Object Oriented Programming with C++ Day 5

Compiled by
M S Anand

Department of Computer Science

OOP with C++
Introduction

Text Book(s):

1. “C++ Primer”, Stanley Lippman, Josee Lajoie, Barbara E Moo, Addison-Wesley Professional, 5th Edition..

Multiple Inheritance

Multiple Inheritance is a feature of C++ where a class can inherit from more than one class. The constructors of inherited classes are called in the same order in which they are inherited. The destructors are called in reverse order of constructors.

Syntax:

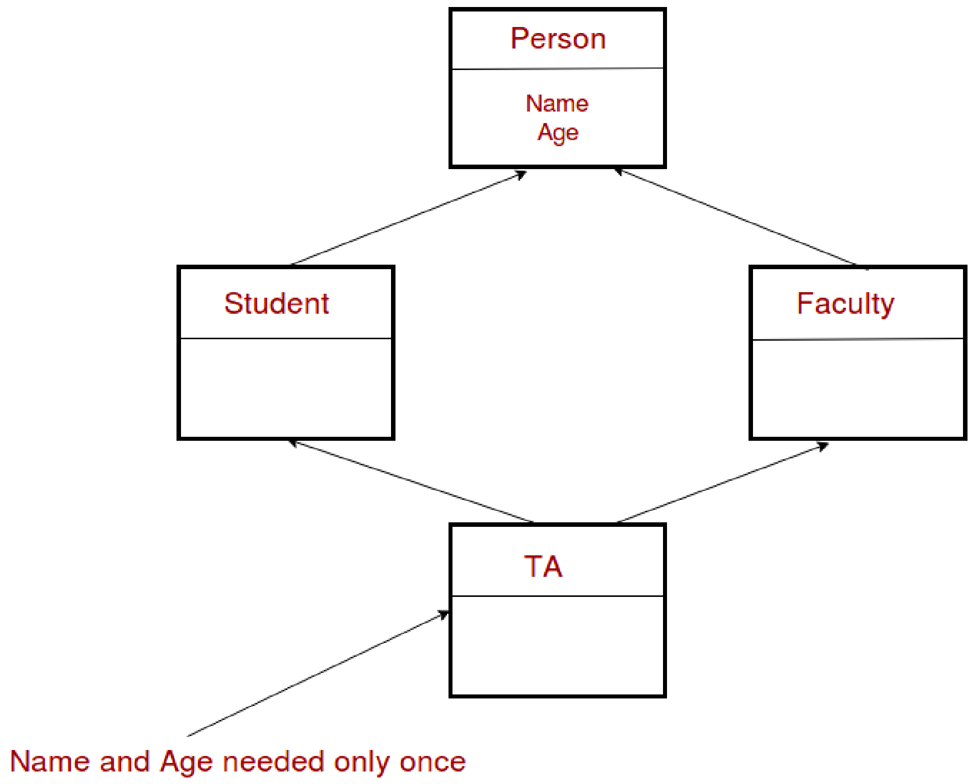
```
class A
{
... .. ...
};
class B
{
... .. ...
};
class C: public A,public B
{
... ... ...
};
```

Sample program: MI1.cpp

OOP with C++

Multiple Inheritance

The diamond problem The diamond problem occurs when two superclasses of a class have a common base class. For example, in the following diagram, the TA class gets two copies of all attributes of Person class, this causes ambiguities.



Multiple Inheritance

An example program: MI2.cpp

In the above program, constructor of ‘Person’ is called two times. Destructor of ‘Person’ will also be called two times when object ‘ta1’ is destructed. So object ‘ta1’ has two copies of all members of ‘Person’, this causes ambiguities. *The solution to this problem is ‘virtual’ keyword*. We make the classes ‘Faculty’ and ‘Student’ as **virtual base classes** to avoid two copies of ‘Person’ in ‘TA’ class.

The program with virtual base classes: MI3.cpp
In the above program, constructor of ‘Person’ is called once. One important thing to note in the above output is, *the default constructor of ‘Person’ is called*. **When we use ‘virtual’ keyword, the default constructor of grandparent class is called by default even if the parent classes explicitly call parameterized constructor**.

Multiple Inheritance

How do you call the parameterized constructor of the 'Person' class?

The constructor has to be called in 'TA' class.
The corresponding program: MI4.cpp

In general, it is not allowed to call the grandparent's constructor directly, it has to be called through parent class. It is allowed only when 'virtual' keyword is used.

