Become a wizard!

PART 1

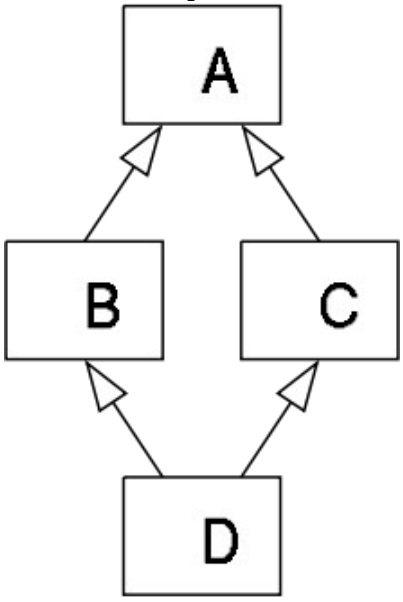
You are going to implement 5 classes of different spells. Every spell will implement a cast() method – so the user of the spell can cast the spell :-). We are interested in your knowledge of polymorphism, inheritance, base classes, subclasses, virtual methods and overrides. The rest is reduced to absolute bare minimum. Your cast() methods should just print simple messages that the spell was cast().

This time you are given only the spell.hpp and the main.cpp file showing the expected usage of 5 classes of spells:

- class Spell – it is a generic type, notice that in main.cpp there are no objects of this class, only of its subclasses.
- class GreenSpell – when used by a skilled wizard it changes the color of the target into green. In your program, for simplicity, it means that the cast() method of this class should print: „Green!”.
- class FireballSpell – when used by an experienced wizard it throws fire balls at enemies. For simplicity your program should print „Fireball!” when the cast method of this class is called.
- class GreenFireballSpell – it is a combination of the previous 2 spells. It may be used only by most experienced wizards. It’s cast() method **must use and call** cast() methods of GreenSpell and FireballSpell.
- class DoubleSpell – it is used by smart and lazy wizards. It may be created as a combination of any of the previous spells. Double spells just take 2 other spells and cast them one after another. If, for example, you create 2 green spells and then create a double spell with those 2 green spells, then the cast() method of the double spell should cast the first spell and then the second. Spells of the class DoubleSpell **are responsible for deleting** spells from which they were created.

Your task:
Design and implement a hierarchical structure of the 5 classes. All the classes implement the cast() method. Only DoubleSpell has member fields to store pointers to the other 2 spells.

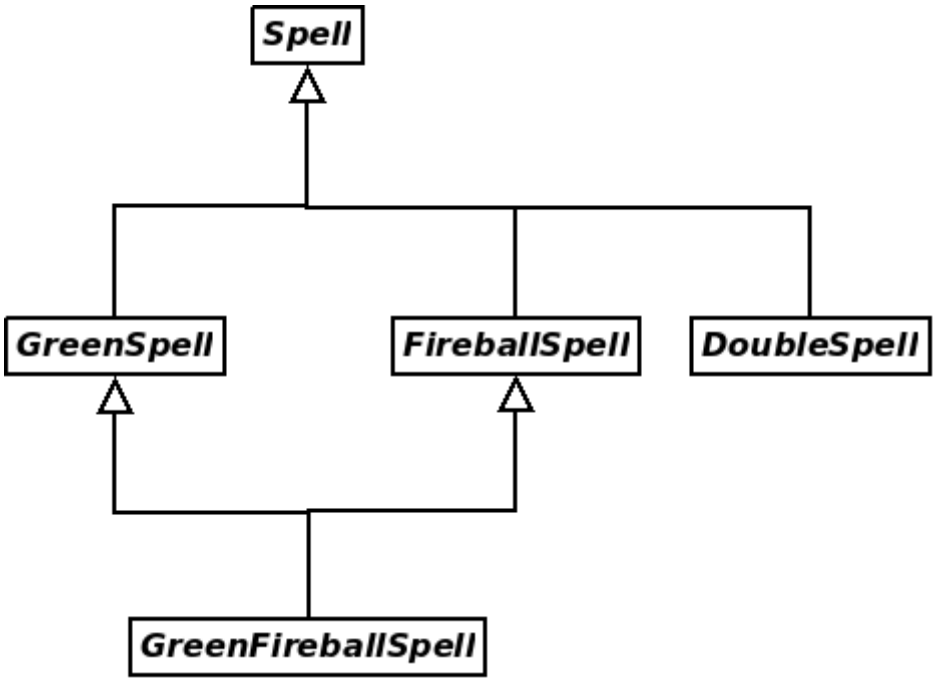
You will face a Deadly Diamond of Death problem.



Here, classes B and C inherit from class A, and class D inherits from both B and C. In such a case we have to tell the compiler to create only one object A and not two in the inheritance path for objects of class D. This is done by adding a **virtual** keyword like this:

```
class A {};
class B : virtual public A {};
class C : virtual public A {};
class D : public B, public C {};
```

The hierarchy between classes **must be**:



Scoring:
1. Proper classes structure: 2 points.
2. Correct implementation of the destructor(s) – where necessary: 0.5 point.
3. Correct implementation of abstract class(es) and pure virtual method(s): 0.5 point.

You cannot modify the part 1 of the main.cpp file. The result of the first part of the program should be:

```
Fireball!
Green!
Fireball!
Green!
Green!
Green!
Fireball!
```

PART 2

Implement a SpellBook class. A spell book contains some number of spells. The contents of the book are defined by the string with names of the spells passed into the constructor. The input string contains only lower-case names: **greenfireball**, **fireball**, and **green** separated by commas (there is

no double spell in this part!). The constructor should allocate a dynamic array of pointers to newly created spells (objects of classes created in part 1) based on the input string. Assume that the input string is always correct.

The spell book should also implement:

1. greenCount() – this method counts and returns the number of all the green and greenfireball spells currently in the book. You **must implement it using run time type information**: https://pl.wikipedia.org/wiki/RTTI.

2. castSpell(int index) – calls the cast() method of the spell stored internally in the book at specified index. If index is incorrect – does nothing.

3. destructor.

Hint 1: Use provided spell_book.hpp header file.
Hint 2: You can use 2 passes over the input string. In the first one count commas to know how many spells are in the input. Then allocate memory to store the spells. Then iterate over the input string again to detect which spells to create and insert into the allocated array.

The results of the second part of the program should be:
Number of green spells in yellow book: 2
Fireball!

Scoring:
1. Proper constructor: 2 points.
2. Properly implemented greenCount(), castSpell(), and destructor: 2 point.

PART 3

1. Extend the implementation of the castSpell() method to throw an exception when when the index is out of bounds.
2. Modify main.cpp file to catch the exceptions that can be thrown by calls to castSpell() method.

Hint: you can implement your own exception or use a standard std::out_of_range.

Scoring:
1. Proper castSpell() modification: 0.5 point.
2. Proper try – cach modyfication in main.cpp: 0.5 point.