



C++ Pool - Exam 01

Looking For Kreog

Maxime "zaz" Montinet zaz@42.fr

Abstract: This document is the subject of the second exam in 42's C++ pool. It is freely inspired by the "Looking For Group" webcomic by R.Sohmer and L.DeSouza. If you don't know what it is, shame on you, go google it after the exam.

Contents

I	Administrative details	2
I.1	General guidelines	2
I.2	Code	3
II	Foreword	5
III	Exercise 00: This day is fantastic !	6
IV	Exercise 01: You like what I do ?	9
V	Exercise 02: Don't fwoosh me, bro !	12
VI	Exercise 03: I. LIKE. TO. KILL. THINGS. How's that not clear by now ?	16

Chapter I


Administrative details

I.1 General guidelines

- No communication **whatsoever** is allowed.
- This is an exam, you don't have a right to chat, listen to music, make noise, or generally do anything that may disturb the other students in any way.
- Your phones and other technological devices must be turned off and put away. If a phone rings, the whole row will be disqualified from the exam and kicked out immediately.
- Your home directory contains two folders: "**rendu**" and "**sujet**".
- The "**sujet**" folder contains the subject of the exam. You have visibly found it.
- The "**rendu**" folder is a clone of your Git turn-in repository. You will work in it, and use it as any regular Git repository.
- We will only grade what you pushed on your Git repository. The system will stop accepting pushes at the exact end time of the exam, so do NOT wait until the last minute to push your work.
- You can only run your programs in the "**rendu**" directory or one of its subdirectories.
- Each exercise must be carried out in the appropriate directory, specified in each exercise's header.
- You must turn in, at the root of your repository, a file named "**auteur**" that contains your login followed by a newline. If this file is absent or wrong in any way you will NOT be graded.

Example:

```
$> cat -e ~/rendu/auteur
xlogin$
```

- 
- You may need to read the man to carry out some exercises...
 - You will be graded by a program. You must respect the specified file/path/function names to the letter.
 - Exercises will always specify which files will be collected :
 - When an exercise asks for specific files, they will be explicitly named. For example "`file1.cpp file1.hpp`".
 - Otherwise, when filenames and/or the number of files is up to you, the exercise will say something along the lines of "`*.cpp *.hpp`".
 - When a Makefile is required, it will ALWAYS be explicitly stated.
 - In case of technical problem, question about the subject, or any other problem, you must get up silently and wait for a member of the staff to come to you. It is forbidden to ask your neighbors, or to verbally call for a staff member.
 - Any equipment not explicitly allowed is implicitly forbidden.
 - Any exit is definitive, you can not come in again.
 - Staff members may kick you out of the exam without warning if they deem it necessary.
 - You are allowed blank pieces of paper, and a pen. No notebooks, notes, or any help of the sort. You are alone to face this exam.
 - All the exercises are mandatory, grading will stop at the first wrong one.
 - Read each exercise FULLY before starting it ! Really, do it.
 - By Odin, by Thor ! Use your brain !!!
 - For any question after the exam, please open a ticket

I.2 Code

- The correction is fully automated, and performed by the program we know as the **Moulinette**.
- In this exam, you will never turn in a **main** function, because we will use our own to test your code, and if you have a **main** function in your own code, it will not compile, and you will get a 0.

- Any function implemented in a header (except in the case of templates), and any unprotected header, means 0 to the exercise.
- Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.
- The imposed filenames must be followed to the letter, as well as class names, function names and method names.
- Remember: You are coding in C++ now, not in C anymore. Therefore, the following functions are FORBIDDEN, and their use will be punished by a -42, no questions asked: `*alloc`, `*printf` and `free`.
- Also note that unless otherwise stated, the C++ keywords "using namespace" and "friend" are forbidden. Their use will be punished by a -42, no questions asked.
- Since you are allowed to use the C++ tools you learned about since the beginning of the pool, you are not allowed to use any external library. And before you ask, that also means no C++11 and derivatives, nor Boost or anything your awesomely skilled friend told you C++ can't exist without.
- Each of your includes must be able to be included independently from others. Includes must contain every other includes they are depending on, obviously.
- When an output example makes a newline, it must not be respected if it is preceded by a backslash ("\"). Same goes for a string between double quotes, obviously. For example, the string "For \ pony !" should be interpreted as "For pony !".

Chapter II

Foreword

Richard is a Warlock. Not exactly sure what that actually MEANS, but that sounds badass enough, doesn't it ?

Whatever it does man, we are certain of a few things : He likes to kill things, he has a fantastic day when he can slaughter an entire family in front of their loved ones, he likes ponies, and he's the mayor of a little village on the coast.


