**Ministry of education and science of the Kyrgyz Republic**

**Kyrgyz State Technical University named after I.Razzakov**

**Faculty of Information Technologies**

**Department of Software of Computer Systems**

**Major: 710400 «Software Engineering»**

Report

Discipline: «**Object-Oriented Design**»

Software requirements

Done by**:** student of the SE(eng)- 1- 21

Imerov Adilhan

Checked by: Musabaev E.B.

Bishkek 2024

**1. Спецификации требований к ПО:**

Функциональные требования:

1. ПО должно позволять пользователю вводить данные о скоростях грузовиков на различных участках трассы (равнина, горы, пустыня).
2. ПО должно сравнивать скорости грузовиков на каждом участке трассы и определять победителя.
3. ПО должно выводить на экран таблицу результатов ралли-марафона, отображая скорости грузовиков на разных участках и объявляя победителя.

Нефункциональные требования:

1. ПО должно быть написано на языке C++ с применением объектно-ориентированного программирования.
2. Интерфейс должен быть интуитивно понятным и простым для пользователя.
3. ПО должно обеспечивать корректное и точное сравнение скоростей грузовиков.

**2. Проектирование ПО:**

Входной интерфейс:

* Пользователю предлагается ввести данные о скоростях грузовиков на разных участках трассы (равнина, горы, пустыня) через консольный ввод.

Выходной интерфейс:

* После ввода данных, программа отобразит на экране таблицу результатов ралли-марафона с указанием скоростей грузовиков на каждом участке трассы и определением победителя.

**3. Структура программы:**

int FrCreater(const Kamaz &k, const Tatra &t);

class Kamaz {

public:

Kamaz(string n, int sp, int sm, int sd);

void display();

private:

string name;

int speed\_plain;

int speed\_mountains;

int speed\_desert;

friend int FrCreater(const Kamaz &k, const Tatra &t);

friend class Tatra;

};

// Класс для грузовика Tatra

class Tatra {

public:

Tatra(string n, int sp, int sm, int sd);

void display();

private:

string name;

int speed\_plain;

int speed\_mountains;

int speed\_desert;

friend int FrCreater(const Kamaz &k, const Tatra &t);

friend class Kamaz;

};

1. **Что такое дружественная функция, для чего она нужна?**

Дружественная функция - это функция, которой разрешен доступ к закрытым членам класса, к которому она объявлена дружественной. Это делается для обеспечения удобства доступа к данным и методам класса извне без нарушения инкапсуляции.

1. **Как она объявляется, каков её синтаксис?**

Дружественная функция объявляется внутри класса с использованием ключевого слова **friend**. Синтаксис:

cppCopy code

class MyClass { public: friend void friendFunction(); // Объявление дружественной функции };

1. **Через что дружественная функция может получить доступ к закрытому полю класса?**

Дружественная функция получает доступ к закрытым полям класса через объект класса, который передается в качестве аргумента или через объект, на который указывает указатель или ссылка.

1. **Какой вариант вызова ДФ правильный:**

Оба варианта вызова могут быть правильными в зависимости от контекста. В классе, где определена дружественная функция, использование **show()** может быть более естественным, а вне класса - **show(a)**.

1. **a.show() или show(a)?**

Первый вариант - **a.show()** - вызывает метод **show()** объекта **a**. Второй вариант - **show(a)** - предполагает вызов функции **show()** с аргументом **a**.

1. **Наследуется ли дружественная функция?**

Нет, дружественная функция не наследуется. Она имеет доступ только к закрытым членам того класса, который её объявил дружественной.

1. **Может ли дружественная функция быть дружественной более чем к одному классу?**

Да, дружественная функция может быть дружественной более чем к одному классу. Для этого её нужно объявить как дружественную для каждого класса, к которому она должна иметь доступ.

1. **Для решения каких задачах есть смысл использовать функцию, дружественную двум и более классам?**

Дружественные функции могут быть полезны для решения задач, связанных с взаимодействием нескольких классов, когда необходим доступ к закрытым данным нескольких классов без создания лишних методов доступа.

1. **Когда нужно использовать дружественные классы?**

Дружественные классы могут использоваться, когда необходим доступ к закрытым членам нескольких классов, например, для упрощения реализации операторов перегрузки, когда операция требует доступа к закрытым членам двух или более классов.

1. **Какому принципу ООП не соответствуют дружественные функции?**

Дружественные функции могут нарушать принцип инкапсуляции, так как они получают доступ к закрытым членам класса извне, не используя методы доступа.

1. **Как можно ограничить использование ДФ?**

Использование дружественных функций следует минимизировать, ограничивая доступ к закрытым членам класса только в тех случаях, когда это действительно необходимо. Это может быть достигнуто путем создания специальных методов-аксессоров или путем управления доступом через интерфейсы.

1. **В каких крайних случаях нужно использовать ДФ?**

В крайних случаях использование дружественных функций может быть оправдано, когда нет других альтернативных способов обеспечения необходимого взаимодействия между классами, и когда без этого доступа не удастся обеспечить работоспособность программы.