Министерство образования и науки Кыргызской Республики

Кыргызский государственный технический университет

им. И.Раззакова

Факультет информационных технологий

Кафедра «Программное обеспечение компьютерных систем»

Направление: 710400 «Программная инженерия»

Дисциплина: « Объектно-ориентрованное проектирование / Объектно-ориентрованное программирование»

**ОТЧЕТ**

По лабораторной работе №2

Тема: «Классы и объекты»

Выполнила: студентка группы

ПИ(б)-3-21 Пак Ксения

Проверил: Мусабаев Э. Б.

Бишкек – 2024

Задания

1. В здании аэропорта на экранах отображается информация о самолетах, а именно: информация о пункте отправления, пункте назначения, номере рейса, времени прибытия, времени отправления, номере секции для регистрации. Экраны – это средство, которое помогает своевременно зарегистрировать и отправить пассажиров. Важно, чтобы информация на экранах была понятной и верной.

Создайте необходимую информацию в виде таблицы для такого экрана, с помощью класса Aeroflot, содержащего в описании следующие поля: номер рейса; название пункта отправления; название пункта назначения рейса; время прибытия; время отправления; место регистрации.

Напишите код программы, выполняющей следующие действия: ввод с клавиатуры значений полей объектов; сортировку записей в таблице в алфавитном порядке по названию пунктов назначения; вывод на консоль значений полей объектов класса в виде таблицы рейсов; если таких рейсов нет, выдать соответствующее сообщение.

Спецификации требований к ПО "Информационные Экраны Аэропорта"

1. Цель

Целью данного программного продукта является обеспечение функциональности для управления информацией о рейсах на информационных экранах аэропорта. Это включает в себя ввод данных, сортировку по названию пунктов назначения и вывод информации на экраны.

2. Основные требования

2.1 Создание и ввод данных о рейсах

ПО должно предоставлять возможность ввода информации о рейсах, включая номер рейса, пункт отправления, пункт назначения, время прибытия, время отправления и номер секции для регистрации.

2.2 Сортировка записей

ПО должно обеспечивать сортировку записей в таблице рейсов в алфавитном порядке по названию пунктов назначения.

2.3 Вывод информации

ПО должно предоставлять возможность вывода информации о рейсах на информационные экраны аэропорта в виде таблицы.

2.4 Обработка отсутствующих рейсов

ПО должно предоставлять сообщение, если нет доступной информации о рейсах для отображения.

3. Интерфейс пользователя

ПО должно содержать меню, предоставляющее пользователю возможность ввода данных, сортировки, вывода информации и выхода из программы.

#include <string>

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

class Aeroflot {

private:

int flightNumber;

string departureCity;

string destinationCity;

string departureTime;

string arrivalTime;

int registrationSection;

public:

Aeroflot(int number, const string& departure, const string& destination,

const string& arrival, const string& departureTime, int section)

: flightNumber(number), departureCity(departure), destinationCity(destination),

arrivalTime(arrival), departureTime(departureTime), registrationSection(section) {}

void inputFlightData() {

cout << "Enter Flight Number: ";

cin >> flightNumber;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Enter Departure City: ";

getline(cin, departureCity);

cout << "Enter Destination City: ";

getline(cin, destinationCity);

cout << "Enter Arrival Time: ";

getline(cin, arrivalTime);

cout << "Enter Departure Time: ";

getline(cin, departureTime);

cout << "Enter Registration Section: ";

cin >> registrationSection;

}

void displayFlightData() const {

cout << flightNumber << "\t\t" << departureCity << "\t\t" << destinationCity << "\t"

<< arrivalTime << "\t\t" << departureTime << "\t\t" << registrationSection << endl;

}

const string& getDestinationCity() const {

return destinationCity;

}

};

bool compareFlights(const Aeroflot& flight1, const Aeroflot& flight2) {

return flight1.getDestinationCity() < flight2.getDestinationCity();

}

int main() {

vector<Aeroflot> flights;

int numFlights;

cout << "Enter the number of flights: ";

cin >> numFlights;

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

cout << "\nEnter details for Flight " << i + 1 << ":\n";

Aeroflot flight(0, "", "", "", "", 0);

flight.inputFlightData();

flights.push\_back(flight);

}

sort(flights.begin(), flights.end(), compareFlights);

cout << "\nFlight Number\tDeparture City\tDestination City\tArrival Time\tDeparture Time\tRegistration Section\n";

for (const auto& flight : flights) {

flight.displayFlightData();