Since we'd like to understand him a little better, and you're a kind and helpful sort of person, you will create an authentic, but domesticated, Warlock for us, so that we can study it until we're bored of it. And maybe, just maybe, it will be useful and rid us of the vermin that's crawling around here.

Get to work !



Chapter III

Exercise 00: This day is fantastic !

	Exercise 00
This day is fantastic !	
Turn-in directory : <i>ex00/</i>	
Files to turn in : Warlock. [hpp,cpp]	
Forbidden functions : None	
Remarks : n/a	

Make a **Warlock** class. It has to be in Coplien's form.

It has the following private attributes :

- **name** (string)
- **title** (string)

Since they're private, you will write the following getters :

- **getName**, returns a reference to constant string
- **getTitle**, returns a reference to constant string

Both these functions will have to be callable on a constant **Warlock**.

Your **Warlock** would not appreciate us being able to change its name, so we'll settle for being able to change its title once in a while. After all, variety is good. So you will write this setter:

- **setTitle**, returns void and takes a reference to constant string

Your **Warlock** will also have, in addition to whatever's required by Coplien's form, a constructor that takes, in this order, its name and title. Your **Warlock** will not be able to be copied, instantiated by copy, or instantiated without a name and a title.

For example :

```
Warlock bob; //Does not compile
Warlock bob("Bob", "the magnificent"); //Compiles
Warlock jim("Jim", "the nauseating"); //Compiles
bob = jim; //Does not compile
Warlock jack(jim); //Does not compile
```

Upon creation, the Warlock says :

<NAME>: This looks like another boring day.

Of course, whenever we use placeholders like <NAME>, <TITLE>, <QUOTE1>, etc... in outputs, you will replace them by the appropriate value. Without the < and >. If you do not, we will have you sit somewhere in the Bocal and laugh at you for about an hour or two. No, this is not a joke.

When he dies, he says:

<NAME>: Ahhh, I see it clearly. This is the plane of SUCK...

Our Warlock must also be able to introduce himself, while boasting with all its might. Pretty normal, mind you, he has a huge... er, ego.

So you will write the following function:

- void introduce() const;

It must display:

<NAME>: I am <NAME>, <TITLE> !

Here's an example of a test main function and its associated output:

```
int main()
{
    Warlock const richard("Richard", "Mistress of Magma");
    richard.introduce();
    std::cout << richard.getName() << " - " << richard.getTitle() << std::endl;

    Warlock* jack = new Warlock("Jack", "the Long");
    jack->introduce();
    jack->setTitle("the Not-So-Long-After-All");
    jack->introduce();


    delete jack;

    return (0);
}
```

```
zaz@naqadah:~$ ./a.out | cat -e
Richard: This looks like another boring day.$
Richard: I am Richard, Mistress of Magma !$
Richard - Mistress of Magma$
Jack: This looks like another boring day.$
Jack: I am Jack, the Long !$
Jack: I am Jack, the Not-So-Long-After-All !$
Jack: Ahhh, I see it clearly. This is the plane of SUCK...$
Richard: Ahhh, I see it clearly. This is the plane of SUCK...$
zaz@naqadah:~$
```

Chapter IV

Exercise 01: You like what I do ?

	Exercise 01
You like what I do ?	
Turn-in directory : <i>ex01/</i>	
Files to turn in : Warlock.[hpp,cpp]	
Forbidden functions : None	
Remarks : n/a	

Someone just tells me that our **Warlock** isn't complete yet.

Indeed, the original Richard has several titles, not just one. He's also a lot more chatty than how our **Warlock** class models it.

Modify the **Warlock** class in the following ways :

- The constructor that took a name and a title now only takes a name.
- The **title** attribute is now a list of strings, and is now called **titles**.
- The **getTitle** and **setTitle** functions disappear.
- The **introduce** function must now display all the **Warlock**'s titles, and an additional one, in the following way:

<NAME>: I am <NAME>, <TITLE1>, <TITLE2> [...], <TITLEn> ! And the mayor of a little v
on the coast. Very scenic during springtime, you should visit sometime !

For example, if our **Warlock** named Richard has these titles :

- Lord of the Undead Ponies
- Master of the Wine Cellar

You must display :

Richard: I am Richard, Lord of the Undead Ponies, Master of the Wine \ Cellar ! And the mayor of a little village on the coast. Very scenic during \ springtime, you should visit sometime !

If your Warlock has no title, you will only mention his name and the fact that he has a little village on the coast (VERY important ...) :

Richard: I am Richard ! And the mayor of a little village on the coast. \ Very scenic during springtime, you should visit sometime !

