**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. И. РАЗЗАКОВА**

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

**ОТЧЕТ**

**по лабораторной работе №4**

**Выполнила:** студент группы ПИ-3-21

Джанышбекова Акмарал

**Проверила:** Мусабаев Э. Б.

**Бишкек 2024**

**Задания:**

1. Для участия в ежегодном трансконтинентальном Ралли-марафоне «Дакар» по усложненной трассе в Южной Америке выбраны грузовики [КАМАЗ](https://ru.wikipedia.org/wiki/%D0%9A%D0%90%D0%9C%D0%90%D0%97) и [Tatra](https://ru.wikipedia.org/wiki/Tatra), которые после 2000 года лидируют в зачете грузовиков.

Решите задачу сравнения скоростей движения грузовиков по разным по проходимости участкам трассы, а именно: по равнине, горам, пустыне. Создайте и выдайте на экран таблицу результатов ралли - марафона. Определите победителя.

Для решения задачи используйте классы **Kamaz** и **Tatra,** а такжефункциюсравнения скоростей **FrCreater (**). Функция  **FrCreater (**) возвращает число +1, если объект **kamaz** движется быстрее объекта **tatra**; нуль, если их скорости одинаковы; число -1, если объект **kamaz** движется медленнее объекта **tatra** .

Оба класса содержат поля: «скорость» и «наименование» грузовой машины, а также методы: инициализация и отображение полей на экране. Определитесь с идентификаторами доступа к членам класса, не нарушая принципа инкапсуляции.

1. Сделайте функцию **FrCreater(**) дружественной: классу **Kamaz,** обоим классам **Kamaz** и **Tatra.**

Сделайте класс **Tatra** дружественным классу **Kamaz.**

**Код программы:**

#include <iostream>

#include <string>

class Tatra; // Предварительное объявление класса Tatra

class Kamaz {

private:

std::string name;

double speed;

public:

Kamaz(const std::string& n) : name(n), speed(0) {}

// Метод для ввода скорости Kamaz

void inputSpeed() {

std::cout << "Введите скорость Kamaz " << name << " (км/ч): ";

std::cin >> speed;

}

// Метод для получения скорости Kamaz

double getSpeed() const {

return speed;

}

// Метод для отображения информации о Kamaz

void displayInfo() const {

std::cout << "Kamaz " << name << " (Скорость: " << speed << " км/ч)\n";

}

// Функция для сравнения скоростей Kamaz и Tatra

friend int FrCreater(const Kamaz& kamaz, const Tatra& tatra);

};

class Tatra {

private:

std::string name;

double speed;

public:

Tatra(const std::string& n) : name(n), speed(0) {}

// Метод для ввода скорости Tatra

void inputSpeed() {

std::cout << "Введите скорость Tatra " << name << " (км/ч): ";

std::cin >> speed;

}

// Метод для получения скорости Tatra

double getSpeed() const {

return speed;

}

// Метод для отображения информации о Tatra

void displayInfo() const {

std::cout << "Tatra " << name << " (Скорость: " << speed << " км/ч)\n";

}

// Дружественный метод для сравнения скоростей Kamaz и Tatra

friend int FrCreater(const Kamaz& kamaz, const Tatra& tatra);

};

// Функция для сравнения скоростей Kamaz и Tatra

int FrCreater(const Kamaz& kamaz, const Tatra& tatra) {

if (kamaz.getSpeed() > tatra.getSpeed()) {

return 1;

}

else if (kamaz.getSpeed() < tatra.getSpeed()) {

return -1;

}

else {

return 0;

}

}

int main() {

setlocale(LC\_ALL, "ru");

Kamaz kamaz1("Master");

Kamaz kamaz2("Navigator");

Tatra tatra1("Phoenix");

