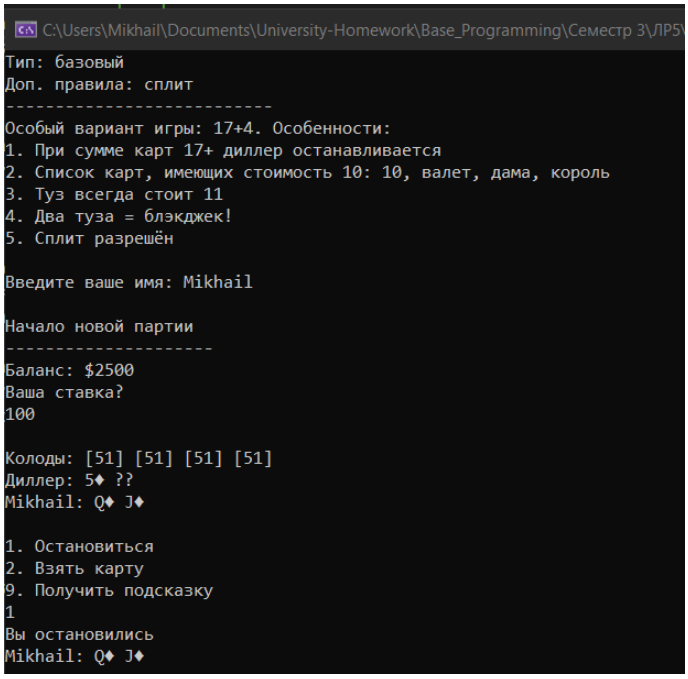
```
#include <iostream>
#include <vector>
#include <string>
#include <stdexcept> // для исключений

template <typename T> // T - тип-заглушка
class Cache {
private:
    std::vector<T> elements;

public:
    void put(T elem) {
        elements.push_back(elem);
    }

    Cache<T>& operator+=(T elem) {
        // Чтобы не дублировать код
        put(elem);
        return *this; // Возвращает сам себя
    }

    // Метод проверки наличия элемента
    bool contains(T elem) {
        for (size_t i = 0; i < elements.size(); i++) {
            if (elements[i] == elem) {
                return true;
            }
        }
        return false;
    }
};

// Явная специализация
template <> // <> знак специализации
class Cache<std::string> {
private:
    std::vector<std::string> elements;

public:
    void put(std::string elem) {
        if (elements.size() >= 100) {
            throw std::runtime_error("Нельзя добавить больше 100
строк!");
        }
        elements.push_back(elem);
    }

    Cache<std::string>& operator+=(std::string elem) {
        put(elem);
        return *this;
    }

    // Метод проверки наличия сравнивает только первые символы
    bool contains(std::string elem) {
```

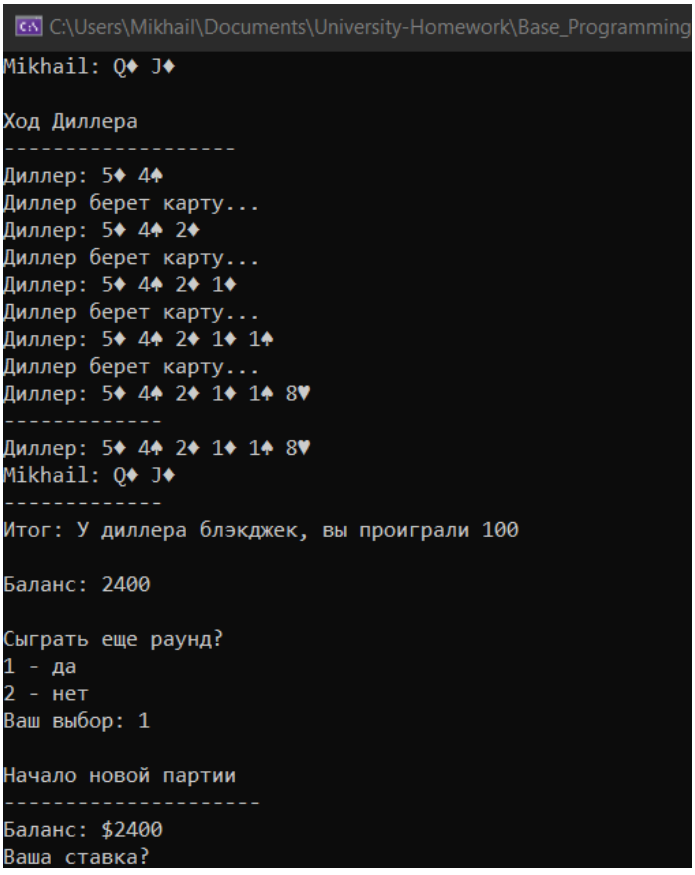
```
            char firstChar = elem[0];

            for (size_t i = 0; i < elements.size(); i++) {
                if (!elements[i].empty() && elements[i][0] ==
firstChar) {
                    return true;
                }
            }
            return false;
        }
    };

    int main() {
        setlocale(LC_ALL, "Russian");

        Cache<int> cache;

        cache.put(1);
        cache.put(5);
        cache += 98;
        cache += 113;
        std::cout << "Содержимое: 1, 2, 98, 113\n\n";

        std::cout << "Содержит 1: " << cache.contains(1) << "\n";
        std::cout << "Содержит 5: " << cache.contains(5) << "\n";
        std::cout << "Содержит 7: " << cache.contains(7) << "\n";

        Cache<std::string> voc;

        voc.put("Text");
        voc.put("Mouse");
        voc += "Lamp";
        std::cout << "\nСодержимое: \"Text\", \"Mouse\", \"Lamp\"\n\n";

        std::cout << "Содержит \"Tesla\": " << voc.contains("Tesla") <<
"\n";
        std::cout << "Содержит \"Apple\": " << voc.contains("Apple") <<
"\n";
        std::cout << "Содержит\"Table\": " << voc.contains("Table") <<
"\n";

        std::cout << "\nТестирование ограничения в 100 строк:" << "\n";
        Cache<std::string> Cache2;

        try {
            for (int i = 0; i < 101; i++) {
                Cache2 += "text " + std::to_string(i);
            }
            std::cout << "101-я строка добавлена" << "\n";
        }
        catch (const std::runtime_error& e) {
            std::cout << "Исключение: " << e.what() << "\n";
        }
        return 0;
    }
```

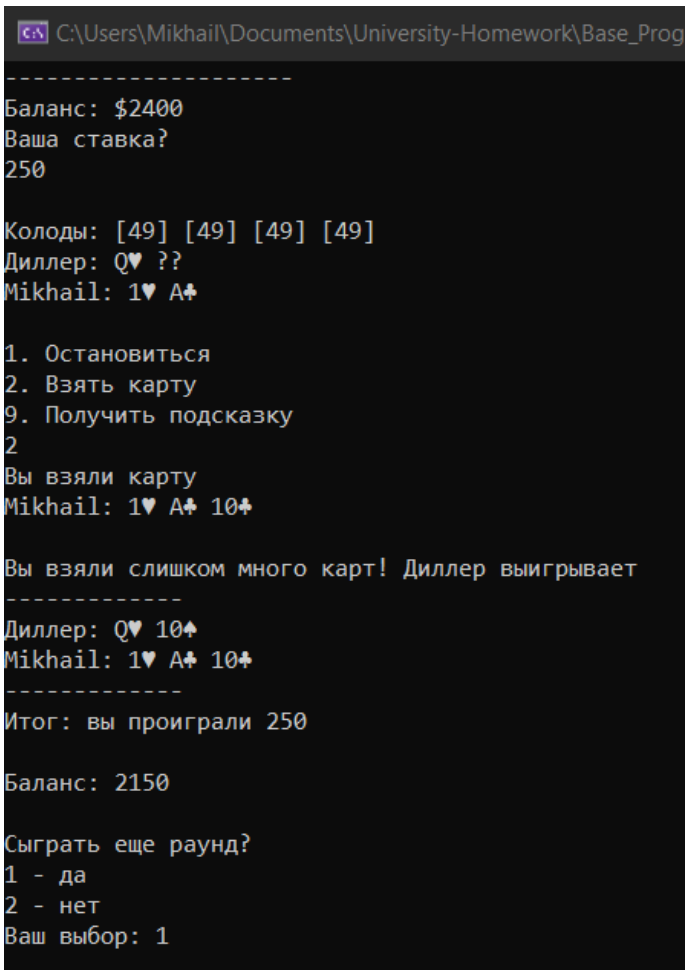
Задание 6.