Since at some point it will be necessary to manipulate its various titles, and we're not about to have him do it by himself (He's got other things to do, he's a busy little warlock), you will make functions to handle this :

- `addTitle`, that takes a reference to constant string and adds a new title to the Warlock's collection.
- `removeTitle`, that takes a reference to constant string and removes a title. If there's no such title, does nothing.

Better ! Now that our Warlock is capable of unveiling his long... list of titles, we'll have to make him able to speak. Because a Warlock that can't say a few well-chosen dark jokes in the face of other people's misery is about as useful as a web developer.

So you will add a private attribute called `quotes`, which will be a list (or a vector, whatever you find best) of strings. One choice is obviously, better than the other.

In very much the same way as with titles, you will create the `addQuote` and `removeQuote` functions.

When it's done, you will write a `talk` function, that takes nothing, returns nothing and is callable on a `const Warlock`.

This function will select a random quote from those our Warlock knows, and output :

<NAME>: <QUOTE>

In the (hopefully very rare) case where your Warlock has nothing to say, this function will do nothing.



To select a random quote, simply use `rand() % number_of_known_quotes`. Do NOT, EVER, use `srand()` in your code, or else the Moulinette will not have the same random selection for its own program and for yours, and you would get a 0. That would be bad.

Here's an example with its associated output. Of course, the order of the quotes may differ, since it is, you know, random.

```
int main()
{
    Warlock richard("Richard");

    richard.addTitle("Chief Warlock of the Brothers of Darkness");
    richard.addTitle("Lord of the Thirteen Hells");
    richard.addTitle("Emperor of the Black");
    richard.addTitle("Lord of the Undead");
    richard.addTitle("Mistress of Magma");
    richard.introduce();

    richard.removeTitle("Mistress of Magma");
    richard.introduce();

    richard.addQuote("You've just been Dick-rolled !");
    richard.addQuote("This day is fantastic ...");
    richard.addQuote("That orphanage attacked ME. It was self-defense !");
    richard.addQuote("You like what I do ?");


    richard.talk();
    richard.talk();
}
```

```
zaz@naqadah:~$ ./a.out | cat -e
Richard: This looks like another boring day.$
Richard: I am Richard, Chief Warlock of the Brothers of Darkness, Lord of the
Thirteen Hells, Emperor of the Black, Lord of the Undead, Mistress of Magma !
And the mayor of a little village on the coast. Very scenic during springtime,
you should visit sometime !$
Richard: I am Richard, Chief Warlock of the Brothers of Darkness, Lord of the
Thirteen Hells, Emperor of the Black, Lord of the Undead ! And the mayor of a
little village on the coast. Very scenic during springtime, you should visit
sometime !$
Richard: This day is fantastic ...$
Richard: You've just been Dick-rolled !$
Richard: Ahhh, I see it clearly. This is the plane of SUCK...$
zaz@naqadah:~$
```



Chapter V

Exercise 02: Don't fwoosh me, bro !

	Exercise 02
Don't fwoosh me, bro !	
Turn-in directory : <i>ex02/</i>	
Files to turn in : <i>Warlock.[hpp,cpp]</i> <i>ASpell.[hpp,cpp]</i> <i>ATarget.[hpp,cpp]</i> <i>Fwoosh.[hpp,cpp]</i> <i>LittleKid.[hpp,cpp]</i>	
Forbidden functions : None	
Remarks : n/a	



In the `Warlock` class, the `switch` statement is FORBIDDEN and its use would result in a -42.

Now we can have some fun. Or, you know, that's what our `Warlock` would say if he knew what we had in store for him.

Now that he can speak and boast to his heart's content, we'll have to keep him busy. If he can't kill people, he gets bored. Poor him.

Create an abstract class called `ASpell`, in Coplien's form, that has the following protected attributes:

- `name` (string)
- `effects` (string)

Both will have getters (`getName` and `getEffects`) that return strings.

Also add a `clone` pure method that returns a pointer to `ASpell`.

All these functions can be called on a constant object.

ASpell has a constructor that takes its name and its effects, in that order.

Now you will create an **ATarget** abstract class, in Coplien's form. It has a **type** attribute, which is a string, and its associated getter, **getType**, that return a reference to constant string.

In much the same way as **ASpell**, it has a **clone()** pure method.

All these functions can be called on a constant object.

It has a constructor that takes its type.

Now, add to your **ATarget** a **getHitBySpell** function that takes a reference to constant **ASpell**.

It will display :

```
<TYPE> has been <EFFECTS> !
```

<TYPE> is the **ATarget**'s type, and <EFFECTS> is the return of the **ASpell**'s **getEffects** function.