}

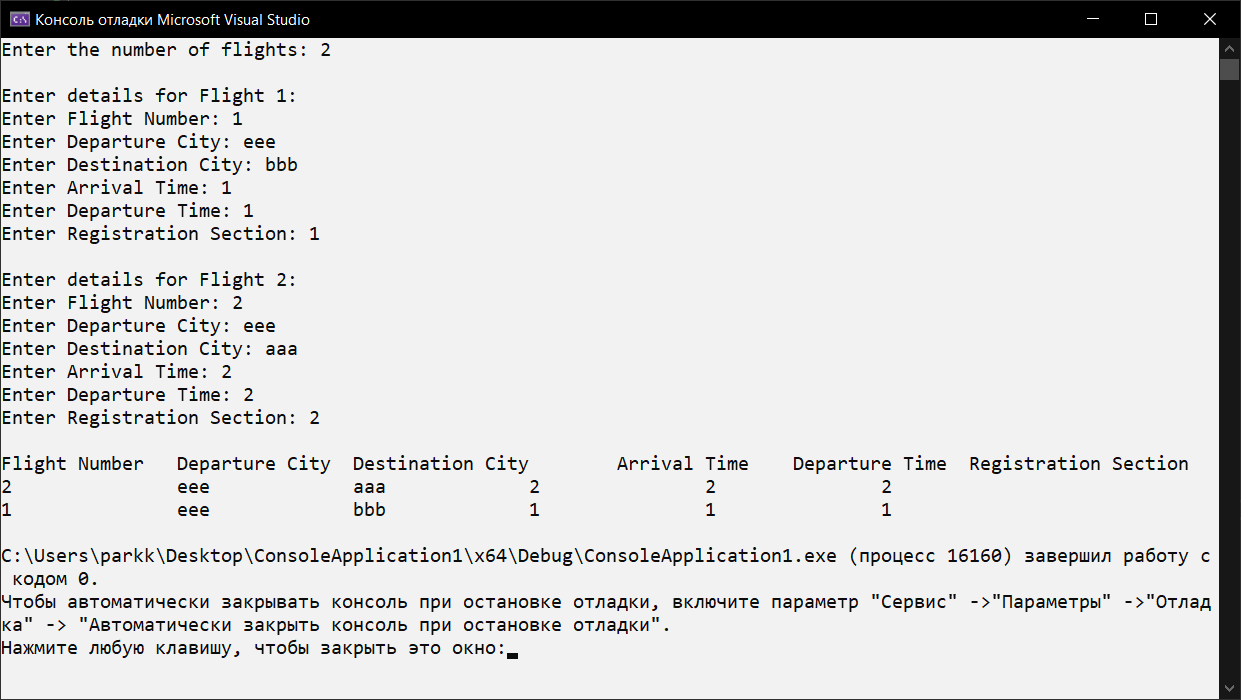
if (flights.empty()) {

cout << "\nNo flights available.\n";

}

return 0;

}



1. **Опишите класс «студенческая группа».**

**Предусмотрите возможность: работы с переменным числом студентов; поиска студента по какому-либо признаку (например, по фамилии, дате рождения или номеру телефона); добавления или удаления записей; сортировки по разным полям.**

**Разработайте программу, демонстрирующую работу с этим классом.**

**Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.**

Спецификации требований к ПО "Студенческая группа"

1. Цель

Целью данного программного продукта является обеспечение функциональности для работы с информацией о студенческой группе, включая добавление, удаление, поиск и сортировку студентов.

2. Основные требования

2.2 Поиск студента

ПО должно предоставлять возможность поиска студента по фамилии, имени, дате рождения и номеру телефона.

2.3 Добавление и удаление записей

ПО должно позволять пользователю добавлять новых студентов в группу.

ПО должно позволять пользователю удалять студентов из группы.

2.4 Сортировка по разным полям

ПО должно предоставлять возможность сортировки студентов по фамилии, имени, дате рождения и номеру телефона.

3. Интерфейс пользователя

ПО должно содержать меню, предоставляющее пользователю возможность выбора операций, таких как добавление, удаление, поиск и сортировка студентов.