Модифицировать код игры «Блек-джек» из лабораторной работы № 4: использовать библиотеку STL для работы с коллекциями объектов.

Результат работы программы:



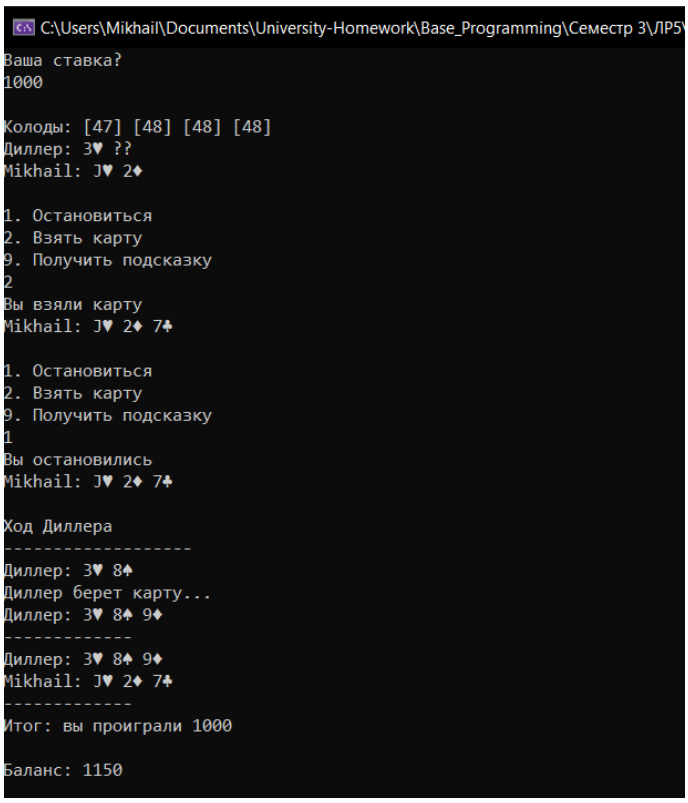



C:\Users\Mikhail\Documents\University-Homework\Base_Prog

Баланс: $2400
Ваша ставка?
250
Колоды: [49] [49] [49] [49]
Диллер: Q♥ ??
Mikhail: 1♥ A♣
1. Остановиться
2. Взять карту
9. Получить подсказку
2
Вы взяли карту
Mikhail: 1♥ A♣ 10♣
Вы взяли слишком много карт! Диллер выигрывает

Диллер: Q♥ 10♠
Mikhail: 1♥ A♣ 10♣

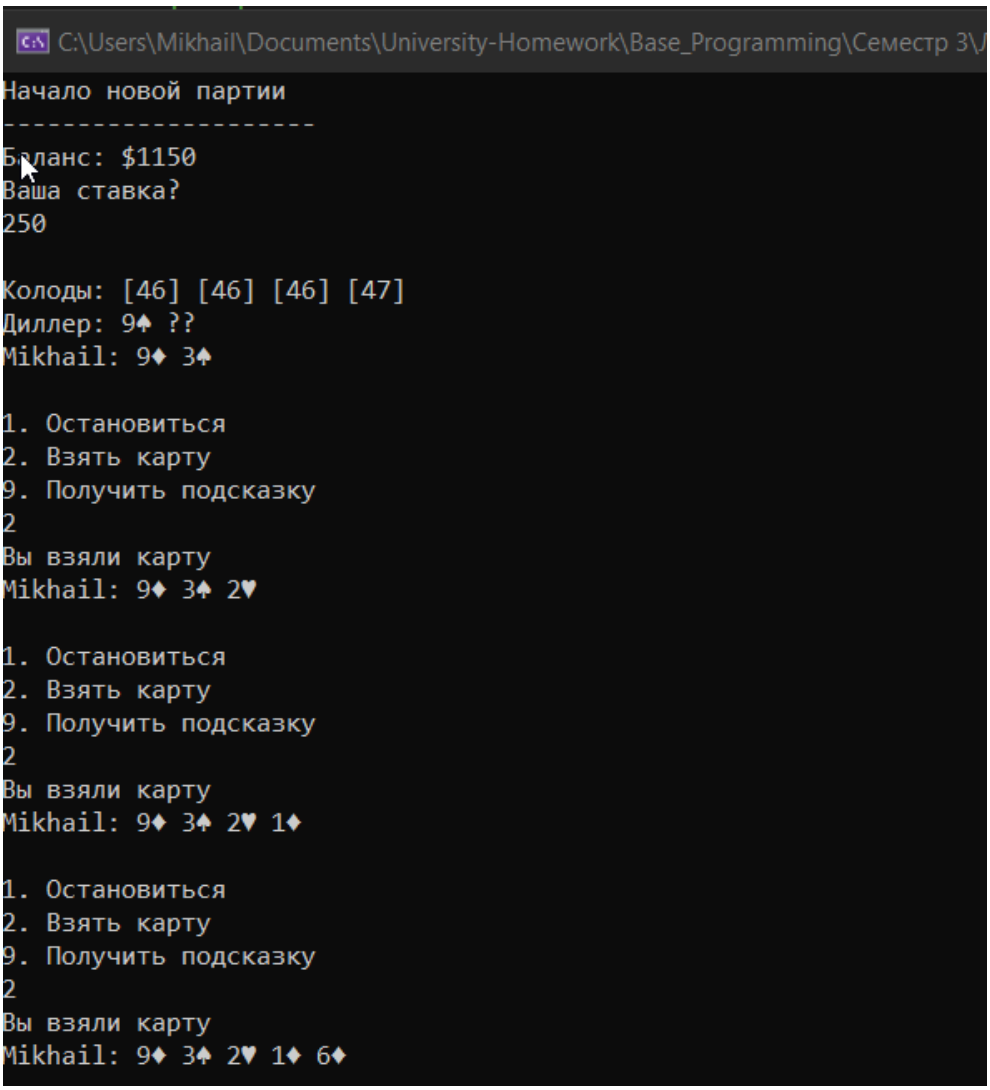
Итог: вы проиграли 250
Баланс: 2150
Сыграть еще раунд?
1 - да
2 - нет
Ваш выбор: 1