Finally, add to your **ASpell** class a **launch** function that takes a reference to constant **ATarget**.

This one will simply call the **getHitBySpell** of the passed object, passing the current instance as parameter.

When all this is done, create an implementation of **ASpell** called **Fwoosh**. Its default constructor will set the name to "Fwoosh" and the effects to "fwooshed". You will, of course, implement the **clone()** method. In the case of **Fwoosh**, it will return a pointer to a new **Fwoosh** object.

In the same way, create a concrete **ATarget** called **LittleKid**, the type of which is "Little Kid". You must also implement its **clone()** method.

Now that we have spells and targets, let's make the **Warlock** able to use them, or else it would still be less useful than a console player.

Add to the **Warlock** the following member functions:

- **learnSpell**, takes a pointer to **ASpell**, that makes the **Warlock** learn a spell
- **forgetSpell**, takes a string corresponding a to a spell's name, and makes the **Warlock** forget it. If it's not a known spell, does nothing.

- `launchSpell`, takes a string (a spell name) and a reference to `ATarget`, that launches the spell on the selected target. If it's not a known spell, does nothing.

You will need a new attribute to store the spells your `Warlock` knows. Several types fit the bill, it's up to you to choose the best one.

Below is a possible test main and its expected output:

```
int main()
{
    Warlock richard("Richard");

    richard.addTitle("Chief Warlock of the Brothers of Darkness");
    richard.addTitle("Lord of the Thirteen Hells");
    richard.addTitle("Emperor of the Black");
    richard.addTitle("Lord of the Undead");
    richard.addTitle("Mistress of Magma");
    richard.addQuote("You've just been Dick-rolled !");
    richard.addQuote("This day is fantastic ...");
    richard.addQuote("That orphanage attacked ME. It was self-defense !");
    richard.addQuote("You like what I do ?");

    LittleKid bob;
    Fwoosh* fwoosh = new Fwoosh();

    richard.learnSpell(fwoosh);

    richard.introduce();
    richard.launchSpell("Fwoosh", bob);
    richard.talk();


    richard.forgetSpell("Fwoosh");
    richard.launchSpell("Fwoosh", bob);
}
```

```
zaz@naqadah:~$ ./a.out | cat -e
Richard: This looks like another boring day.$
Richard: I am Richard, Chief Warlock of the Brothers of Darkness, Lord of the
Thirteen Hells, Emperor of the Black, Lord of the Undead, Mistress of Magma !
And the mayor of a little village on the coast. Very scenic during springtime
, you should visit sometime !$
Little Kid has been fwooshed !$
Richard: That orphanage attacked ME. It was self-defense !$
Richard: Ahhh, I see it clearly. This is the plane of SUCK...$
```



Chapter VI

Exercise 03: I. LIKE. TO. KILL. THINGS. How's that not clear by now ?

	Exercise 03
I. LIKE. TO. KILL. THINGS. How's that not clear by now ?	
Turn-in directory : <i>ex03/</i>	
Files to turn in : Warlock.[hpp,cpp] ASpell.[hpp,cpp] ATarget.[hpp,cpp] Fwoosh.[hpp,cpp] LittleKid.[hpp,cpp] Sprotch.[hpp,cpp] Ponymorph.[hpp,cpp] InnocentWoman.[hpp,cpp] SpellBook.[hpp,cpp] TargetGenerator.[hpp,cpp]	
Forbidden functions : None	
Remarks : n/a	



In the Warlock, SpellBook and TargetGenerator classes, the `switch` statement is FORBIDDEN and its use would result in a -42.

Our Warlock, after an afternoon spent burning small children alive (According to him, they "smell like chocolate". Creeeeepy...), is already starting to get bored. Well, duh. He only has ONE spell. That would be kind of like having to code only in C# for the rest of your mortal life.

So, you'll help him finally be happy. Create the following two spells, on the same model as Fwoosh:

- Sprotch (Name: "Sprotch", Effects: "sprotched")
- Ponymorph (Name: "Ponymorph", Effects: "turned into a nice, cute little pony")

In addition to this, just so he won't have only kids to slaughter, let's make a new

target for him, which will be the `InnocentWoman` (Type: "Innocent Young Woman").

Well, that should do it. Or, you know, not ? We just learned that the real Richard knew how to actually COMBINE his spells, for ever more imaginative murders ("Hey, guys ! Guys ! I made a zombified pony ! You like ?")...

That means you'll have to do it too.

