**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И.Раззакова**

**ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

Кафедра: **Программное обеспечение компьютерных систем**

Курс «Объектно-ориентированное программирование»

**ОТЧЕТ**

**Лабораторная работа №4**

Выполнил: студент группы ПИ-3-21

Урманбетов Султан

Проверил: Мусабаев Э.Б.

**Бишкек 2024**

# **Задание №1-3**

**Постановка задачи:**

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.

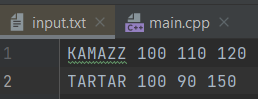
Для решения задачи используйте классы **Kamaz** и **Tatra,** а такжефункциюсравнения скоростей **FrCreater (**). Функция  **FrCreater (**) возвращает число +1, если объект **kamaz** движется быстрее объекта **tatra**; нуль, если их скорости одинаковы; число -1, если объект **kamaz** движется медленнее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

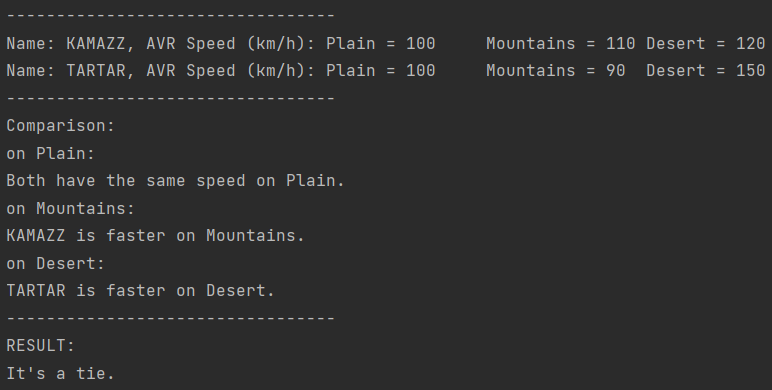
Оба класса содержат поля: «скорость» и «наименование» грузовой машины, а также методы: инициализация и отображение полей на экране. Определитесь с идентификаторами доступа к членам класса, не нарушая принципа инкапсуляции.

**Исходный код программы на C++:**

#include <bits/stdc++.h>  
using namespace std;  
  
class Kamaz {  
private:  
 string name;  
 int speed\_plain;  
 int speed\_mountains;  
 int speed\_desert;  
  
public:  
 Kamaz(string n, int plain, int mountains, int desert) : name(n), speed\_plain(plain), speed\_mountains(mountains), speed\_desert(desert) {}  
  
 int getSpeedPlain() const {  
 return speed\_plain;  
 }  
  
 int getSpeedMountains() const {  
 return speed\_mountains;  
 }  
  
 int getSpeedDesert() const {  
 return speed\_desert;  
 }  
  
 string getName() const {  
 return name;  
 }  
  
 void display() const {  
 cout<<"Name: "<<name<<", AVR Speed (km/h):\tPlain = "<<speed\_plain<<"\tMountains = "<<speed\_mountains<<"\tDesert = "<<speed\_desert<< endl;  
 }  
};  
  
class Tatra {  
private:  
 string name;  
 int speed\_plain;  
 int speed\_mountains;  
 int speed\_desert;  
  
public:  
 Tatra(string n, int plain, int mountains, int desert) : name(n), speed\_plain(plain), speed\_mountains(mountains), speed\_desert(desert) {}  
  
 int getSpeedPlain() const {  
 return speed\_plain;  
 }  
  
 int getSpeedMountains() const {  
 return speed\_mountains;  
 }  
  
 int getSpeedDesert() const {  
 return speed\_desert;  
 }  
  
 string getName() const {  
 return name;  
 }  
  
 void display() const {  
 cout<<"Name: "<<name<<", AVR Speed (km/h):\tPlain = "<<speed\_plain<<"\tMountains = "<<speed\_mountains<<"\tDesert = "<<speed\_desert<< endl;  
 }  
};  
  
int FrCreater(const Kamaz& kamaz, const Tatra& tatra, const string& terrain) {  
 if (terrain == "Plain") {  
 if (kamaz.getSpeedPlain() > tatra.getSpeedPlain())  
 return 1;  
 else if (kamaz.getSpeedPlain() < tatra.getSpeedPlain())  
 return -1;  
 else  
 return 0;  
 } else if (terrain == "Mountains") {  
 if (kamaz.getSpeedMountains() > tatra.getSpeedMountains())  
 return 1;  
 else if (kamaz.getSpeedMountains() < tatra.getSpeedMountains())  
 return -1;  
 else  
 return 0;  
 } else if (terrain == "Desert") {  
 if (kamaz.getSpeedDesert() > tatra.getSpeedDesert())  
 return 1;  
 else if (kamaz.getSpeedDesert() < tatra.getSpeedDesert())  
 return -1;  
 else  
 return 0;  
 } else {  
 cout<<"Invalid terrain!"<<endl;  
 return 0;  
 }  
}  
  