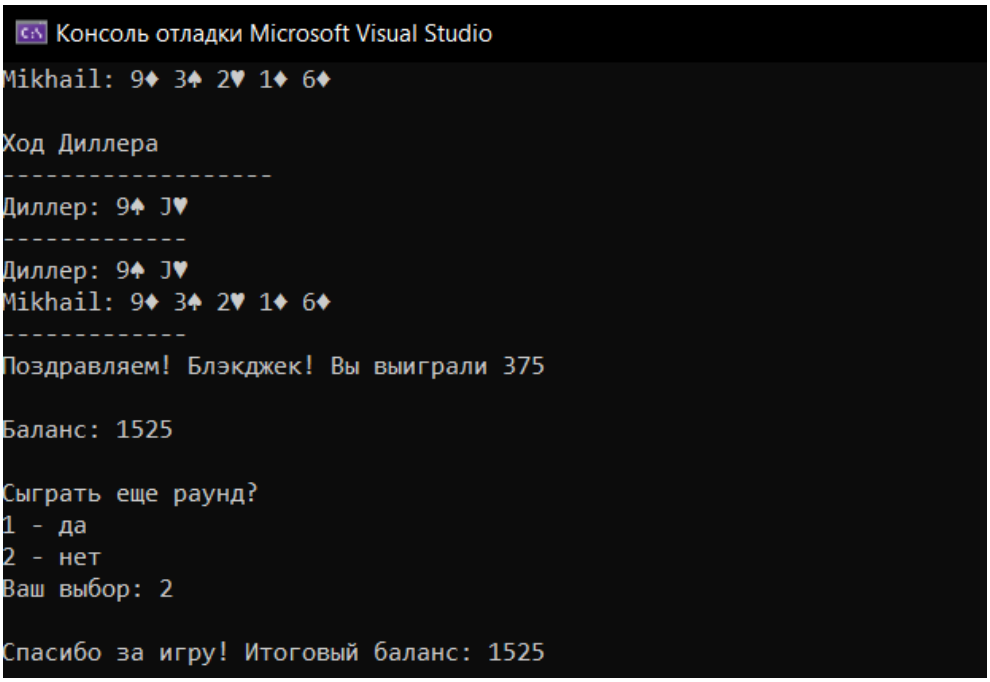

C:\Users\Mikhail\Documents\University-Homework\Base_Programming\Семестр 3\ЛР5\
Ваша ставка?
1000
Колоды: [47] [48] [48] [48]
Диллер: 3♥ ??
Mikhail: J♥ 2♦
1. Остановиться
2. Взять карту
9. Получить подсказку
2
Вы взяли карту
Mikhail: J♥ 2♦ 7♣
1. Остановиться
2. Взять карту
9. Получить подсказку
1
Вы остановились
Mikhail: J♥ 2♦ 7♣
Ход Диллера

Диллер: 3♥ 8♠
Диллер берет карту...
Диллер: 3♥ 8♠ 9♦

Диллер: 3♥ 8♠ 9♦
Mikhail: J♥ 2♦ 7♣

Итог: вы проиграли 1000
Баланс: 1150


C:\Users\Mikhail\Documents\University-Homework\Base_Programming\Семестр 3\
Начало новой партии
Баланс: $1150
Ваша ставка?
250
Колоды: [46] [46] [46] [47]
Диллер: 9♠ ??
Mikhail: 9♦ 3♠
1. Остановиться
2. Взять карту
9. Получить подсказку
2
Вы взяли карту
Mikhail: 9♦ 3♠ 2♥
1. Остановиться
2. Взять карту
9. Получить подсказку
2
Вы взяли карту
Mikhail: 9♦ 3♠ 2♥ 1♦
1. Остановиться
2. Взять карту
9. Получить подсказку
2
Вы взяли карту
Mikhail: 9♦ 3♠ 2♥ 1♦ 6♦


Консоль отладки Microsoft Visual Studio
Mikhail: 9♦ 3♠ 2♥ 1♦ 6♦
Ход Диллера
Диллер: 9♠ J♥
Диллер: 9♠ J♥
Mikhail: 9♦ 3♠ 2♥ 1♦ 6♦
Поздравляем! Блэкджек! Вы выиграли 375
Баланс: 1525
Сыграть еще раунд?
1 - да
2 - нет
Ваш выбор: 2
Спасибо за игру! Итоговый баланс: 1525

Код:

Card.h

```
#ifndef CARD_H
#define CARD_H

#include <string>
#include <iostream>

// Suit - масть
enum class Suit { HEARTS, DIAMONDS, CLUBS, SPADES };

// Ace - туз
enum class Rank {
    ACE = 11, TWO = 2, THREE = 3, FOUR = 4, FIVE = 5, SIX = 6,
SEVEN = 7,
    EIGHT = 8, NINE = 9, TEN = 10, JACK = 12, QUEEN = 13, KING = 14
};

class Card {
private:
    Suit suit;
    Rank rank;
    bool faceUp; // Нужно для определения, какие карты диллера
показывать, а какие нет

public:
    Card(Suit s, Rank r);
    int getValue() const;
    Rank getRank() const;
    bool isAce() const;
    bool isTenValue() const;
    void flip(); // Показать карту диллера
    bool isFaceUp() const; // Показано ли содержание карты. Нужно
для подсказок т.к. они делаются только на основе той информации,
    // которая есть у пользователя
    std::string suitToString() const;
    std::string rankToString() const;
    void display() const;
};

#endif
```

Card.cpp

```
#include "card.h"

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

int Card::getValue() const {

    if (isTenValue())
        return 10;
```

```
        return static_cast<int>(rank);
    }

    Rank Card::getRank() const {
        return rank;
    }

    bool Card::isAce() const {
        return rank == Rank::ACE;
    }

    bool Card::isTenValue() const {
        return rank == Rank::TEN || rank == Rank::JACK || rank ==
Rank::QUEEN || rank == Rank::KING;
    }

    void Card::flip() {
        faceUp = !faceUp;
    }

    bool Card::isFaceUp() const {
        return faceUp;
    }

    std::string Card::suitToString() const {

        switch (suit) {
        case Suit::HEARTS: return std::string(1, char(3));    // Сердце
        case Suit::DIAMONDS: return std::string(1, char(4));  // Ромб
        case Suit::CLUBS: return std::string(1, char(5));     // Клевер
        case Suit::SPADES: return std::string(1, char(6));    // пика
        default: return "?";
        }
    }

    std::string Card::rankToString() const {
        switch (rank) {
        case Rank::ACE: return "A";
        case Rank::JACK: return "J";
        case Rank::QUEEN: return "Q";
        case Rank::KING: return "K";
        default: return std::to_string(getValue());
        }
    }


    void Card::display() const {

        // Встроенная функция показа только тех карт, которые можно
видеть игроку
        if (faceUp) {
            std::cout << rankToString() << suitToString();
        }
        else {
            std::cout << "??";
        }
```

```
}
```

Dealer.h

```
#ifndef DEALER_H
#define DEALER_H

#include "participant.h"


// Player и Dealer наследуют у Participante
class Dealer : public Participant {
public:
    Dealer();
    void displayHand(bool showAll = false) const override;
    bool shouldHit() const; // Нужно для подсказок
    int getVisibleCardValue() const; // Тоже нужно для подсказок,
так как они будут на основе только видимых карт диллера
};

#endif
```

Dealer.cpp

```
#include "dealer.h"

Dealer::Dealer() : Participant() {}

void Dealer::displayHand(bool showAll) const {

    if (!hands.empty()) {
        if (showAll) {
            hands[0].display();
        }
        else {
            if (hands[0].getCardAmount() > 0) {
                hands[0].getCard(0).display();
                std::cout << " ??";
            }
        }
    }
}

bool Dealer::shouldHit() const {
    int total = hands[0].getTotal();
    bool result = total < 17;
    return result;
}

int Dealer::getVisibleCardValue() const {
    if (hands.empty()) {
        return 0;
    }
```

```
        // Проверяем, есть ли карты в первой руке
        if (hands[0].getCardAmount() == 0) {
            return 0;
        }

        // Получаем видимую карту диллера
        Card firstCard = hands[0].getCard(0);

        return firstCard.getValue();
    }
```