#include <iostream>

#include <vector>

#include <algorithm>

#include <string>

#include <iomanip>

#include <sstream>

using namespace std;

class Student {

private:

string lastName;

string firstName;

string birthDate;

string phoneNumber;

public:

Student(const string& last, const string& first, const string& birth, const string& phone)

: lastName(last), firstName(first), birthDate(birth), phoneNumber(phone) {}

const string& getLastName() const {

return lastName;

}

const string& getFirstName() const {

return firstName;

}

const string& getBirthDate() const {

return birthDate;

}

const string& getPhoneNumber() const {

return phoneNumber;

}

void displayStudent() const {

cout << "Last Name: " << lastName << " | First Name: " << firstName

<< " | Birth Date: " << birthDate << " | Phone Number: " << phoneNumber << endl;

}

};

class StudentGroup {

private:

vector<Student> students;

public:

void addStudent(const string& last, const string& first, const string& birth, const string& phone) {

students.emplace\_back(last, first, birth, phone);

}

void removeStudent(const string& lastName) {

auto it = remove\_if(students.begin(), students.end(),

[lastName](const Student& student) { return student.getLastName() == lastName; });

students.erase(it, students.end());

}

void displayAllStudents() const {

if (students.empty()) {

cout << "No students in the group.\n";

}

else {

for (const auto& student : students) {

student.displayStudent();

}

}

}

// search

void searchByLastName(const string& lastName) const {

auto it = find\_if(students.begin(), students.end(),

[lastName](const Student& student) { return student.getLastName() == lastName; });

if (it != students.end()) {

cout << "Student found:\n";

it->displayStudent();

}

else {

cout << "Student not found.\n";

}

}

void searchByFirstName(const string& firstName) const {

auto it = find\_if(students.begin(), students.end(),

[firstName](const Student& student) { return student.getFirstName() == firstName; });

if (it != students.end()) {

cout << "Student found:\n";

it->displayStudent();

}

else {

cout << "Student not found.\n";

}

}

void searchByBirthDate(const string& birthDate) const {

auto it = find\_if(students.begin(), students.end(),

[birthDate](const Student& student) { return student.getBirthDate() == birthDate; });

if (it != students.end()) {

cout << "Student found:\n";

it->displayStudent();

}

else {

cout << "Student not found.\n";

}

}

void searchByPhoneNumber(const string& phoneNumber) const {

auto it = find\_if(students.begin(), students.end(),

[phoneNumber](const Student& student) { return student.getPhoneNumber() == phoneNumber; });

if (it != students.end()) {

cout << "Student found:\n";

it->displayStudent();

}

else {

cout << "Student not found.\n";

}

}

// sort

void sortByLastName() {

sort(students.begin(), students.end(),

[](const Student& s1, const Student& s2) { return s1.getLastName() < s2.getLastName(); });

}

void sortByFirstName() {

sort(students.begin(), students.end(),

[](const Student& s1, const Student& s2) { return s1.getFirstName() < s2.getFirstName(); });

}

void sortByBirthDate() {

sort(students.begin(), students.end(), [](const Student& s1, const Student& s2) {

std::tm tm1 = {}, tm2 = {};

std::istringstream ss1(s1.getBirthDate()), ss2(s2.getBirthDate());

ss1 >> std::get\_time(&tm1, "%Y-%m-%d");

ss2 >> std::get\_time(&tm2, "%Y-%m-%d");

return std::mktime(&tm1) < std::mktime(&tm2);

});

}

void sortByPhoneNumber() {

sort(students.begin(), students.end(),

[](const Student& s1, const Student& s2) { return s1.getPhoneNumber() < s2.getPhoneNumber(); });

}

};

int main() {

StudentGroup group;

int choice;

do {

cout << "\nMenu:\n";

cout << "1. Add Student\n";

cout << "2. Remove Student\n";

cout << "3. Display All Students\n";

cout << "4. Search Student\n";

cout << "5. Sort Students\n";

cout << "0. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

string last, first, birth, phone;

cout << "Enter Last Name: ";

cin >> last;

cout << "Enter First Name: ";

cin >> first;

cout << "Enter Birth Date: ";

cin >> birth;

cout << "Enter Phone Number: ";

cin >> phone;

group.addStudent(last, first, birth, phone);

break;

}

case 2: {

string lastName;

cout << "Enter Last Name of the Student to Remove: ";

cin >> lastName;

group.removeStudent(lastName);

break;

}

case 3:

group.displayAllStudents();

break;

case 4: {

int searchChoice;

cout << "\nSearch Options:\n";

cout << "1. By Last Name\n";

cout << "2. By First Name\n";

cout << "3. By Birth Date\n";

cout << "4. By Phone Number\n";

cout << "Enter your choice: ";

cin >> searchChoice;

switch (searchChoice) {

case 1: {

string lastName;

cout << "Enter Last Name of the Student to Search: ";

cin >> lastName;

group.searchByLastName(lastName);

break;

}

case 2: {

string firstName;

cout << "Enter First Name of the Student to Search: ";

cin >> firstName;

group.searchByFirstName(firstName);

break;

}

case 3: {

string birthDate;

cout << "Enter Birth Date of the Student to Search: ";

cin >> birthDate;

group.searchByBirthDate(birthDate);

break;