Modify the `ASpell` class in the following ways :

- Add a protected attribute called `combined_with`, typed as a pointer to `ASpell`, initialized to 0. We will consider the spell "combined" if `combined_with` is not 0.
- Add a `combine` function that returns void and takes a pointer to `ASpell`, which will be a setter for the `combined_with` attribute.
- Add an `isCombined` function, that returns True if the spell is combined with something, False otherwise.
- Modify the `getEffects` function to return, in addition to its own effects, the effects of the spell it's combined with (if any). The return value must look like this (If it's not clear, look at the example) :
 - If the spell is combined :
 - * If the "inner" spell is combined too : "<EFFECT1>, <EFFECT2>"
 - * Otherwise : "<EFFECT1> and <EFFECT2>"
 - Otherwise : "<EFFECT1>"
- Also, modify the `getName` function in the same way, to return the name of the current spell and the name of the contained spell, if any. It will be "<NAME1>--<NAME2>" if it's called on a combined spell, "<NAME1>" otherwise.
- For example, for a `Fwoosh` that is combined with a `Ponymorph`, the effects will be "fwooshed and turned into a nice, cute little pony" and the name will be "Fwoosh-Ponymorph".
- Add a `cloneCombine` function, with the same prototype as `clone`, which will make a deep copy of the current spell, including combined spells. You do NOT have to handle combination loops, like `SpellA` combined with `SpellB` itself combined with `SpellA`.

Since combining spells is a tiresome business, our `Warlock` will have to get a spell book where he'll be able to store his favorite spell combinations.

Make a `SpellBook` class, in canonical form, that can't be copied or instantiated by copy. It will have the following functions:

- `void learnSpell(ASpell*)`, that COPIES a spell in the book
- `void forgetSpell(string const &)`, that deletes a spell from the book, except if it isn't there
- `ASpell* createSpell(string const &)`, that receives a string corresponding to the name of a spell (or spell combo), creates it, and returns it.

Modify the `Warlock`, now, make it have a spell book that will be created with him and destroyed with him. Also make his `learnSpell` and `forgetSpell` functions call those of the spell book. (Note from the author : Right here is the exact moment when my pizza has been delivered. Hungry yet ?)

The `launchSpell` function will have to use the `SpellBook` to create the spell it's attempting to launch.

Finally, so that we don't have to go troll around for expendable people to give our `Warlock`, you'll make a target generator.

So, make a `TargetGenerator` class, in canonical form, and as before, non-copyable.

It will have the following functions:

- `void learnTargetType(ATarget*)`, teaches a target to the generator
- `void forgetTargetType(string const &)`, that makes the generator forget a target type if it's known
- `ATarget* createTarget(string const &)`, that creates a target of the specified type

Phew, that's done. Now here's a test main. Since I'm eating my pizza right now, it's not very thorough, and only tests part of what's asked of you. But hey, you're all grown up now, I don't have to do it for you, do I ?

```

int main()
{
    Warlock richard("Richard");

    richard.addTitle("Chief Warlock of the Brothers of Darkness");
    richard.addTitle("Lord of the Thirteen Hells");
    richard.addTitle("Emperor of the Black");
    richard.addTitle("Lord of the Undead");
    richard.addTitle("Mistress of Magma");
    richard.addQuote("You've just been Dick-rolled !");
    richard.addQuote("This day is fantastic ...");
    richard.addQuote("Please ignore the fact that my hand is on fire. It's not\
        meant to be an aggressive gesture, it's how I say hello !");
    richard.addQuote("You like what I do ?");

    Ponymorph* ponymorph = new Ponymorph();

    richard.learnSpell(ponymorph);

    Sprotch* sprotch_ponymorph = new Sprotch();
    sprotch_ponymorph->combine(ponymorph);

    richard.learnSpell(sprotch_ponymorph);

    InnocentWoman stephanieDeMonaco;

    richard.introduce();
    richard.launchSpell("Ponymorph", stephanieDeMonaco);
    richard.launchSpell("Sprotch-Ponymorph", stephanieDeMonaco);
    richard.talk();
}

```

```

zaz@naqadah:~$ ./a.out | cat -e
Richard: This looks like another boring day.$
Richard: I am Richard, Chief Warlock of the Brothers of Darkness, Lord of the
Thirteen Hells, Emperor of the Black, Lord of the Undead, Mistress of Magma !
And the mayor of a little village on the coast. Very scenic during springtime
, you should visit sometime !$
Innocent Young Woman has been turned into a nice, cute little pony !$
Innocent Young Woman has been sprotched and turned into a nice, cute little
pony !$
Richard: Please ignore the fact that my hand is on fire. It's not meant to be
an aggressive gesture, it's how I say hello !$
Richard: Ahhh, I see it clearly. This is the plane of SUCK...$

```