Predict the output from the following programs:

MI5.cpp
MI6.cpp

Multiple Inheritance

Multilevel inheritance

An example: ML1.cpp

- ❑In this program, class C is derived from class B (which is derived from base class A).
- ❑The obj object of class C is defined in the main() function.
- ❑When the display() function is called, display() in class A is executed. It's because there is no display() function in class C and class B.
- ❑The compiler first looks for the display() function in class C. Since the function doesn't exist there, it looks for the function in class B (as C is derived from B).
- ❑The function also doesn't exist in class B, so the compiler looks for it in class A (as B is derived from A).
- ❑If display() function exists in C, the compiler overrides display() of class A (because of member function overriding).

OOP with C++

Multiple Inheritance

Hierarchical Inheritance

An example: HI1.cpp

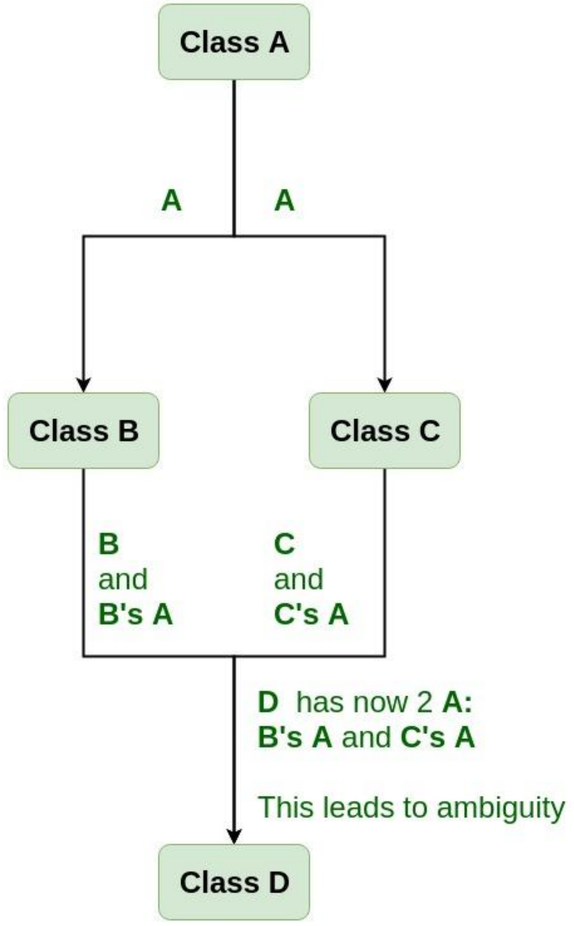
Multiple Inheritance

<u>**Virtual Base Class**</u>

Virtual base classes are used in virtual inheritance as a way of preventing multiple “instances” of a given class appearing in an inheritance hierarchy when using multiple inheritance.

Need for Virtual Base Classes: Consider the situation where we have one class **A** . This class **A** is inherited by two other classes **B** and **C**. Both these class are inherited into another in a new class **D** as shown in figure below.

As we can see from the figure, data members/function of class **A** are inherited twice to class **D**. One through class **B** and second through class **C**. When any data / function member of class **A** is accessed by an object of class **D**, ambiguity arises as to which data/function member would be called? One inherited through **B** or the other inherited through **C**. This confuses compiler and it displays error.



Typecasting

Typecasting

Type casting refers to the conversion of one data type to another in a program. Typecasting can be done in two ways: automatically by the compiler and manually by the programmer or user. Type Casting is also known as Type Conversion.

Typecasting

RTTI (Run-Time Type Information) in C++

In C++, **RTTI (Run-time type information)** is a mechanism that exposes information about an object's data type at runtime and is available only for the classes which have at least one virtual function. It allows the type of an object to be determined during program execution.

Runtime Casts

The runtime cast, which checks that the cast is valid, is the simplest approach to ascertain the runtime type of an object using a pointer or reference. This is especially beneficial when we need to cast a pointer from a base class to a derived type. When dealing with the inheritance hierarchy of classes, the casting of an object is usually required.

Typecasting

There are two types of casting:

Upcasting: When a pointer or a reference of a derived class object is treated as a base class pointer.

Downcasting: When a base class pointer or reference is converted to a derived class pointer..

OOP with C++

Typecasting

Using '*dynamic_cast*': In an inheritance hierarchy, it is used for downcasting a base class pointer to a child class. On successful casting, it returns a pointer of the converted type; however, it fails if we try to cast an invalid type such as an object pointer that is not of the type of the desired subclass.

For example, dynamic_cast uses RTTI and the following program fails with the error "***cannot dynamic_cast `b' (of type `class B*') to type `class D*' (source type is not polymorphic)*** " because there is no virtual function in the base class B.