}

case 4: {

string phoneNumber;

cout << "Enter Phone Number of the Student to Search: ";

cin >> phoneNumber;

group.searchByPhoneNumber(phoneNumber);

break;

}

default:

cout << "Invalid choice. Please try again.\n";

}

break;

}

case 5: {

int sortChoice;

cout << "\nSort Options:\n";

cout << "1. By Last Name\n";

cout << "2. By First Name\n";

cout << "3. By Birth Date\n";

cout << "4. By Phone Number\n";

cout << "Enter your choice: ";

cin >> sortChoice;

switch (sortChoice) {

case 1:

group.sortByLastName();

cout << "Students sorted by Last Name.\n";

break;

case 2:

group.sortByFirstName();

cout << "Students sorted by First Name.\n";

break;

case 3:

group.sortByBirthDate();

cout << "Students sorted by Birth Date.\n";

break;

case 4:

group.sortByPhoneNumber();

cout << "Students sorted by Phone Number.\n";

break;

default:

cout << "Invalid choice. Please try again.\n";

}

break;

}

case 0:

cout << "Exiting...\n";

break;

default:

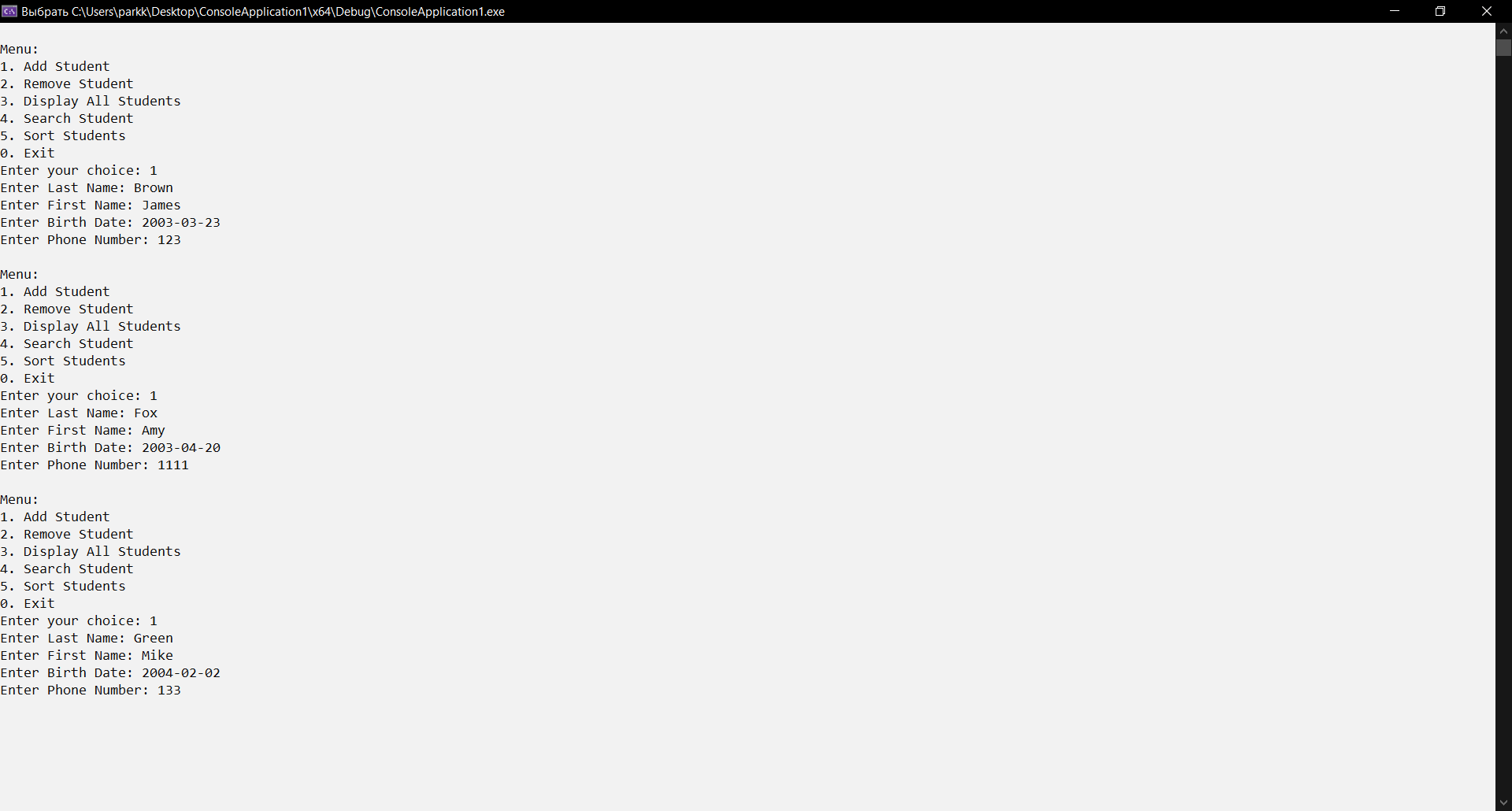
cout << "Invalid choice. Please try again.\n";