int main() {  
 ifstream inputFile("input.txt");  
 if (!inputFile.is\_open()) {  
 cout<<"Failed to open input file."<<endl;  
 return -1;  
 }  
  
 string kamazName, tatraName;  
 int kamazPlain, kamazMountains, kamazDesert;  
 int tatraPlain, tatraMountains, tatraDesert;  
  
 inputFile>>kamazName>>kamazPlain>>kamazMountains>>kamazDesert>>tatraName>>tatraPlain>>tatraMountains>>tatraDesert;  
  
 Kamaz kamaz(kamazName, kamazPlain, kamazMountains, kamazDesert);  
 Tatra tatra(tatraName, tatraPlain, tatraMountains, tatraDesert);  
  
 cout<<"Results:"<<endl;  
 cout<<"------------------"<<endl;  
 cout<<"Kamaz: ";  
 kamaz.display();  
 cout<<"Tatra: ";  
 tatra.display();  
 cout<<"------------------"<<endl;  
  
 cout<<"Comparisons:"<<endl;  
 cout<<"Plain: ";  
 int comparisonPlain = FrCreater(kamaz, tatra, "Plain");  
 if (comparisonPlain > 0)  
 cout<<"Kamaz is faster."<<endl;  
 else if (comparisonPlain < 0)  
 cout<<"Tatra is faster."<<endl;  
 else  
 cout<<"Both Kamaz and Tatra have the same speed."<<endl;  
  
 cout<<"Mountains: ";  
 int comparisonMountains = FrCreater(kamaz, tatra, "Mountains");  
 if (comparisonMountains > 0)  
 cout<<"Kamaz is faster."<<endl;  
 else if (comparisonMountains < 0)  
 cout<<"Tatra is faster."<<endl;  
 else  
 cout<<"Both Kamaz and Tatra have the same speed."<<endl;  
  
 cout<<"Desert: ";  
 int comparisonDesert = FrCreater(kamaz, tatra, "Desert");  
 if (comparisonDesert > 0)  
 cout<<"Kamaz is faster."<<endl;  
 else if (comparisonDesert < 0)  
 cout<<"Tatra is faster."<<endl;  
 else  
 cout<<"Both Kamaz and Tatra have the same speed."<<endl;  
  
 inputFile.close();  
  
 return 0;  
}

**Тест:**

****

****

# **Ответы на вопросы:**

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

**Дружественная функция**—это функция, не являющаяся членом класса, но обеспечивающая доступ ко всем членам класса, для которого она дружественна. Независима от класса **дружественная функция** находится вне класса. Но она объявляется внутри класса как обычная функция и перед ней указывается ключевое слово **friend**. Вне класса перед описанием дружественной функции это ключевое слово не ставится**.**

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

Синтаксис прототипа дружественной функции

***friend*** тип имя ([список\_ параметров])

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

Так как дружественная функция не является методом класса, для которого она дружественна, нельзя вызвать дружественную функцию обычным способом в точечном формате с помощью команды **a.show()**, а можно только командой **show(a).** Так как, хотя дружественная функция "знает" о закрытом поле класса, для которого она является дружественной, доступ к нему она может получить только через объект этого класса, который объявлен внутри функции и передан ей в качестве аргумента. ДФ вызывается для объекта класса.

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

a.show() или show(a)?

Если функция дружественная, то второй вариант.

**5. Наследуется ли дружественная функция?**

Дружественная функция не наследуется. То есть, если в базовый класс дружественная функция включается, то она не является таковой для производных классов.

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

Дружественная функция может быть дружественной более чем к одному классу.

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

Такая дружественная функция полезна тогда, когда у двух и более разных классов имеется нечто общее, с чем необходимо произвести какие-то действия, например, сравнение или арифметические действия закрытых полей объектов, принадлежащих разным классам.

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

С одной стороны, дружественные функции повышают гибкость языка, но, с другой стороны, они не соответствуют принципу ограничения доступа к данным, в соответствии с которым только методы класса могут иметь доступ к сокрытым данным класса. Множество дружественных функций фактически стирают границы между классами, поэтому они должны быть разбросаны по программе и встречаться как можно реже.

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

Инкапсуляция.

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

Не использовать их, использовать открытые методы класса. Или каждый раз наследуя один класс от дружественного, в таком случае, ДФ не наследуется.

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

Если требуется для базового класса сделать доступ к приватным данным, а для наследуемых нет. В этом случае играет роль то, что дружественные функции не наследуются.