Assignment 5: C++

Introduction
1. The assignment is due on 28.6.15 at 23:55.
2. The TA in charge of this assignment is Eliran Weiss. Questions should be sent to the mail eliranw@cs.technion.ac.il with the subject “236703 HW5”.
3. Read the instructions carefully, both in this document and in the provided code.
4. Make sure to write code that is clear, readable and reasonably documented. Document non-trivial parts of the code.
5. Execution speed is not a central issue in the homework assignments in this course. In any case of uncertainty between simplicity and performance, choose the simpler implementation.
6. Try to avoid code duplication – use methods you’ve already written as much as possible.
7. In order to avoid mistakes please read the FAQ section in the course website which will be updated on an ongoing basis.

Preface
In the lectures and tutorials of this course we touched on the subject of the casting mechanism in C++. In this assignment you will write your own versions of two of these casts: static_cast and dynamic_cast. Your version will be weaker than the original, but will comprise the main capabilities or static and dynamic casting in C++.

Part 1 – Implementing a New Static_Cast
In this section you will implement static casting. You can read more about this mechanism here. You must implement the static method:

```cpp
template<typename Dst, typename Src>
static Dst my_static_cast(Src src);
```

The method should be implemented in the class OOP5 (the class interface can be found in the provided code for the assignment).

For simplicity, the method should cast src to the type Dst only in the following cases:

- There exists an implicit cast from Src to Dst. To know whether an implicit cast exists, follow the provided implicit cast link and search for 'implicitly convertible'.
- If src is of a pointer/reference type, and the referenced type is a non-virtual ancestor type of the type Dst points to/references. (Reminder: in static casting there is no runtime check that the cast is 'legal').
The cast must be performed **statically** (during compile time). This means that if the method is called with illegal parameters (for example two parameters that are not in the same inheritance hierarchy and there is no implicit cast between them) then the program will **not compile**.

**Do not perform any checks during run time, an implementation that does not meet this requirement will receive 0 for this part of the assignment.**

If the cast is not possible, the compiler must report an error, the exact type of error does not matter as long as it only rises when the cast is illegal.

**Guidance:** Note that you may (and should) take advantage of tools in the language that will enable such checks. The main tool here is the *template specialization* mechanism, which you saw in the lecture. The mechanism allows you to create a sort of *if* during compile time. You may also use other static mechanisms such as: sizeof, enum/constexpr, implicit cast, static_assert and so on.

Try to break the task up into smaller sub-tasks. For example, try to think how you would check inheritance in a static manner, and then how you can use this to execute code accordingly.

Of course, you may not use the existing static_cast mechanism or any external libraries in order to implement the method.

**Example**
A use example for my_static_cast:

```cpp
class A {}
class B : public A{}
class C{}
int main() {
    A* b=new B();
    B* b2=OOP5::my_static_cast<B*>(b);
    A aa=OOP5::my_static_cast<A>(*b2);
    A* a=new A();
    b2=OOP5::my_static_cast<B*>(a); //works, but result is not safe
    C* c=OOP5::my_static_cast<C*>(a); //doesn't COMPIL
    return 0;
}
```

**Part 2.1 – Implementing a New Dynamic Cast**
In this section you will implement dynamic casting. You can read more about this mechanism [here](#). You must implement the following static method in the class OOP5:

```cpp
template<typename Dst, typename Src>
static Dst my_dynamic_cast(Src src);
```

The dynamic cast you implement should handle the following cases:
• If \textit{Src} and \textit{Dst} are not pointers or references the cast must fail \textit{during compilation}. (Either both should be pointers or both should be references).

• If an implicit cast is possible between \textit{Src} and \textit{Dst}, the cast should take place \textit{during compilation}.

• Otherwise, if \textit{Src} and \textit{Dst} inherit from the class OOPPolymorphic (described below) try to perform the cast during runtime. If the types are not an expansion of this class, the cast must fail \textit{during compilation}. The Dynamic casting algorithm should work as follow:
  1. Find the dynamic type of src, we'll call it Dyn. If Dyn isn't \textit{Src} or doesn't inherits from \textit{Src} - we should fail and stop.
  2. If \textit{Dst} inherits from \textit{Src} exactly once (whether directly or not), we'll check if a downcast is possible:
     2.1. If Dyn is \textit{Dst} or inherits from \textit{Dst} (doesn't matter how many times) the casting is possible and we will perform a downcast from \textit{Src} to \textit{Dst}.
     2.2. Otherwise, this means that \textit{Dyn} is above \textit{Dst} in the inheritance hierarchy and therefore the casting is impossible and we should fail.
  3. In any other case- we should fail. Notice that sidecasting is impossible in our mechanism.

When failing in runtime, pointers casting should return NULL and references casting should throw a \texttt{std::bad_cast} exception.

You may assume that there is no virtual inheritance in classes in this assignment.

\textbf{Guidance}: Try to use the same tools you used in the first part of the assignment to implement the static part of the cast.

\textbf{Part 2.2 Implementing the Dynamic Part of Dynamic Cast}

In order to inspect the types during run-time, each type must have a representation similar to the existing type\_info class.