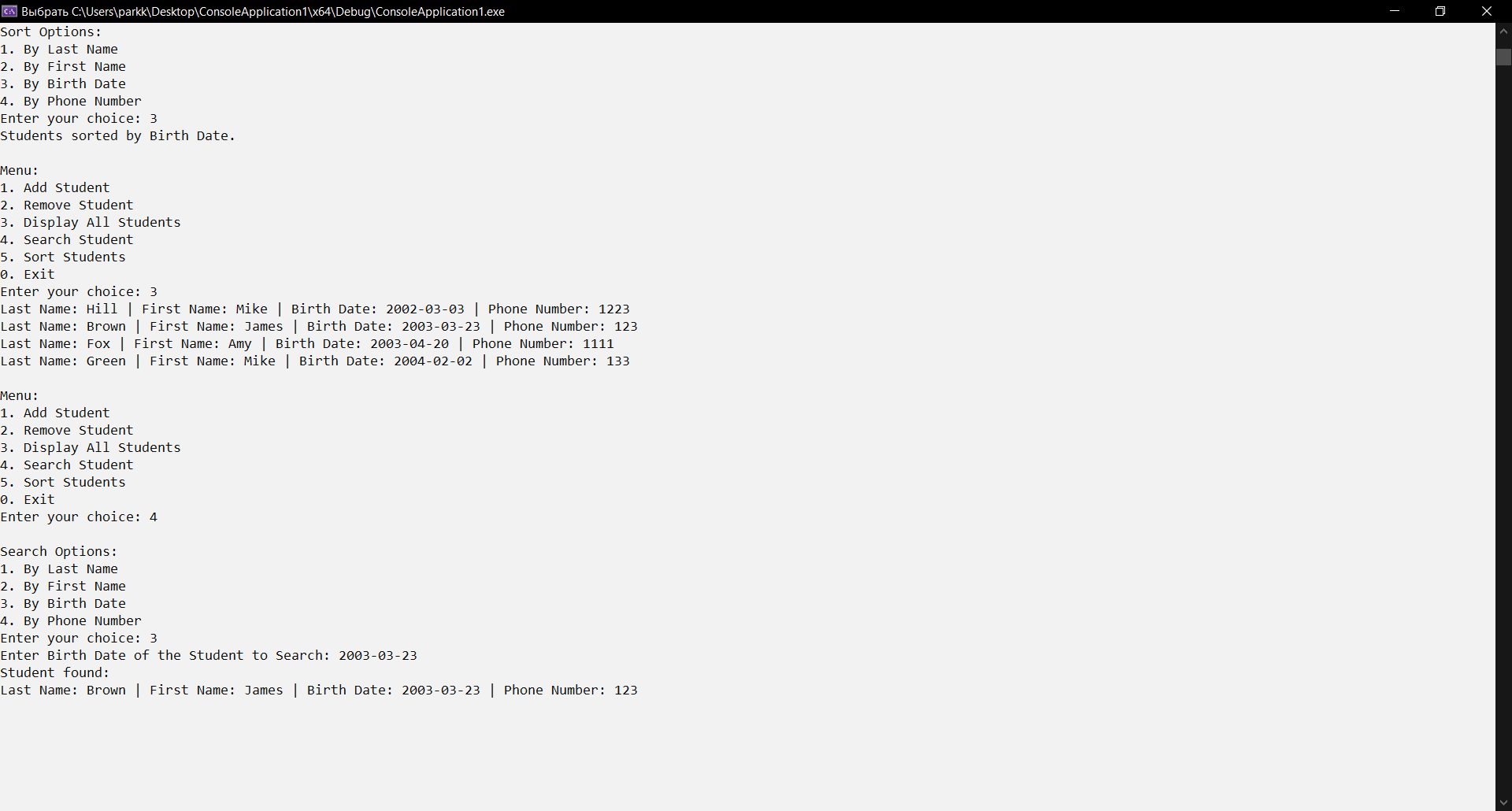
}

} while (choice != 0);

return 0;