Deck.h

```
    #ifndef DECK_H
    #define DECK_H

    #include "card.h"
    #include <vector>
    #include <random>
    #include <ctime>

    #include <algorithm> // для shuffle
    #include <deque>

    class Deck {
    private:
        std::vector<std::deque<Card>> decks;;  // deque из stl
        std::vector<int> currentIndices; // Текущая позиция в каждой
колоде
        int nextDeckIndex; // Следующая карта для каждой колоды

    public:
        Deck();
        void initialize();
        Card giveCard();
        void shuffleDeck(int deckIndex); // shuffle - перемешать
        void displayDecksRemainingCards() const; // отобразить
оставшиеся карты в каждой колоде
        int getRemainingAmountInDeck(int deckIndex) const; //
Оставшееся колво карт в конкретной колоде
    };

    #endif
```

Deck.cpp

```
    #include "deck.h"

    Deck::Deck() {
        initialize();
    }
```

```
void Deck::initialize() {
    decks.clear();
    currentIndices.clear();

    for (int deckNum = 0; deckNum < 4; deckNum++) {
        std::deque<Card> deck;

        for (int suitIndex = 0; suitIndex < 4; suitIndex++) {
            Suit suit = static_cast<Suit>(suitIndex);

            for (int rankValue = 1; rankValue <= 13; rankValue++) {
                Rank rank = static_cast<Rank>(rankValue);
                deck.push_back(Card(suit, rank));
            }
        }

        decks.push_back(deck);
        currentIndices.push_back(0);
    }

    for (int i = 0; i < 4; i++) {
        shuffleDeck(i);
    }

    nextDeckIndex = 0;
}


void Deck::shuffleDeck(int deckIndex) {
    // Создаем генератор случайных чисел
    std::random_device rd;
    std::mt19937 g(rd());

    // Преобразуем deque в vector для shuffle
    std::vector<Card> tempVector(decks[deckIndex].begin(),
decks[deckIndex].end());
    std::shuffle(tempVector.begin(), tempVector.end(), g);


    // Возвращаемся к deque
    decks[deckIndex].clear();
    for (const auto& card : tempVector) {
        decks[deckIndex].push_back(card);
    }

    currentIndices[deckIndex] = 0;
}

// Дать одну карту и перейти к следующей колоде
Card Deck::giveCard() {
    // Если текущая колода закончилась, перетасовать ее
    if (currentIndices[nextDeckIndex] >= 52) {
        shuffleDeck(nextDeckIndex);
    }
```

```
        Card card =
decks[nextDeckIndex][currentIndices[nextDeckIndex]];
        currentIndices[nextDeckIndex]++;

        nextDeckIndex = (nextDeckIndex + 1) % 4;

        return card;
    }

    // Показать оставшиеся карты в каждой колоде
    void Deck::displayDecksRemainingCards() const {
        std::cout << "Колоды: ";
        for (int i = 0; i < 4; i++) {
            int remaining = 52 - currentIndices[i];
            std::cout << "[" << remaining << "] ";
        }

        std::cout << std::endl;
    }


    // Получить оставшиеся карты в конкретной колоде
    int Deck::getRemainingAmountInDeck(int deckIndex) const {
        if (deckIndex >= 0 && deckIndex < 4) {
            return 52 - currentIndices[deckIndex];
        }
        return 0;
    }
```

Game.h

```
    #ifndef GAME_H
    #define GAME_H

    #include "deck.h"
    #include "player.h"
    #include "dealer.h"
    #include <memory>

    #include <iostream>
    #include <limits>
    #include <algorithm>  // Для STL алгоритмов
    #include <numeric>    // Для accumulate
    #include <vector>     // Для vector
    #include <string>     // Для string
    #include <sstream>    // Для stringstream

    class Game {
    private:
        Deck deck;
        std::unique_ptr<Player> player; // unique_ptr для
автоматического освобождения памяти
        std::unique_ptr<Dealer> dealer;
        bool gameOver;
```

```
        void betResults();
        std::string getHint();
        void playPlayerHand();
        void playDealerHand();

    public:
        Game();
        void initialize();
        void playRound();
        void run(); // Нужно, чтобы все этапы игры были в классе, а не
часть в main
        void printCards();
    };

    #endif
```

