**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ им. И. РАЗЗАКОВА**

**Кафедра «Программное обеспечение компьютерных систем»**

**Отчет**

**ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ С++**

**Лабораторная работа №4**

**Выполнила: Эмилбекова Элиза ПИ-2-21**

**Проверил: 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.

**Код:**

#include <iostream>

#include <string>

using namespace std;

class Kamaz {

private:

string name;

int speed\_plain;

int speed\_mountains;

int speed\_desert;

public:

Kamaz(string n, int sp, int sm, int sd) : name(n), speed\_plain(sp), speed\_mountains(sm), speed\_desert(sd) {}

int getPlainSpeed() const { return speed\_plain; }

int getMountainsSpeed() const { return speed\_mountains; }

int getDesertSpeed() const { return speed\_desert; }

string getName() const { return name; }

};

class Tatra {

private:

string name;

int speed\_plain;

int speed\_mountains;

int speed\_desert;

public:

Tatra(string n, int sp, int sm, int sd) : name(n), speed\_plain(sp), speed\_mountains(sm), speed\_desert(sd) {}

int getPlainSpeed() const { return speed\_plain; }

int getMountainsSpeed() const { return speed\_mountains; }

int getDesertSpeed() const { return speed\_desert; }

string getName() const { return name; }

};

int main() {

string kamazName, tatraName;

int kamazPlain, kamazMountains, kamazDesert;

int tatraPlain, tatraMountains, tatraDesert;

cout << "Enter Kamaz name: ";

cin >> kamazName;

cout << "Enter Kamaz speed on Plain (km/h): ";

cin >> kamazPlain;

cout << "Enter Kamaz speed in Mountains (km/h): ";

cin >> kamazMountains;

cout << "Enter Kamaz speed in Desert (km/h): ";

cin >> kamazDesert;

cout << "Enter Tatra name: ";

cin >> tatraName;

cout << "Enter Tatra speed on Plain (km/h): ";

cin >> tatraPlain;

cout << "Enter Tatra speed in Mountains (km/h): ";

cin >> tatraMountains;

cout << "Enter Tatra speed in Desert (km/h): ";

cin >> tatraDesert;

cout << "=== Results ===" << endl;

Kamaz kamaz(kamazName, kamazPlain, kamazMountains, kamazDesert);

Tatra tatra(tatraName, tatraPlain, tatraMountains, tatraDesert);

int kamazWins = 0, tatraWins = 0;

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

cout << "Plain: " << kamaz.getName() << " wins" << endl;

kamazWins++;

}

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

cout << "Plain: " << tatra.getName() << " wins" << endl;

tatraWins++;

}

else {

cout << "Plain: It's a tie!" << endl;

}

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

cout << "Mountains: " << kamaz.getName() << " wins" << endl;

kamazWins++;

}

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

cout << "Mountains: " << tatra.getName() << " wins" << endl;

tatraWins++;

}

else {

cout << "Mountains: It's a tie!" << endl;

}

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

cout << "Desert: " << kamaz.getName() << " wins" << endl;

kamazWins++;

}

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

cout << "Desert: " << tatra.getName() << " wins" << endl;

tatraWins++;

}

else {

cout << "Desert: It's a tie!" << endl;

}

if (kamazWins > tatraWins) {

cout << kamaz.getName() << " wins overall!" << endl;

}

else if (kamazWins < tatraWins) {

cout << tatra.getName() << " wins overall!" << endl;

}

else {

cout << "Overall: It's a tie!" << endl;

}

return 0;

}

У нас есть 2 класса Kamaz и Tatra. Идет сравнение по 3 критериям и выбирается победитель тот кто набрал больше очков.

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

#include <iostream>

#include <string>

using namespace std;

class Tatra;

class Kamaz {

private:

string name;

int speed\_plain;

int speed\_mountains;

int speed\_desert;

public:

Kamaz(string n, int sp, int sm, int sd) : name(n), speed\_plain(sp), speed\_mountains(sm), speed\_desert(sd) {}

int getPlainSpeed() const { return speed\_plain; }

int getMountainsSpeed() const { return speed\_mountains; }

int getDesertSpeed() const { return speed\_desert; }

string getName() const { return name; }

friend void FrCreater(Kamaz& kamaz);

friend void FrCreater(Kamaz& kamaz, Tatra& tatra);

friend void compareSpeed(Kamaz& kamaz, Tatra& tatra);

};

class Tatra {

private:

string name;

int speed\_plain;

int speed\_mountains;

int speed\_desert;

public:

Tatra(string n, int sp, int sm, int sd) : name(n), speed\_plain(sp), speed\_mountains(sm), speed\_desert(sd) {}

int getPlainSpeed() const { return speed\_plain; }

int getMountainsSpeed() const { return speed\_mountains; }

int getDesertSpeed() const { return speed\_desert; }

string getName() const { return name; }

friend void FrCreater(Kamaz& kamaz, Tatra& tatra);

friend void compareSpeed(Kamaz& kamaz, Tatra& tatra);

};

void FrCreater(Kamaz& kamaz) {

// Функция, обращающаяся к приватным членам класса Kamaz

cout << "Friend function accessing Kamaz private members." << endl;

cout << "Name: " << kamaz.name << endl;

cout << "Plain Speed: " << kamaz.speed\_plain << endl;

cout << "Mountains Speed: " << kamaz.speed\_mountains << endl;

cout << "Desert Speed: " << kamaz.speed\_desert << endl;

}

void FrCreater(Kamaz& kamaz, Tatra& tatra) {

// Функция, обращающаяся к приватным членам классов Kamaz и Tatra

cout << "Friend function accessing Kamaz and Tatra private members." << endl;

cout << "Kamaz Name: " << kamaz.name << endl;

cout << "Kamaz Plain Speed: " << kamaz.speed\_plain << endl;

cout << "Kamaz Mountains Speed: " << kamaz.speed\_mountains << endl;

cout << "Kamaz Desert Speed: " << kamaz.speed\_desert << endl;

cout << "Tatra Name: " << tatra.name << endl;

cout << "Tatra Plain Speed: " << tatra.speed\_plain << endl;

cout << "Tatra Mountains Speed: " << tatra.speed\_mountains << endl;

cout << "Tatra Desert Speed: " << tatra.speed\_desert << endl;

}

void compareSpeed(Kamaz& kamaz, Tatra& tatra) {

int kamazWins = 0, tatraWins = 0;

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

cout << "Plain: " << kamaz.getName() << " wins" << endl;

kamazWins++;

}

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

cout << "Plain: " << tatra.getName() << " wins" << endl;

tatraWins++;

}

else {

cout << "Plain: It's a tie!" << endl;

}

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

cout << "Mountains: " << kamaz.getName() << " wins" << endl;

kamazWins++;

}

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

cout << "Mountains: " << tatra.getName() << " wins" << endl;

tatraWins++;

}

else {

cout << "Mountains: It's a tie!" << endl;

}

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

cout << "Desert: " << kamaz.getName() << " wins" << endl;

kamazWins++;

}

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

cout << "Desert: " << tatra.getName() << " wins" << endl;

tatraWins++;

}

else {

cout << "Desert: It's a tie!" << endl;

}

if (kamazWins > tatraWins) {

cout << kamaz.getName() << " wins overall!" << endl;

}

else if (kamazWins < tatraWins) {

cout << tatra.getName() << " wins overall!" << endl;

}

else {

cout << "Overall: It's a tie!" << endl;

}

}

int main() {

string kamazName, tatraName;

int kamazPlain, kamazMountains, kamazDesert;

int tatraPlain, tatraMountains, tatraDesert;

cout << "Enter Kamaz name: ";

cin >> kamazName;

cout << "Enter Kamaz speed on Plain (km/h): ";

cin >> kamazPlain;

cout << "Enter Kamaz speed in Mountains (km/h): ";

cin >> kamazMountains;

cout << "Enter Kamaz speed in Desert (km/h): ";

cin >> kamazDesert;

cout << "Enter Tatra name: ";

cin >> tatraName;

cout << "Enter Tatra speed on Plain (km/h): ";

cin >> tatraPlain;

cout << "Enter Tatra speed in Mountains (km/h): ";

cin >> tatraMountains;

cout << "Enter Tatra speed in Desert (km/h): ";

cin >> tatraDesert;

Kamaz kamaz(kamazName, kamazPlain, kamazMountains, kamazDesert);

Tatra tatra(tatraName, tatraPlain, tatraMountains, tatraDesert);

FrCreater(kamaz);

FrCreater(kamaz, tatra);

compareSpeed(kamaz, tatra);

return 0;