Tatra tatra2("Terra");

int choice;

int kamazWins = 0;

int tatraWins = 0;

do {

std::cout << "Меню:\n";

std::cout << "1. Ввести скорости Kamaz\n";

std::cout << "2. Ввести скорости Tatra\n";

std::cout << "3. Сравнить скорости по равнине\n";

std::cout << "4. Сравнить скорости в горах\n";

std::cout << "5. Сравнить скорости в пустыне\n";

std::cout << "6. Итоговый результат:\n";

std::cout << "Выберите действие (1-6): ";

std::cin >> choice;

switch (choice) {

case 1:

kamaz1.inputSpeed();

kamaz2.inputSpeed();

break;

case 2:

tatra1.inputSpeed();

tatra2.inputSpeed();

break;

case 3:

if (kamaz1.getSpeed() == 0 || tatra1.getSpeed() == 0) {

std::cout << "Сначала введите скорости для Kamaz и Tatra.\n";

}

else {

int result = FrCreater(kamaz1, tatra1);

if (result > 0) {

std::cout << "Kamaz " << kamaz1.getSpeed() << " > Tatra " << tatra1.getSpeed() << " - Kamaz победил\n";

kamazWins++;

}

else if (result < 0) {

std::cout << "Kamaz " << kamaz1.getSpeed() << " < Tatra " << tatra1.getSpeed() << " - Tatra победил\n";

tatraWins++;

}

else {

std::cout << "Kamaz " << kamaz1.getSpeed() << " = Tatra " << tatra1.getSpeed() << " - Ничья\n";

}

}

break;

case 4: // Горы

if (kamaz2.getSpeed() == 0 || tatra2.getSpeed() == 0) {

std::cout << "Сначала введите скорости для Kamaz и Tatra.\n";

}

else {

int result = FrCreater(kamaz2, tatra2);

if (result > 0) {

std::cout << "Kamaz " << kamaz2.getSpeed() << " > Tatra " << tatra2.getSpeed() << " - Kamaz победил в горах\n";

kamazWins++;

}

else if (result < 0) {

std::cout << "Kamaz " << kamaz2.getSpeed() << " < Tatra " << tatra2.getSpeed() << " - Tatra победил в горах\n";

tatraWins++;

}

else {

std::cout << "Kamaz " << kamaz2.getSpeed() << " = Tatra " << tatra2.getSpeed() << " - Ничья в горах\n";

}

}

break;

case 5: // Пустыня

if (kamaz1.getSpeed() == 0 || tatra2.getSpeed() == 0) {

std::cout << "Сначала введите скорости для Kamaz и Tatra.\n";

}

else {

int result = FrCreater(kamaz1, tatra2);

if (result > 0) {

std::cout << "Kamaz " << kamaz1.getSpeed() << " > Tatra " << tatra2.getSpeed() << " - Kamaz победил в пустыне\n";

kamazWins++;

}

else if (result < 0) {

std::cout << "Kamaz " << kamaz1.getSpeed() << " < Tatra " << tatra2.getSpeed() << " - Tatra победил в пустыне\n";

tatraWins++;

}

else {

std::cout << "Kamaz " << kamaz1.getSpeed() << " = Tatra " << tatra2.getSpeed() << " - Ничья в пустыне\n";

}

}

break;

case 6:

std::cout << "Итоговый результат:\n";

std::cout << "Kamaz победил " << kamazWins << " раза.\n";

std::cout << "Tatra победил " << tatraWins << " раза.\n";

if (kamazWins > tatraWins) {

std::cout << "Kamaz является общим победителем!\n";

}

else if (tatraWins > kamazWins) {

std::cout << "Tatra является общим победителем!\n";

}

else {

std::cout << "Ничья! Kamaz и Tatra равны в количестве побед.\n";

}

break;

default:

std::cout << "Неверный выбор!\n";

break;