The program: RTTI1.cpp (results in compilation error)

Corrected program: RTTI2.cpp

Composition

<u>Composition</u>
A class can have one or more objects of other classes as members. A class is written in such a way that object of another existing class becomes a member of the new class.

This interconnection between classes is known as C++ Composition. It is also known as containment, part-whole, or has-a relationship. A common form of software reusability is C++ Composition.

In C++ Composition, an object is a part of another object. The object that is a part of another object is known as sub object. When a C++ Composition is destroyed, then all of its sub objects are destroyed as well.

Composition

Composition

A simple program: Composition1.cpp

Composition

Types of Object Compositions:

There are two basic subtypes of object composition:

1. Composition: The composition relationships are part-whole relationships where a part can only be a part of one object at a time. This means that the part is created when the object is created and destroyed when the object is destroyed. To qualify as a composition, the object and a part must have the following relationship-

The part (member) is part of the object (class).

The part (member) can only belong to one object (class).

The part (member) has its existence managed by the object (class).

The part (member) does not know about the existence of the object (class).

Multiple Inheritance

2. Aggregation: The aggregation is also a part-whole relationship but here in aggregation, the parts can belong to more than one object at a time, and the whole object is not responsible for the existence of the parts. To qualify as aggregation, a whole object and its part must have the following relationships:

The part (member) is part of the object (class).

The part (member) can belong to more than one object (class) at a time.

The part (member) does not have its existence managed by the object (class).

The part (member) does not know about the existence of the object (class).

Templates

Templates

A **template** is a simple yet very powerful tool in C++. The simple idea is to pass the data type as a parameter so that we don't need to write the same code for different data types. For example, a software company may need to sort() for different data types. Rather than writing and maintaining multiple codes, we can write one sort() and pass the datatype as a parameter.

C++ adds two new keywords to support templates:
'template' and ***'type name'***. The second keyword can always be replaced by the keyword **'class'**.

How Do Templates Work?

Templates are expanded at compile time. This is like" macros". The difference is, that the compiler does type-checking before template expansion. The idea is simple, source code contains only function/class, but compiled code may contain multiple copies of the same function/class.

Templates

Sample program1: Template1.cpp

Sample program 2: Template2.cpp

Class templates

Class templates
Class templates like function templates, are useful when a class defines something that is independent of the data type. Can be useful for classes like LinkedList, BinaryTree, Stack, Queue, Array, etc.

Sample program: CTemplate1.cpp

Sample program: CTemplate2.cpp

Can we specify a default value for template arguments?
Yes, like normal parameters, we can specify default arguments to templates. The following example demonstrates the same.

Program: CTemplate3.cpp

Templates

What is the difference between function overloading and templates?
Both function overloading and templates are examples of polymorphism feature of OOP. Function overloading is used when multiple functions do quite similar (not identical) operations, templates are used when multiple functions do identical operations.

What happens when there is a static member in a template class/function?
Each instance of a template contains its own static variable.

Templates

Non type parameters to templates:
Program: Template3.cpp

Here is an example of a C++ program to show different data types using a constructor and template. We will perform a few actions
passing character value by creating an object in the main() function.
passing integer value by creating an object in the main() function.
passing float value by creating an object in the main() function.
Program: Template4.cpp

Exception handling

Exception Handling

Exception Handling. Exceptions are runtime anomalies or abnormal conditions that a program encounters during its execution.

C++ provides the following specialized keywords for this purpose:
try: Represents a block of code that can throw an exception.
catch: Represents a block of code that is executed when a particular exception is thrown.
throw: Used to throw an exception. Also used to list the exceptions that a function throws but doesn’t handle itself.

Exception handling

Exception Handling

C++ Exceptions:

When executing C++ code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, C++ will normally stop and generate an error message. The technical term for this is: C++ will throw an exception (error).

C++ try and catch:

Exception handling in C++ consists of three keywords: try, throw and catch:

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The throw keyword throws an exception when a problem is detected, which lets us create a custom error.

The catch statement allows you to define a block of code to be executed if an error occurs in the try block.

OOP with C++

Exception handling

Sample program1: CException1.cpp

There is a special catch block called the ‘catch all’ block, written as catch(…), that can be used to catch all types of exceptions. For example, in the following program, an int is thrown as an exception, but there is no catch block for int, so the catch(…) block will be executed.

Sample program: CException2.cpp

THANK YOU

M S Anand
Department of Computer Science Engineering
anandms@yahoo.com