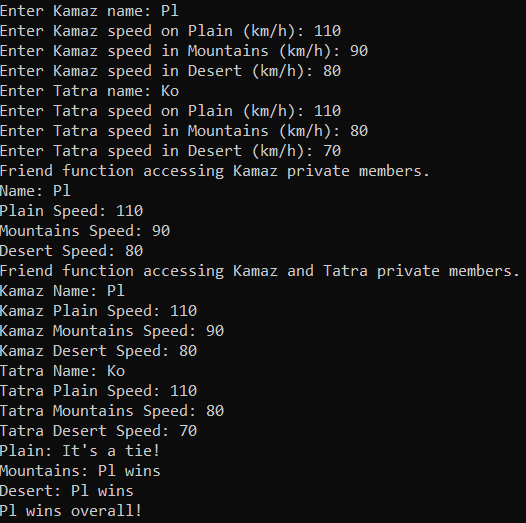
}

Здесь функции FrCreater() и compareSpeed() объявлены как дружественные для классов Kamaz и Tatra. Это означает, что эти функции имеют доступ к приватным членам этих классов и могут использовать их в своем коде.

Когда функция объявляется как дружественная внутри класса, это означает, что она является членом этого класса, но не является его методом. Она все еще определяется и реализуется за пределами класса, но имеет специальный доступ к приватным членам класса.

В нашем случае, FrCreater() и compareSpeed() могут обращаться к приватным переменным классов Kamaz и Tatra для вывода информации и сравнения скоростей, несмотря на то, что они не являются методами этих классов.