}

1. **Опишите класс, реализующий стек (Stack).**

**Разработать программу, использующую этот класс для моделирования Т-образного сортировочного узла на железной дороге. Программа должна разделять на два направления состав, состоящий из вагонов двух типов (на каждое направление формируется состав из вагонов одного типа). Предусмотреть возможность формирования состава из файла или с клавиатуры. Возможно использование стандартных функций при работе со стеком в виде контейнера из библиотеки STL:**

**push() - поместить элемент в вершину стека;**

**pop() - удалить элемент из вершины стека.**

#include <iostream>

#include <fstream>

#include <stack>

#include <string>

using namespace std;

class TrainSortingNode {

private:

std::stack<int> leftDirection;

std::stack<int> rightDirection;

public:

void addWagon(int wagon, bool isLeftDirection) {

if (isLeftDirection) {

leftDirection.push(wagon);

std::cout << "Added wagon " << wagon << " to the left direction.\n";

}

else {

rightDirection.push(wagon);

std::cout << "Added wagon " << wagon << " to the right direction.\n";

}

}

void removeWagon(bool isLeftDirection) {

if (isLeftDirection) {

if (!leftDirection.empty()) {

std::cout << "Removed wagon " << leftDirection.top() << " from the left direction.\n";

leftDirection.pop();

}

else {

std::cout << "Left direction is empty. Cannot remove wagon.\n";

}

}

else {

if (!rightDirection.empty()) {

std::cout << "Removed wagon " << rightDirection.top() << " from the right direction.\n";

rightDirection.pop();

}

else {

std::cout << "Right direction is empty. Cannot remove wagon.\n";

}

}

}

void displayContents() const {

cout << "Left Direction: ";

displayStackContents(leftDirection);

cout << "Right Direction: ";

displayStackContents(rightDirection);

}

private:

// Helper method to display the contents of a stack

void displayStackContents(const stack<int>& directionStack) const {

if (directionStack.empty()) {

cout << "Empty";

}

else {

stack<int> tempStack = directionStack;

while (!tempStack.empty()) {

cout << tempStack.top() << " ";

tempStack.pop();

}

}

cout << "\n";

}

};

int main() {

TrainSortingNode sortingNode;

int choice;

do {

std::cout << "\nMenu:\n";

std::cout << "1. Add Wagon\n";

std::cout << "2. Remove Wagon\n";

std::cout << "3. Display Contents\n";

std::cout << "0. Exit\n";

std::cout << "Enter your choice: ";

std::cin >> choice;

switch (choice) {

case 1: {

int wagon, directionChoice;

std::cout << "Enter Wagon Number: ";

std::cin >> wagon;

std::cout << "Select Direction (1. Left / 2. Right): ";

std::cin >> directionChoice;

bool isLeftDirection = (directionChoice == 1);

sortingNode.addWagon(wagon, isLeftDirection);

break;

}

case 2: {

int directionChoice;

std::cout << "Select Direction to Remove Wagon (1. Left / 2. Right): ";

std::cin >> directionChoice;

bool isLeftDirection = (directionChoice == 1);

sortingNode.removeWagon(isLeftDirection);

break;

}

case 3:

sortingNode.displayContents();

break;

case 0:

std::cout << "Exiting...\n";

break;

default:

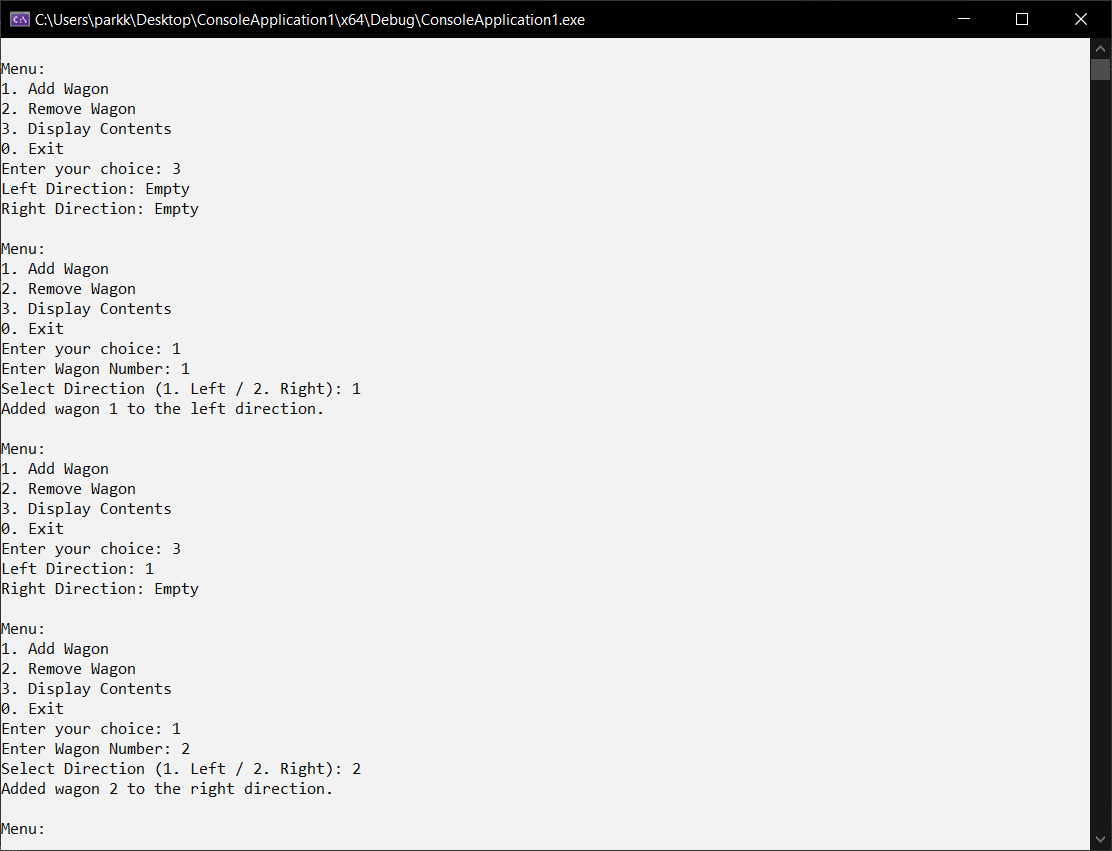
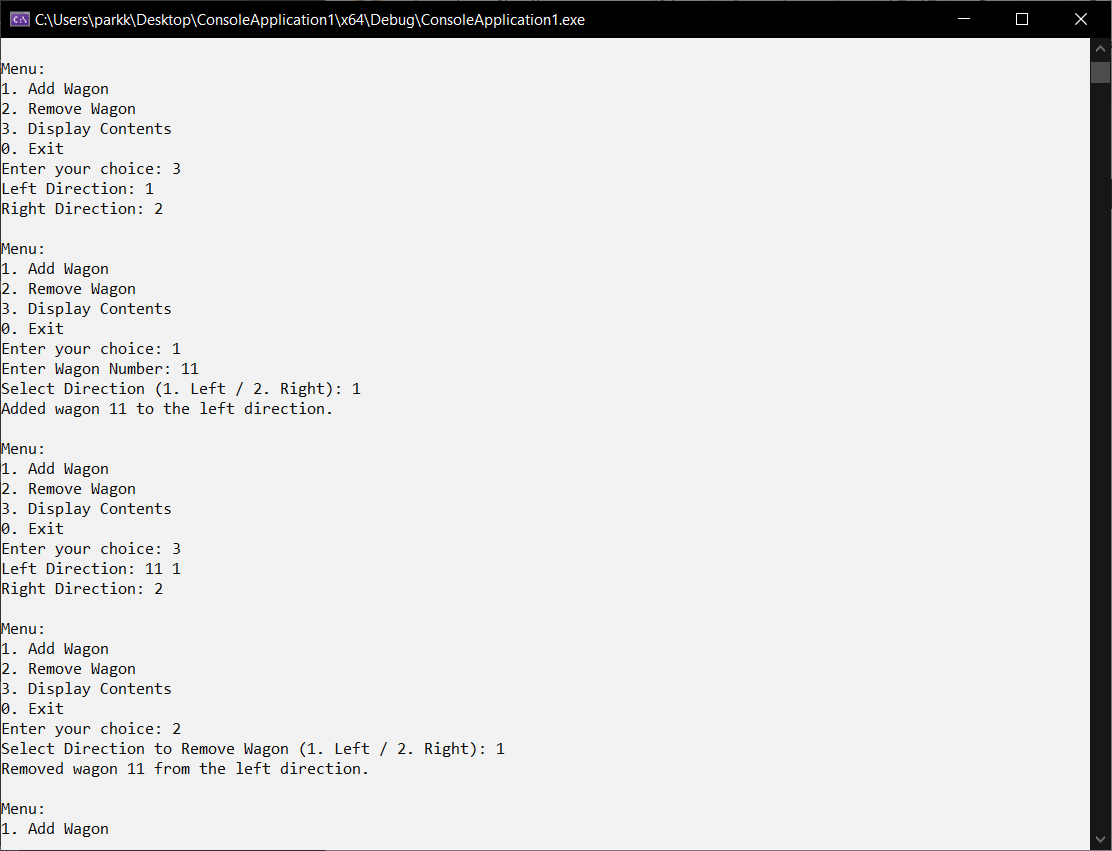
std::cout << "Invalid choice. Please try again.\n";

}

} while (choice != 0);

return 0;

}

Ответы на вопросы

1. Как вы понимаете, что такое класс и объект? Приведите примеры.

