CS310

“Advanced C++: Templates and Generic Programming”

Templates & Design
Templates and Inheritance

Hüseyin Akcan
huseyin.akcan@ieu.edu.tr
The Empty Base Class Optimization (EBCO)

- C++ classes are often "empty", which means that their internal representation does not require any bits of memory at run time.
- This is the case typically for classes that contain only type members, nonvirtual function members, and static data members.
- Nonstatic data members, virtual functions, and virtual base classes, on the other hand, do require some memory at run time.
- Even empty classes, however, have nonzero size

```cpp
class EmptyClass {
};

int main()
{
    std::cout << "sizeof(EmptyClass): " << sizeof(EmptyClass) << "\n";
}
```
The Empty Base Class Optimization (EBCO)

- The designers of C++ had various reasons to avoid zero-size classes. For example, an array of zero-size classes would presumably have size zero too, but then the usual properties of pointer arithmetic would not apply anymore.

- However, even though there are no zero-size types in C++, the C++ standard does specify that when an empty class is used as a base class, no space needs to be allocated for it provided that it does not cause it to be allocated to the same address as another object or subobject of the same type.

```cpp
class Empty {
    // typedef members don't make a class nonempty.
    typedef int Int;  
};

class EmptyToo : public Empty {
    
};

class EmptyThree : public EmptyToo {
    
};
```

Write the code to display the sizes of these classes and tell me the result!!!
The Empty Base Class Optimization (EBCO)

- Layout of EmptyThree by a compiler that implements the EBCO

![Diagram of EBCO implementation]

- Layout of EmptyThree by a compiler that does not implement the EBCO

![Diagram of EBCO non-implementation]
The Empty Base Class Optimization (EBCO)

- Any guess for the size of NonEmpty?
- Write the following code and display the sizes of the classes:

```cpp
class Empty {
    // typedef members don't make a class nonempty
    typedef int Int;
};

class EmptyToo : public Empty {
};

class NonEmpty : public Empty, public EmptyToo {
};
```
The Empty Base Class Optimization (EBCO)

- Class NonEmpty is not an empty class
- **The reason:** the base classes Empty and EmptyToo of NonEmpty cannot be allocated to the same address because this would cause the base class Empty of EmptyToo to end up at the same address as the base class Empty of class NonEmpty.

- The rationale for the constraint on empty base optimization stems from the fact that it is desirable to be able to compare whether two pointers point to the same object. Because pointers are nearly always internally represented as just addresses, we must ensure that two different addresses (that is, pointer values) correspond to two different objects.
The Curiously Recurring Template Pattern (CRTP)

- This pattern refers to a general class of techniques that consists of passing a derived class as a template argument to one of its own base classes.

```cpp
template <typename Derived>
class CuriousBase {
    ...
};

template <typename T>
class CuriousTemplate : public CuriousBase<CuriousTemplate<T> > {
    ...
};
```

- A simple application of CRTP consists of keeping track of how many objects of a certain class type were created. This is easily achieved by incrementing an integral static data member in every constructor and decrementing it in the destructor. However, having to provide such code in every class is tedious. Instead, we can write the code using CRTP.

See CRTP.cpp for the code