**Работа программы:**

****

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

**#include <iostream>**

**#include <string>**

**using namespace std;**

**class Tatra;**

**class Kamaz {**

**private:**

**string name;**

**int speed\_plain;**

**int speed\_mountains;**

**int speed\_desert;**

**public:**

**Kamaz(string n, int sp, int sm, int sd) : name(n), speed\_plain(sp), speed\_mountains(sm), speed\_desert(sd) {}**

**int getPlainSpeed() const { return speed\_plain; }**

**int getMountainsSpeed() const { return speed\_mountains; }**

**int getDesertSpeed() const { return speed\_desert; }**

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

**friend void FrCreater(Kamaz& kamaz);**

**friend void FrCreater(Kamaz& kamaz, Tatra& tatra);**

**friend void compareSpeed(Kamaz& kamaz, Tatra& tatra);**

**friend class Tatra;**

**};**

**class Tatra {**

**private:**

**string name;**

**int speed\_plain;**

**int speed\_mountains;**

**int speed\_desert;**

**public:**

**Tatra(string n, int sp, int sm, int sd) : name(n), speed\_plain(sp), speed\_mountains(sm), speed\_desert(sd) {}**

**int getPlainSpeed() const { return speed\_plain; }**

**int getMountainsSpeed() const { return speed\_mountains; }**

**int getDesertSpeed() const { return speed\_desert; }**

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

**friend void FrCreater(Kamaz& kamaz, Tatra& tatra);**

**friend void compareSpeed(Kamaz& kamaz, Tatra& tatra);**

**};**

**void FrCreater(Kamaz& kamaz) {**

**cout << "Friend function accessing Kamaz private members." << endl;**

**// Здесь можно вывести информацию о приватных членах класса Kamaz**

**}**

**void FrCreater(Kamaz& kamaz, Tatra& tatra) {**

**cout << "Friend function accessing Kamaz and Tatra private members." << endl;**

**// Здесь можно вывести информацию о приватных членах классов Kamaz и Tatra**

**}**

**void compareSpeed(Kamaz& kamaz, Tatra& tatra) {**

**int kamazWins = 0, tatraWins = 0;**

**if (kamaz.getPlainSpeed() > tatra.getPlainSpeed()) {**

**cout << "Plain: " << kamaz.getName() << " wins" << endl;**

**kamazWins++;**

**} else if (kamaz.getPlainSpeed() < tatra.getPlainSpeed()) {**

**cout << "Plain: " << tatra.getName() << " wins" << endl;**

**tatraWins++;**

**} else {**

**cout << "Plain: It's a tie!" << endl;**

**}**

**if (kamaz.getMountainsSpeed() > tatra.getMountainsSpeed()) {**

**cout << "Mountains: " << kamaz.getName() << " wins" << endl;**

**kamazWins++;**

**} else if (kamaz.getMountainsSpeed() < tatra.getMountainsSpeed()) {**

**cout << "Mountains: " << tatra.getName() << " wins" << endl;**

**tatraWins++;**

**} else {**

**cout << "Mountains: It's a tie!" << endl;**

**}**

**if (kamaz.getDesertSpeed() > tatra.getDesertSpeed()) {**

**cout << "Desert: " << kamaz.getName() << " wins" << endl;**

**kamazWins++;**

**} else if (kamaz.getDesertSpeed() < tatra.getDesertSpeed()) {**

**cout << "Desert: " << tatra.getName() << " wins" << endl;**

**tatraWins++;**

**} else {**

**cout << "Desert: It's a tie!" << endl;**

**}**

**if (kamazWins > tatraWins) {**

**cout << kamaz.getName() << " wins overall!" << endl;**

**} else if (kamazWins < tatraWins) {**

**cout << tatra.getName() << " wins overall!" << endl;**

**} else {**

**cout << "Overall: It's a tie!" << endl;**

**}**

**}**

**int main() {**

**string kamazName, tatraName;**

**int kamazPlain, kamazMountains, kamazDesert;**

**int tatraPlain, tatraMountains, tatraDesert;**

**cout << "Enter Kamaz name: ";**

**cin >> kamazName;**

**cout << "Enter Kamaz speed on Plain (km/h): ";**

**cin >> kamazPlain;**

**cout << "Enter Kamaz speed in Mountains (km/h): ";**

**cin >> kamazMountains;**

**cout << "Enter Kamaz speed in Desert (km/h): ";**

**cin >> kamazDesert;**

**cout << "Enter Tatra name: ";**

**cin >> tatraName;**

**cout << "Enter Tatra speed on Plain (km/h): ";**

**cin >> tatraPlain;**

**cout << "Enter Tatra speed in Mountains (km/h): ";**

**cin >> tatraMountains;**

**cout << "Enter Tatra speed in Desert (km/h): ";**

**cin >> tatraDesert;**

**Kamaz kamaz(kamazName, kamazPlain, kamazMountains, kamazDesert);**

**Tatra tatra(tatraName, tatraPlain, tatraMountains, tatraDesert);**

**compareSpeed(kamaz, tatra);**

**return 0;**

**}**

В этом коде класс Tatra объявлен с ключевым словом friend внутри класса Kamaz, что делает его другом класса Kamaz. Это позволяет классу Tatra иметь доступ к приватным членам класса Kamaz.

Теперь функции FrCreater() и compareSpeed() могут обращаться к приватным членам обоих классов Kamaz и Tatra, так как они оба объявлены как друзья в классе Kamaz.

**ВОПРОСЫ К ЗАЩИТЕ ЛАБОРАТОРНОЙ РАБОТЫ №4**

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

Дружественная функция — это функция, объявленная вне тела класса, но имеющая доступ ко всем его членам, включая закрытые. Она может быть использована для предоставления внешним функциям доступа к закрытым членам класса.

2. Как она объявляется, каков её синтаксис?

- Дружественная функция объявляется внутри тела класса с использованием ключевого слова `friend` перед её объявлением.

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

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

- Дружественная функция получает доступ к закрытым полям класса через использование ключевого слова `friend`. Она рассматривается как "друг" класса и, следовательно, имеет доступ к его закрытым членам.

4. Какой вариант вызова ДФ правильный:

5. a.show() или show(a)?

- Правильный вариант вызова дружественной функции – `show(a)`.

6. Наследуется ли дружественная функция?

- Нет, дружественная функция не наследуется. Она имеет доступ только к членам класса, для которого была объявлена как друг.

7. 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 с использованием ключевого слова `friend` внутри каждого класса.

8. Для решения каких задач есть смысл использовать функцию, дружественную двум и более классам?

- Дружественные функции могут быть полезны, если есть необходимость взаимодействия между несколькими классами, при этом необходим доступ к закрытым членам каждого из них.

9. Когда нужно использовать дружественные классы?

- Дружественные классы могут использоваться, когда один класс должен иметь доступ к закрытым членам другого класса, а также когда требуется обмен информацией между классами.

10. Какому принципу ООП не соответствуют дружественные функции?

- Дружественные функции могут нарушать принцип инкапсуляции, так как предоставляют внешнему коду доступ к закрытым членам класса.

11. Как можно ограничить использование ДФ?

- Используйте ключевое слово `friend` только для тех функций, которым действительно необходим доступ к закрытым членам класса. Минимизируйте количество дружественных функций, чтобы уменьшить потенциальные нарушения инкапсуляции.

12. В каких крайних случаях нужно использовать ДФ?

- Дружественные функции следует использовать осторожно и, как правило, в крайних случаях, когда другие методы взаимодействия между классами оказываются недостаточными. Они могут быть полезны, например, при реализации сложных операций, которые требуют доступа к закрытым членам нескольких классов.