**Класс (class)** - это абстрактный тип данных, заданный программистом. Класс описывает структуру своих объектов (поля) и их поведение (методы). Класс представляет модель реального объекта в виде данных и функций для работы с ними. Данные класса называются полями, а функции класса методами.

1. Каков синтаксис и смысл определения класса? Как называются составные части класса?

Общий вид синтаксиса класса**:**

*// имя должно быть уникальным среди имен других классов, определенных в вашей программе.*

**сlass <**имя\_ класса**>**

**{**

**private:**

<сокрытые поля и методы класса>

**public:**

<общедоступные поля и методыкласса**>**

**} [**список объектов]**;**

**Смысл определения класса**: фактически определение класса задает внутреннюю структуру объектов класса, т.е. из каких полей они состоят и какими методами пользуется объект класса при работе с этими полями. При определении класса не происходит выделения физической памяти. Память отводится при объявлении (создании) объекта.

Поля и методы класса называют **членами** или **элементами класса**.

1. Что такое идентификаторы доступа к элементам класса? Дайте определения.

Модификаторы доступа — это ключевые слова, с помощью которых осуществляется управление сокрытием данных в классе. Простыми словами, модификаторы доступа задают уровень доступности членам класса, ограничивая область их видимости.

**Разграничение доступ к полям и методам** класса можно регулировать с помощью идентификаторов доступа **private** и **public**. Элементы, описанные после служебного слова **private,** доступнытолько внутри класса. Этот вид доступа принят в классе по умолчанию (т.е. все элементы класса – данные и методы будут скрыты по умолчанию). К закрытым элементам класса можно получить доступ только с помощью методов самого класса. Элементы, описанные после служебного слова **public,** доступныи за пределами класса. Действие любого спецификатора распространяется до следующего спецификатора или до конца класса.

1. Что такое инкапсуляция?

**Инкапсуляция** – это механизм, который объединяет данные и код, манипулирующий с этими данными, а также защищает и то и другое от внешнего вмешательства или неправильного использования. Позволяет скрыть конкретную реализацию класса, облегчая отладку и модификацию программ.

1. Где в программе с классами можно создавать объекты? Внутри функций, в куче (с использованием оператора “new”), в стеке (как часть массива или коллекции), в качестве члена другого класса.
2. Каков смысл объявления объектов?

При определении класса мы не создали объекты. Определение класса лишь задает структуру будущего объекта. Все операции программа производит с объектами. Объявление объекта похоже на определение переменной, оно означает выделение памяти, необходимой для хранения объекта.

1. В какой момент метод готов для вызова объектом? Когда объект этого класса создан и находится в корректном состоянии.
2. Как называются методы, встроенные в структуру класса? Членами или элементами класса.
3. Как определяется метод, если внутри класса записан только его заголовок, сам метод определен вне класса? Это называется "разделением объявления и определения метода". В языке C++ это делается для того, чтобы вынести реализацию метода за пределы заголовка класса. Такой подход может быть полезен для улучшения структурированности кода, уменьшения сложности заголовков и обеспечения более гибкой организации кода.

class MyClass {

public:

// Заголовок метода

void myMethod();

};

// Реализация метода

void MyClass::myMethod() {

std::cout << "Implementation of myMethod" << std::endl;

}

1. Если в классе два поля данных и два объекта, сколько полей принадлежит каждому объекту? Совпадет ли имена и значения этих полей для объектов? Если в классе определены два поля данных, то каждому объекту этого класса будут принадлежать свои собственные два экземпляра этих полей. Каждый объект содержит свой собственный набор данных, который может иметь разные значения. Имена полей могут совпадать, но их значения будут независимыми для каждого объекта.
2. Тиражируются ли методы класса? Методы класса не тиражируются, то есть всегда хранятся в единственном экземпляре.
3. Как в программе написать доступ к открытым и закрытым полям класса?

Доступ к открытым полям объекта класса **напрямую** через операцию (точка):

part1.ModelNumber=2;

part1. ComponentNumber =5;

Чтобы обратиться к полю, надо указать имя объекта класса, которому принадлежит поле, а затем через точку – имя поля.

Доступ к закрытым полям объекта класса **только через методы** объекта:

part1.ShowCost();

Чтобы вызвать метод, надо указать имя объекта класса, для которого будет вызван метод, а затем через точку – имя метода.