Game.cpp

```
    #include "game.h"
    #include <iostream>
    #include <limits>

    Game::Game() : gameOver(false) {
        dealer = std::make_unique<Dealer>();
    }

    // initialize отдельно от конструктора, чтобы не было ввода данных
в конструкторе
    void Game::initialize() {
        std::string playerName;

        std::cout << "Тип: базовый\nДоп. правила: сплит\n----------
----------\nОсобый вариант игры: 17+4. Особенности:\n";
        std::cout << "1. При сумме карт 17+ диллер останавливается\n";
        std::cout << "2. Список карт, имеющих стоимость 10: 10, валет,
дама, король\n";
        std::cout << "3. Туз всегда стоит 11\n";
        std::cout << "4. Два туза = блэкджек!\n";
        std::cout << "5. Сплит разрешён\n";

        std::cout << "\nВведите ваше имя: ";
        std::getline(std::cin, playerName);

        player = std::make_unique<Player>(playerName);
    }

    std::string Game::getHint() {

        int playerTotal = player->getCurrentHand().getTotal();
        int dealerCardValue = dealer->getVisibleCardValue();
        bool canSplitHand = player->canSplit();


        // Если есть возможность сплита
```

```
        if (canSplitHand) {

            int cardValue = player-
>getCurrentHand().getCard(0).getValue();

            switch (cardValue) {
            case 10: return "Никогда не разделяйте карты стоимостью 10.
Остановитесь";
            case 9:
                if (dealerCardValue == 7 || dealerCardValue == 10 ||
dealerCardValue == 11) {
                    return "Не разделяйте девятки, если у дилера 7, 10
или туз. Остановитесь";
                }
                else {
                    return "Разделяйте девятки, если у диллера 2-6, 8,
9";
                }
            case 8: return "Всегда разделяйте восьмерки";
            case 7:
                if (dealerCardValue >= 2 && dealerCardValue <= 7) {
                    return "Разделяйте семерки, если у дилера 2-7";
                }
                else {

                    return "Не разделяйте семерки, если у диллера 8 и
больше в сумме. Берите карту";
                }
            case 6:
                if (dealerCardValue >= 2 && dealerCardValue <= 6) {
                    return "Разделяйте шестерки, если у диллера 2-6";
                }
                else {
                    return "Не разделяйте шестерки, если у диллера 7 и
выше. Берите карту";
                }
            case 5: return "Никогда не разделяйте пятерки. Берите карту
если у вас меньше 17";
            case 4:
                if (dealerCardValue == 5 || dealerCardValue == 6) {
                    return "Разделяйте четверки, если у диллера 5 или
6";
                }
                else {
                    return "Не разделяйте четверки. Берите карту";
                }
            case 3:
            case 2:
                if (dealerCardValue >= 2 && dealerCardValue <= 7) {
                    return "Разделяйте 2 или 3, если у диллера 2-7";
                }
                else {
                    return "Не разделяйте 2 или  3, если у диллера 8
или более. Берите карту";
                }
            }
```

```
        }

        if (playerTotal >= 17) {
            return "Если у вас больше 17, всегда останавливайтесь";
        }

        else if (playerTotal >= 13 && playerTotal <= 16) {
            if (dealerCardValue >= 2 && dealerCardValue <= 6) {
                return "Останавливайтесь, если у вас 13-16, а у диллера
2-6";
            }
            else {
                return "Берите карту, если у вас 13 -16, а у диллера 7
и более";
            }
        }
        else {
            return "Всегда берите карту на 12 или меньше";
        }
    }

    void Game::playRound() {
        player->resetRound();
        dealer->clearHands();

        if (player->getMoney() <= 0) {
            std::cout << "\nУ вас закончились деньги. Игра окончена\n";
            gameOver = true;
            return;
        }

        std::cout << "\nНачало новой партии\n--------------------\n";

        std::cout << "Баланс: $" << player->getMoney() << "\n";

        int betAmount;

        // выполняем пока пользователь не укажет допустимую сумму
        auto isValidBet = [this](int bet) -> bool { // Новый код STL
алгоритм с лямбдой
            return bet > 0 && bet <= player->getMoney();
            };

        do {
            std::cout << "Ваша ставка?\n";
            std::cin >> betAmount;
            if (!isValidBet(betAmount)) {
                std::cout << "Невозможная сумма ставки\n";
            }
        } while (!isValidBet(betAmount));

        player->placeBet(betAmount);

        // Раздача первой партии карт
        player->addCard(deck.giveCard());
        dealer->addCard(deck.giveCard());
```

```
            player->addCard(deck.giveCard());
            dealer->addCard(deck.giveCard());

            // Показать количество карт в колодах
            std::cout << "\n";
            deck.displayDecksRemainingCards();

            // Проверить на блэкджек диллера
            if (dealer->getCurrentHand().hasBlackjack()) {
                betResults();
                return;
            }

            // Проверить на блэкДжек игрока
            if (player->getCurrentHand().hasBlackjack()) {
                betResults();
                return;
            }

            // Ход игрока для каждой руки
            player->resetToFirstHand();

            while (!player->allHandsDone()) {
                if (!player->getNextHandToPlay()) {
                    break;
                }

                playPlayerHand();
            }


            // Ход диллера если у игрока есть руки, в которых не более 21
очка
            // Новый код с any_of
            const auto& hands = player->getHands();
            bool hasPlayableHands = false;

            if (std::any_of(hands.begin(), hands.end(),
                [](const Hand& hand) { return !hand.isBust(); })) {
                playDealerHand();
            }
            else {
                std::cout << "\nВы взяли слишком много карт! Диллер
выигрывает\n";
            }

            betResults();
        }

        void Game::printCards() {
            std::cout << "Диллер: ";
            dealer->displayHand(false);
            // Выводим сообщение про руку только если больше одной руки

            std::cout << "\n";
            if (player->getTotalHands() > 1) {
```

```
            std::cout << "---Рука " << (player->getCurrentHandIndex() +
1) << "---\n";
        }
        player->displayHands();
    }

    void Game::playPlayerHand() {
        bool handDone = false;

        printCards();

        while (!handDone && !player->getCurrentHand().isBust()) {


            std::cout << "\n1. Остановиться\n2. Взять карту\n";
            if (player->canSplit()) {
                std::cout << "3. Сплит\n";
            }
            std::cout << "9. Получить подсказку\n";

            int choice;
            std::cin >> choice;

            switch (choice) {
            case 1: // Остановиться
                std::cout << "Вы остановились\n";
                player->standCurrentHand();
                handDone = true;
                player->displayHands();
                break;

            case 2: // Взять карту
                player->addCard(deck.giveCard());
                std::cout << "Вы взяли карту\n";
                player->displayHands();
                break;

            case 3: // Сплит
                if (player->canSplit()) {
                    if (player->split()) {
                        std::cout << "Рука разделена!\n";

                        // Раздать карты в обе руки
                        player->addCard(deck.giveCard());

                        if (player->getHands().size() > 1) {
                            Card newCard = deck.giveCard();
                            player->getHands()[1].addCard(newCard);
                        }

                        //printCards();
                        player->displayHands();
                    }
                    else {
                        std::cout << "Невозможно сделать сплит\n";
                    }
```

```
                }
                else {
                    std::cout << "Невозможно сделать сплит\n";
                }
                break;

            case 9: // Подсказка
                std::cout << "ПОДСКАЗКА: " << getHint() << "\n";
                break;

            default:
                std::cout << "Неверный выбор\n";
                break;
            }
        }

        if (player->getCurrentHand().isBust()) {
            player->markCurrentHandDone();
        }
    }

    void Game::playDealerHand() {
        std::cout << "\nХод Диллера\n-------------------\n";
        std::cout << "Диллер: ";
        dealer->displayHand(true); // показать все карты диллера

        while (dealer->shouldHit() && !dealer->isBust()) {
            std::cout << "\nДиллер берет карту...\n";
            dealer->addCard(deck.giveCard());

            std::cout << "Диллер: ";
            dealer->displayHand(true);
        }
        std::cout << "\n";

        if (dealer->isBust()) {
            std::cout << "Диллер перебрал\n";
            // Такая ситуация однако не произойдёт, т.к. мы
обрабатываем перебор как исключение
        }
    }

    void Game::betResults() {
        int dealerTotal = dealer->getCurrentHand().getTotal();
        bool dealerBust = dealer->isBust();
        bool dealerBlackjack = dealer->getCurrentHand().hasBlackjack();


        std::cout << "-------------\n";
        std::cout << "Диллер: ";
        dealer->displayHand(true);
        // Выводим сообщение про руку только если больше одной руки

        std::cout << "\n";
        if (player->getTotalHands() > 1) {
```

```
            std::cout << "---Рука " << (player->getCurrentHandIndex() +
1) << "---\n";
        }
        player->displayHands();
        std::cout << "-------------\n";

        for (int i = 0; i < player->getTotalHands(); i++) {
            Hand& hand = player->getHands()[i];
            int playerTotal = hand.getTotal();
            bool playerBlackjack = hand.hasBlackjack();

            if (player->getTotalHands() > 1)
                std::cout << "---Рука " << i + 1 << "---\n";


            if (hand.isBust()) {
                std::cout << "Итог: вы проиграли " << player-
>getCurrentBet() << "\n";
                player->lose(i);
            }
            else if (playerBlackjack && !dealerBlackjack) {
                std::cout << "Поздравляем! Блэкджек! Вы выиграли " <<
(player->getCurrentBet() * 1.5) << "\n";

                player->win(player->getCurrentBet() * 1.5, i);
            }
            else if (dealerBlackjack && !playerBlackjack) {
                std::cout << "Итог: У диллера блэкджек, вы проиграли "
<< player->getCurrentBet() << "\n";
                player->lose(i);
            }
            else if (playerBlackjack && dealerBlackjack) {
                std::cout << "Итог: У обоих блэкджек, ничья\n";
                player->draw(i);
            }
            else if (dealerBust) {
                std::cout << "Поздравляем! Диллер перебрал, вы выиграли
" << player->getCurrentBet() << "\n";
                player->win(player->getCurrentBet(), i);
            }
            else if (playerTotal > dealerTotal) {
                std::cout << "Поздравляем! Вы выиграли " << player-
>getCurrentBet() << "\n";
                player->win(player->getCurrentBet(), i);
            }
            else if (playerTotal < dealerTotal) {
                std::cout << "Итог: вы проиграли " << player-
>getCurrentBet() << "\n";
                player->lose(i);
            }
            else {
                std::cout << "Итог: Ничья, ставка возвращена\n";
                player->draw(i);
            }
        }
```

```
        std::cout << "\nБаланс: " << player->getMoney() << "\n";
    }


    void Game::run() {
        initialize();

        while (!gameOver) {

            playRound();

            if (!gameOver) {
                std::cout << "\nСыграть еще раунд?\n1 - да\n2 -
нет\nВаш выбор: ";
                std::string choice;
                std::cin >> choice;

                // Используем вектор и алгоритм find для проверки ввода
                std::vector<std::string> validChoices = { "1", "2" };

                if (choice == "1") {
                    continue;
                }
                else if (choice == "2") {
                    gameOver = true;

                    std::cout << "\nСпасибо за игру! Итоговый баланс: "
                        << player->getMoney() << "\n";
                }
                else {
                    std::cout << "Неправильный вариант ответа\n";
                }
            }
        }
    }
```