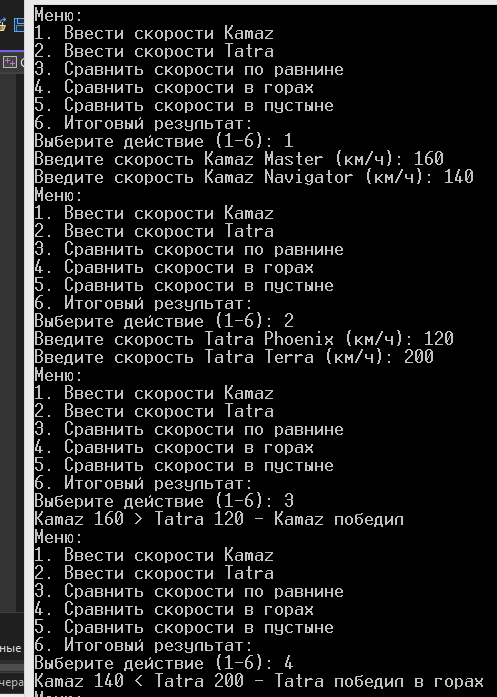
}

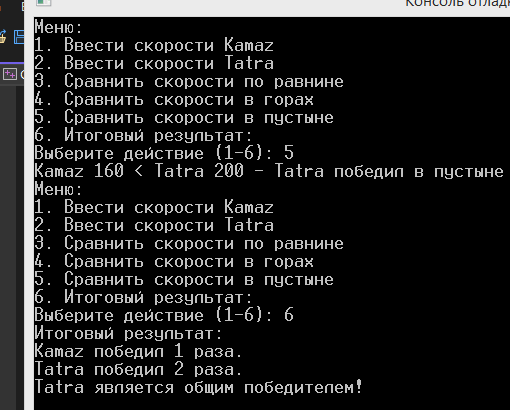
} while (choice != 6);

return 0;

}

**Результат работы программы:**





**ВОПРОСЫ К ЗАЩИТЕ ЛАБОРАТОРНОЙ РАБОТЫ №4**

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

Дружественная функция - это функция, которая имеет доступ к закрытым членам (полям и методам) класса, для которого она объявлена "дружественной". Это нужно, чтобы позволить функции выполнять операции с закрытыми членами класса без нарушения инкапсуляции и обеспечить гибкость в проектировании классов.

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

Дружественная функция объявляется внутри класса, перед ключевым словом friend, за которым следует прототип функции. Синтаксис объявления дружественной функции следующий:

class MyClass {

public:

friend void friendFunction([список\_ параметров]); // Объявление дружественной функции

};

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

Дружественная функция может получить доступ к закрытому полю класса, так как она объявлена как "друг" (friend) класса. Это позволяет ей обращаться к закрытым членам класса напрямую, так как она считается частью класса и имеет специальные привилегии.

1. *Какой вариант вызова ДФ правильный:a.show() или show(a)?*

Правильным вариантом вызова дружественной функции (ДФ) будет `show(a)`. Это означает, что функция `show` вызывается с объектом `a` в качестве аргумента и имеет доступ к его закрытым членам благодаря своему статусу "друга" данного класса.

1. *Наследуется ли дружественная функция?*

Дружественная функция не наследуется. Она имеет доступ только к членам класса, к которому она была объявлена как друг, и не имеет доступа к членам классов, производных от этого класса.

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

Да, дружественная функция может быть объявлена дружественной к нескольким классам. В таком случае она имеет доступ к закрытым членам каждого из этих классов.

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

Смысл использования функции, дружественной двум и более классам, заключается в том, чтобы предоставить этой функции доступ к закрытым членам этих классов, когда это необходимо для выполнения определенных операций или операторов, и при этом не делать эти члены открытыми для всех. Такие функции могут использоваться, например, для сравнения, конвертации данных, или выполнения других операций, которые требуют доступа к закрытым данным нескольких классов.

1. *Когда нужно использовать дружественные классы?*

Дружественные классы используются, когда несколько классов должны иметь доступ к закрытым членам друг друга для выполнения определенных операций или совместной работы, но при этом не следует делать эти члены открытыми для всего кода.

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

Дружественные функции могут нарушать принцип инкапсуляции, так как они предоставляют доступ к закрытым членам класса другого класса без использования методов доступа.

1. *Как можно ограничить использование ДФ?*

Для ограничения использования дружественных функций можно внимательно выбирать, какие функции или классы объявлять дружественными. Также можно ограничивать их использование с помощью модификаторов доступа, таких как `private` или `protected`, чтобы предоставить доступ только определенным частям класса.

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

Дружественные функции следует использовать в крайних случаях, когда другие методы доступа (геттеры и сеттеры) не являются удовлетворительным решением, и есть необходимость предоставить доступ к закрытым членам класса для конкретных операций.