To this end you must complete the definition of the struct \textit{Type} (defined in the provided code) with an implementation of your choosing. You must choose wisely a representation that will be unique for every class. You may not use \texttt{typeid}, or \texttt{type\_info} in general. Your struct should atleast implement the \texttt{==} operator for checking equality of types.

Additionally, we will want to know the relationship between representations of different types, in order to see if there exists an inheritance connection between them.

To do this you must manage some kind of static data structure in the class OOP5. Make sure to choose a fitting data structure, taking into account that multiple inheritance is allowed. Your data structure must support the following method:
1. static int InheritsFrom(Type* derived, Type* base);

The method returns the number of times the type represented by “derived” inherits, directly and indirectly, from the type represented by “base”.

A description of how the data structure should be updated:

Each new type that should be able to use the new dynamic downcast must inherit from the following class;

2. template <class T>
class OOPPolymorphic

The class definition can be found in the provided code. You may add methods and fields to the class as you wish, but you may not change existing definitions. You must implement the following methods:

3. private:
   
   void RegisterInheritance(const Type* base)

This method should update the static data structure in OOP5. The method will update the data structure such that base is a type representation that describes a parent class of the current object. Note that this is a private method, so the only objects that can execute it are friends of the class, meaning objects of type T (see the provided code).

You may assume that every type that inherits from OOPPolymorphic will call this method for each of its direct parent classes, except for the parent class OOPPolymorphic. The method will be called at the beginning of the bodies of class constructors. If the class has no parent other than OOPPolymorphic the method will be called with a NULL argument.

4. public:
   static const Type* GetType()

This method returns the type representation of the class. The method must work correctly after at least one instance of the current type is created (or an instance of an inheriting type). If no instance of the current type has yet been created, the method must return NULL.

After you have implemented these classes, think how they can be used to implement my_dynamic_cast easily.

5. public:
   virtual const Type* MyType() = 0

This is the polymorphic version of GetType, it should return the same representation as the static method of the type. You can assume this function will be implemented by every class that inherits from OOPPolymorphic by calling the
current GetType. Note: you don’t need to implement this function yourself, it is only described to show you the proper use when you create a class by yourself.

Example
A use example for my_dynamic_cast:

```cpp
class Base : public OOPPolymorphic<Base> {
public:
    Base() {
        RegisterInheritance(NULL); //we don't register "OOPPolymorphic" as a parent
    }
};
class Derived : public Base, public OOPPolymorphic<Derived> {
public:
    Derived() {
        OOPPolymorphic<Derived>::RegisterInheritance(Base::GetType());
    }
    virtual void f() {
        cout << "Derived Successfully" << endl;
    }
};
int main() {
    Base* b = new Derived();
    Derived* d = OOP5::my_dynamic_cast<Derived*>(b);
    d->f(); //prints 'Derived Successfully'
    return 0;
}
```

Additional Requirements
- You may use the standard libraries (especially when implementing the data structure) except for cases in which it has been stated otherwise.
- **You cannot use compilations warning as errors to abort compilation. E.g, flags like -- Werror are forbidden.**
- As stated above, you may not use existing methods that help to perform casting, such as static_cast, dynamic_cast or existing methods for run-time type information such as typeid.
- You may assume that the types do not include const.
- You may use C++11, but you may not use the method std::is_base_of.
- The assignment will be tested using g++ version 4.8.2 on Ubuntu 64 bit. You may not submit a makefile that uses a different compiler, and make sure the program can compile with the specified compiler version. **For your convenience, you may install and user g++ 4.7 on the t2 computers by running source /usr/local/gcc4.7/setup.sh, but note that not all features of c++11 are supported in this version, therefore if a feature doesn't work on 4.7, you should make sure it works on 4.8.2**
General Notes:

- The assignment is not long in terms of code, but it does require thought. Take time to consider your implementation before you start to code, especially how you will implement the compile-time requirements.
- You are encouraged to go over the relevant lectures and tutorials dealing with casting and templates.
- In any case of a contradiction between the original casting mechanisms and the assignment, the instructions in the assignment will determine the desired behaviour.
- Avoid code duplication.
- Testing of the assignment will be done using the casting mechanisms you implemented and the helper classes you created for implementation of dynamic casting.

Submission Details

- Requests for postponement, for any reason, must be sent to the TA in charge of the course (Nurit). Note that the course has a late-submission policy, meaning you can submit your assignment late without a postponement approval. Details can be found in the course site under “General Info”.
- The assignment must be submitted electronically. (Save the submission confirmation).
- Your submission should consist of a zip file with the name format: OOP5_<ID1>_<ID2>.zip containing:
  o A file called readme.txt with the following format:
    name1 id1 email1
    name2 id2 email2
  o Code files: you should submit the OOP5.h file with your changes and the OOP5.cpp file with the implementations, along with a makefile named Makefile.
- Points will be deducted for not conforming to the submission requirements (rar instead of zip, extra files, a readme file with the wrong name, etc.).

Good Luck!