Hand.h

```
#ifndef HAND_H
#define HAND_H

#include "card.h"
#include <vector>
#include <stdexcept>


#include <numeric> // для accumulate
#include <algorithm> // Для count_if

class Hand {
private:
    std::vector<Card> cards;

public:
```

```
        Hand() = default;
        void addCard(const Card& card); // Добавить карту в руку
        void clear(); // Убрать карты из руки
        int getTotal() const; // Общая стоимость руки

        Card getCard(int index) const; // Вернуть карту из руки под
определённым индексом
        int getCardAmount() const;
        const std::vector<Card>& getCards() const;  // Все карты руки
        bool isBust() const; // Был ли привышен лимит в 21 очко
        bool hasBlackjack() const;
        bool canSplit() const;
        void display() const;
    };

    #endif
```

Hand.cpp

```
    #include "hand.h"

    // Добавить карту в руку и проверить на перебор
    void Hand::addCard(const Card& card) {
        cards.push_back(card);
        // Исключение если перебор
        if (getTotal() > 21) {
            throw std::runtime_error("ПЕРЕБОР! Сумма карт: " +
std::to_string(getTotal()));
        }
    }

    void Hand::clear() {
        cards.clear();
    }


    int Hand::getTotal() const {
        int total = 0;
        int aceCount = 0;

        std::for_each(cards.begin(), cards.end(), [&](const Card& card)
{
            if (card.isFaceUp()) {
                total += card.getValue();
                if (card.isAce()) {
                    aceCount++;
                }
            }
            });

        if (cards.size() == 2 && aceCount == 2) {
            return 21;
        }
```

```
        return total;
    }

    int Hand::getCardAmount() const {
        return cards.size();
    }

    Card Hand::getCard(int index) const {
        if (index >= 0 && index < static_cast<int>(cards.size())) {
            return cards[index];
        }

        throw std::out_of_range("Неверный индекс карты");
    }

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

    bool Hand::isBust() const {
        return getTotal() > 21;
    }

    bool Hand::hasBlackjack() const {
        if (getTotal() == 21)
            return true;
        return false;
    }

    bool Hand::canSplit() const {
        if (cards.size() == 2) {
            return cards[0].getRank() == cards[1].getRank();
        }
        return false; // на случай, если карт не 2
    }

    void Hand::display() const {
        // Тут можно и через цикл проще, но для задания использую
for_each
        std::for_each(cards.begin(), cards.end(), [](const Card& card)
{
            card.display();
            std::cout << " ";
        });
    }
```

Participant.h

```
    #ifndef PARTICIPANT_H
    #define PARTICIPANT_H

    #include "hand.h"
    #include <vector>
```

```
// из этого класса наследуются диллер и игрок
class Participant {
protected:
    std::vector<Hand> hands;
    int currentHandIndex;

public:
    Participant();
    virtual void addCard(const Card& card);
    Hand& getCurrentHand();
    std::vector<Hand>& getHands();
    const std::vector<Hand>& getHands() const;
    void clearHands();
    bool isBust() const;
    virtual void displayHand(bool showAll = true) const;
    virtual ~Participant() = default;
};

#endif
```

Participant.cpp

```
#include "participant.h"

Participant::Participant() : currentHandIndex(0) {
    hands.emplace_back(); // добавляет новый объект Hand в конец
вектора
}

void Participant::addCard(const Card& card) {
    hands[currentHandIndex].addCard(card);
}

Hand& Participant::getCurrentHand() {
    return hands[currentHandIndex];
}

// для безопасности и совместимости 2 метода, константный и не
константный
std::vector<Hand>& Participant::getHands() {
    return hands;
}

const std::vector<Hand>& Participant::getHands() const {
    return hands;
}

void Participant::clearHands() {

    hands.clear();
    hands.emplace_back();
    currentHandIndex = 0;
}
```

```
bool Participant::isBust() const {
    return hands[currentHandIndex].isBust();
}

void Participant::displayHand(bool showAll) const {
    hands[currentHandIndex].display();
}
```

Player.h

```
#ifndef PLAYER_H
#define PLAYER_H

#include "participant.h"
#include <string>
#include <vector>

#include <list>

class Player : public Participant {
private:
    std::string name;
    int money;
    std::list<int> bets; // list вместо vector
    // может быть несколько ставок на несколько рук
    bool hasSplit; // был ли сплит
    std::vector<bool> handDone;
    std::vector<bool> handStand;

public:
    Player(std::string n);
    bool placeBet(int amount, int handIndex = 0);
    bool split();
    bool canSplit() const;
    bool hasCurrentHandStood() const; // Была ли остановлена
текущая рука
    void standCurrentHand(); // Остановить текущую руку, больше
карт не брать
    void markCurrentHandDone(); // Завершить партию для
    void resetToFirstHand(); // выбрать первую руку
    bool allHandsDone() const; // Завершить партию для всех рук
    void resetRound();
    void win(int amount, int handIndex = 0);
    void lose(int handIndex = 0);
    void draw(int handIndex = 0);
    bool getNextHandToPlay();

    // Методы получения данных
    std::string getName() const;
    int getMoney() const;
    int getCurrentBet() const;
    bool getHasSplit() const;
    int getCurrentHandIndex() const;
    int getTotalHands() const;
```

```
        void displayHands() const;
        int getBetForHand(int handIndex) const;
    };

    #endif
```

Player.cpp

```
    #include "player.h"

    Player::Player(std::string n) : name(std::move(n)), money(2500),
hasSplit(false) {

        // name(std::move(n)) передаёт владение строкой без копирования
        bets.push_back(0);
        handDone.push_back(false);
        handStand.push_back(false);
    }

    bool Player::placeBet(int amount, int handIndex) {
        if (money < amount) return false;

        // Если нужно, расширяем список
        while (handIndex >= static_cast<int>(bets.size())) {
            bets.push_back(0);
        }

        auto iter = bets.begin();
        std::advance(iter, handIndex);

        money -= amount;
        *iter = amount;

        return true;
    }


    bool Player::split() {
        // Нужно чтобы руку можно было поделить, она не была уже
поделена и количество карт было равно 2
        if (hands[currentHandIndex].canSplit() && !hasSplit &&
            hands[currentHandIndex].getCardAmount() == 2) {

            Hand newHand;
            Card secondCard = hands[currentHandIndex].getCard(1);
            newHand.addCard(secondCard);

            Hand currentHand;
            currentHand.addCard(hands[currentHandIndex].getCard(0));
            hands[currentHandIndex] = currentHand;

            hands.push_back(newHand);

            bets.push_back(getCurrentBet());
```

```
            money -= getCurrentBet();

            handDone.push_back(false);
            handStand.push_back(false);

            hasSplit = true;
            return true;
        }
        return false;
    }

    bool Player::canSplit() const {
        return hands[currentHandIndex].canSplit() && !hasSplit &&
            hands[currentHandIndex].getCardAmount() == 2;
    }

    bool Player::hasCurrentHandStood() const {
        // проверка, что не будет выхода за границу вектора handStand
        if (currentHandIndex < static_cast<int>(handStand.size())) {
            return handStand[currentHandIndex];

        }
        return false;
    }

    void Player::standCurrentHand() {
        if (currentHandIndex < static_cast<int>(handStand.size())) {
            handStand[currentHandIndex] = true;
            handDone[currentHandIndex] = true;
        }
    }

    void Player::markCurrentHandDone() {
        if (currentHandIndex < static_cast<int>(handDone.size())) {
            handDone[currentHandIndex] = true;
        }
    }

    bool Player::getNextHandToPlay() {
        for (size_t i = 0; i < handDone.size(); i++) {
            if (!handDone[i] && !hands[i].isBust()) {
                currentHandIndex = i;
                return true;
            }
        }
        return false;
    }

    void Player::resetToFirstHand() {
        currentHandIndex = 0;
    }

    bool Player::allHandsDone() const {
        // Используем all_of
        return std::all_of(handDone.begin(), handDone.end(), [](bool
done) {
```

