Приятелски класове и функции

Нека разгледаме следните два класа:

```
class SteeringWheel
{
    double rotDeg; //Градус на завиване
    int blinkers; //Мигачи
};
```

```
class Car
{
    int maxHorsePower;
    int year;
    SteeringWheel* wheel;

    public:
        Car(); //Конструктор по подразбиране
        Car(SteeringWheel*); //Кола без волан не е хубаво да имаме
        void driveTheCar();
};
```

Ще се фокусираме върху функцията `driveTheCar()`

```
void Car::driveTheCar()
{
    char pos;
    std::cout << "Where to go ?";
    std::cin >> pos;

    if(pos == 'l') //Ще завиваме наляво
    else if(pos == 'r') //Ще завиваме надясно
}
```

Как ще променим член-данната `rotDeg` на `SteeringWheel` ?

- Ще направим get-ъри и set-ъри ?
- Ще направим член-данните на `SteeringWheel` да са `public` ?

И при двата случая ще променим член-данните на `SteeringWheel`. Но какво ще се случи, ако използваме `SteeringWheel` извън класа `Car` ? (Това е възможно, заради факта, че имаме `public` достъп или до методите или до член-данните и нищо не пречи да кажем `SteeringWheel.rotDeg = 5`, докато някои друг кара колата)

Как можем да решим този проблем ?

Приятелски функции

Можем да дадем достъп до `private` часта на един клас, чрез т.нар. приятелски функции. Това става чрез поставяне на запазената дума `friend` пред функцията, която искаме да има достъп до другия клас.

Нека създадем `private` метод `setRot(double)` в класът `SteeringWheel`

```
class SteeringWheel
{
    private:

    double rotDeg; //Градус на завиване
    int blinkers; //Мигачи
    friend void Car::driveTheCar();
    void setRot(double a)
    {
        rotDeg = a;
    }
};
```

И нека допълним `driveTheCar()`

```
void Car::driveTheCar()
{
        char pos;
        std::cout << "Where to go ?";
        std::cin >> pos;

        if(pos == 'l')
        {
            //Ще завиваме наляво
            wheel->setRot(5);
        }
        else if(pos == 'r')
        {
            //Ще завиваме надясно
        }
}
```

Вече имаме достъп до `setRot()` , въпреки факта, че тя е `private`.

Това стана поради факта, че в класът `SteeringWheel` добавихме функцията от класът `Car`, `driveTheCar()` като приятелска на класа `SteeringWheel` и тя има достъп до `private` данните и методи на класа. **Ако опитаме да достъпим `setRot()` от което и да е друго място, няма да успеем.

Приятелски класове

В предишният пример дадохме достъп на един метод на `Car` до член-данните на `SteeringWheel`. Възниква въпросът - може ли да имаме достъп до `SteeringWheel` от целият клас `Car` ?

Това се осъществява чрез т.нар. **приятелски класове**

Вместо да пишем `friend` за някои метод, можем да направим следното

```
#pragma once
#include "Car.h"

class SteeringWheel
{
    private:

    double rotDeg; //Градус на завиване
    int blinkers; //Мигачи
    //friend void Car::driveTheCar();
    void setRot(double a)
    {
        rotDeg = a;
    }

    friend class Car;
};
```

И отново ще имаме достъп до `setRot()` в метода `driveTheCar()`. Разликата е, че ще имаме достъп до `setRot()` и другите член-данни навсякъде в класа `Car`

Задачи:

Задача 1:

Да се реализира функция `sort(MyVector&)`, която да сортира елементите на класа `MyVector`, но е извън класа.

MyVector.h

```
#pragma once

class MyVector
{
    private:
        int* values;
        int size;
        int maxSize;

        void resize();
    public:
        MyVector(); //Default
        MyVector(const MyVector&); //Copy
        MyVector& operator=(const MyVector&); //Operator=

        void set_element(int, int); //pos, val
        void push_back(int);
        void insert(int, int); //pos, val
        void remove(int); //pos
        void print() const;
```

```
        int get_element(int) const;
        int pop_back();
        int begin() const;
        int end() const;
        int get_size() const;

        friend void sort(MyVector&);
        ~MyVector();
};
```

main.cpp

```
#include <iostream>
#include "MyVector.h"

void sort(MyVector& arr)
{
    //Bubble sort implementation
    int size = arr.size;

    for(int i = 0; i < size-1; i++)
    {
        for(int j = i+1; j < size; j++)
        {
            if(arr.values[i] > arr.values[j])
            {
                int swap = arr.values[i];
                arr.values[i] = arr.values[j];
                arr.values[j] = swap;
            }
        }
    }
}

int main() {

    MyVector a;

    a.push_back(3);
    a.push_back(8);
    a.push_back(7);
    a.push_back(5);
    a.push_back(2);

    a.print();

    sort(a);

    a.print();
}
```


lyubo@lyubo-Lenovo-Z710: /mnt/C044EDEE44EDE6DE/OOP_2019/WorkDir/bin
File Edit View Search Terminal Help
lyubo@lyubo-Lenovo-Z710:/mnt/C044EDEE44EDE6DE/OOP_2019/WorkDir/bin$ ./main
3 8 7 5 2
2 3 5 7 8
lyubo@lyubo-Lenovo-Z710:/mnt/C044EDEE44EDE6DE/OOP_2019/WorkDir/bin$