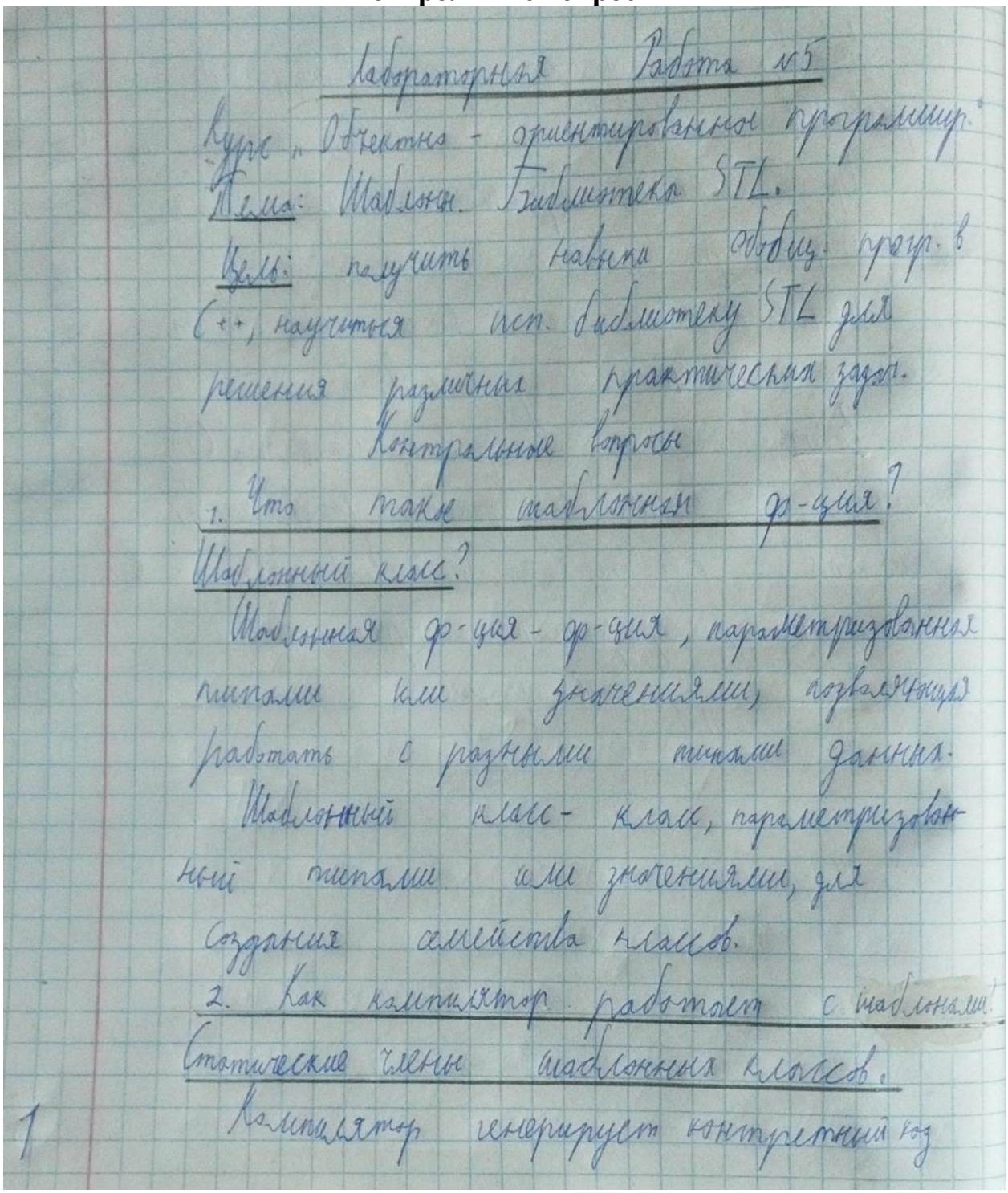
```
        return done;
    });
}

void Player::resetRound() {
    clearHands();
    bets.clear();
    bets.push_back(0);
    handDone.clear();

    handDone.push_back(false);
    handStand.clear();
    handStand.push_back(false);
    currentHandIndex = 0;
    hasSplit = false;
}

void Player::win(int amount, int handIndex) {

    if (handIndex < static_cast<int>(bets.size())) {
        auto iter = bets.begin();
        std::advance(iter, handIndex);
        money += amount + (*iter);
        *iter = 0;
    }
}

void Player::lose(int handIndex) {
    if (handIndex < static_cast<int>(bets.size())) {
        auto iter = bets.begin();
        std::advance(iter, handIndex);
        *iter = 0;
    }
}

void Player::draw(int handIndex) {
    if (handIndex < static_cast<int>(bets.size())) {
        auto iter = bets.begin();
        std::advance(iter, handIndex);
        money += *iter;
        *iter = 0;
    }
}

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

int Player::getMoney() const {
    return money;
}

int Player::getCurrentBet() const {
    if (currentHandIndex < static_cast<int>(bets.size())) {
        auto iter = bets.begin();
        std::advance(iter, currentHandIndex);
```

```
            return *iter;
        }
        return 0;
    }

    bool Player::getHasSplit() const {
        return hasSplit;
    }


    int Player::getCurrentHandIndex() const {
        return currentHandIndex;
    }

    int Player::getTotalHands() const {
        return hands.size();
    }

    void Player::displayHands() const {
        std::cout << name << ": ";
        for (size_t i = 0; i < hands.size(); i++) {
            if (i > 0) std::cout << "  ";
            hands[i].display();
        }
        std::cout << std::endl;
    }

    int Player::getBetForHand(int handIndex) const {
        if (handIndex >= 0 && handIndex <
static_cast<int>(bets.size())) {
            auto iter = bets.begin();
            std::advance(iter, handIndex);
            return *iter;
        }
        return 0;
    }
```

Main.cpp

```
#include "game.h"

int main() {
    setlocale(LC_ALL, "Russian");

    // Рандом
    std::srand(static_cast<unsigned int>(std::time(nullptr)));

    // Всё делается внутри класса
    Game blackjackGame;
    blackjackGame.run();

    return 0;
}
```

Контрольные вопросы



Лабораторная Работа №5

Курс „Объектно - ориентированное программир."

Тема: Шаблоны. Библиотека STL.

Цель: получить навыки обобщ. прогр. в C++, научиться исп. библиотеку STL для решения различных практических задач.

Контрольные вопросы

1. Что такое шаблонная ф-ция? Шаблонный класс?

Шаблонная ф-ция - ф-ция, параметризованная типами или значениями, позволяющая работать с разными типами данных.

Шаблонный класс - класс, параметризованный типами или значениями, для создания семейства классов.

2. Как компилятор работает с шаблонами. Статические члены шаблонных классов.

Компилятор генерирует конкретный код

для каждого уникального набора параметров шаблона при его исп.

Статические члены шаблонного класса существ. отдельно для каждой инстанциации шаблона (для каждого типа Т свой экземпляр статического поля).

3. Что означает явная специализация шаблона? Частичная специализация шаблона?

Явная специализация - полное опред. шаблона для конкретного типа (значения параметров.

Частичная специализация - специализация шаблона с некот., но не всеми, заданными параметрами.

4. Что означает идиома SFINAE?

Substitution Failure Is ~~A~~ Not An Error - правило, при к-м неудачная подстановка типа в шаблон не вызывает ошибку

Кравченко Михаил ИВТ-1

2

компиляции, а исключает этому специализ
ац расш. Исп. для условной
компиляции и ог. шаблонов.

5. Кратко описать типы контейнеров STL.

Послед: vector, list, deque, array, forward_list
Ассоциативные: set, map, multiset, multimap
Неупоряд ассоциативные: unordered_set, unordered_map
и их multi-версии
Адаптеры: stack, queue, priority-queue.

6. Чем отличаются контейнеры vector, deque и list? Как они реализованы в STL?

- vector - динамич. массив с быстрым случ. доступом, вставка/удаление в конец $O(1)$, в середине $O(n)$.
- deque - двусторонняя очередь, реализована как массив указателей на блоки, вставка/

удаление в начале и конце O(1).

- list - двусвязный список, вставка/удаление в любом месте O(1), но нет случайного доступа.

7. Кратко опишите типы итераторов STL и методы работы с ними.

Типы:

- Input/Output - однонапр., только чтение/запись,
- Forward - однонапр. многопроходные,
- Bidirectional - двунапр. (list, set, map),
- Random Access - произв. доступ (vector, deque, array). Итераторы абстрагируют доступ к элементам контейнеров, исп. в алг. STL.

3. Что такое ассоциативный массив? Кратко опишите контейнеры map, multimap, set.

Кравец Михаил ИВТ-4

9

Ассоциативный массив - контейнер, отобр. ключи на значения.

map - уникальные ключи, пары ключ-значение упорядочен по ключам

multimap - допускает дублирование ключей

set - хранит только уникальные ключи (значения - сами ключи).

9. Кратко опишите адаптеры контейнеров stack, queue, priority-queue.

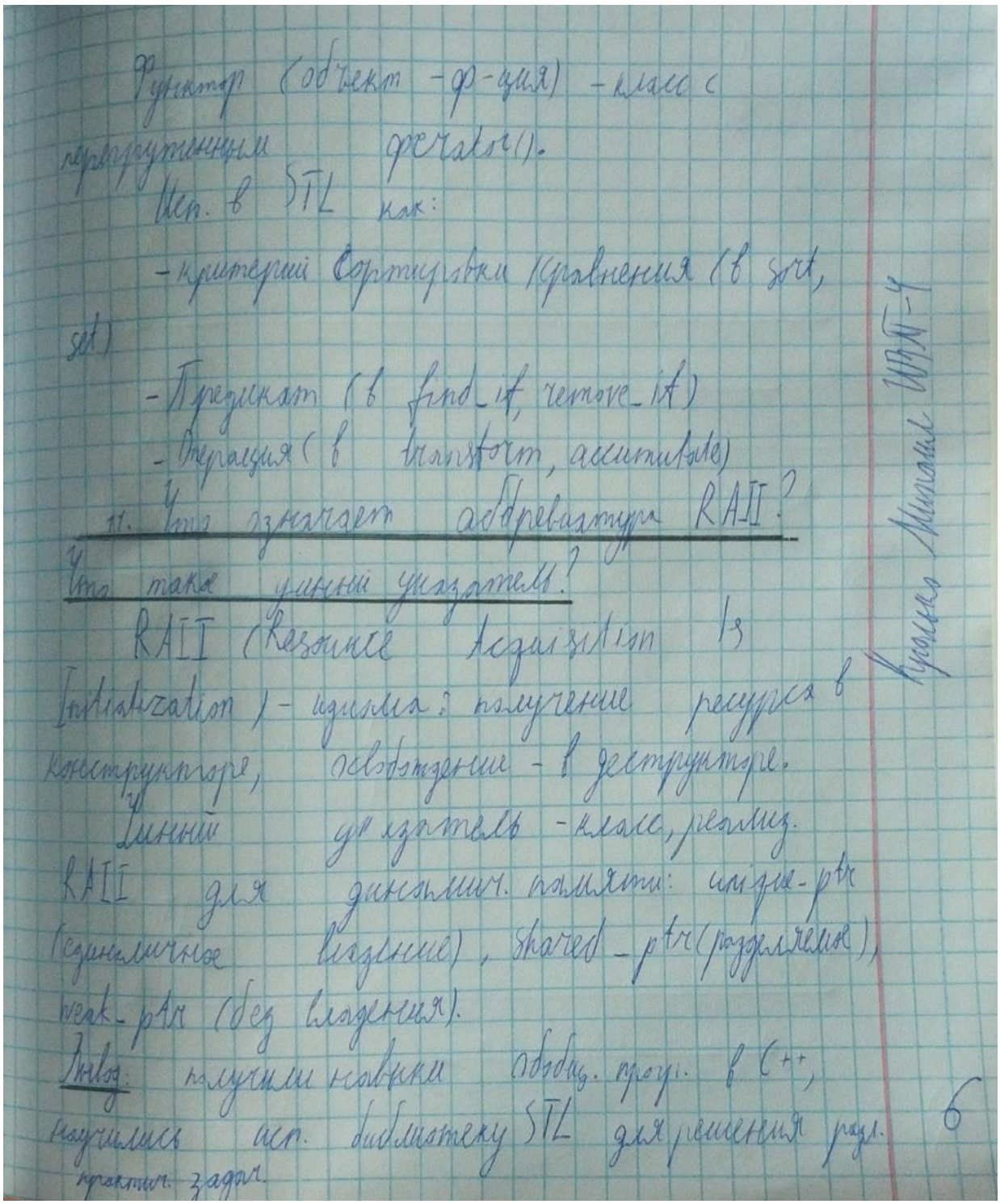
- stack LIFO (last input, first output), адаптирует deque, vector или list.

- queue - FIFO, адаптирует deque или list.

- priority-queue - очередь с приоритетом, использует vector или deque с heap.

10. Что такое функтор? Где и как функторы исп. в библиотеке STL?

5


Функтор (объект - ф-ция) - класс с перегруженным оператором ().
Исп. в STL как:
- критерий сортировки (сравнения (в sort, set)
- Предикат (в find_if, remove_if)
- Операция (в transform, accumulate)

11. Что означает аббревиатура RAII?
Что такое умный указатель?

RAII (Resource Acquisition Is Initialization) - идиома: получение ресурса в конструкторе, освобождение - в деструкторе.
Умный указатель - класс, реализ. RAII для динамич. памяти: unique_ptr (единоличное владение), shared_ptr (разделяемое), weak_ptr (без владения).
Вывод: получили навыки обобщ. прогр. в C++, научились исп. библиотеку STL для решения разл. практич. задач.

Кравец Михаил ИВТ-4

6

Вывод: получили навыки обобщенного программирования в C++, научились использовать библиотеку STL для решения различных практических задач.