C++ Programming

Trainer : Rajiv K

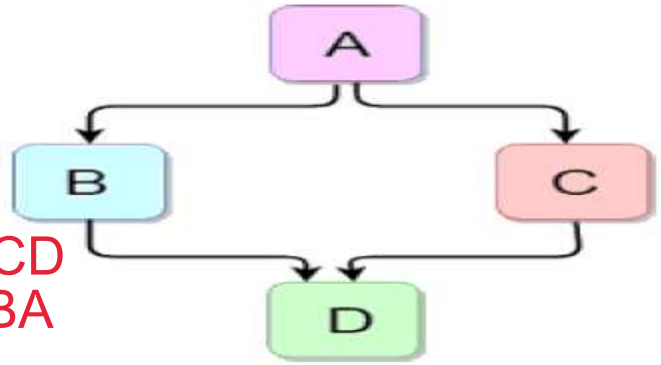
Email: rajiv.kamune@sunbeaminfo.com

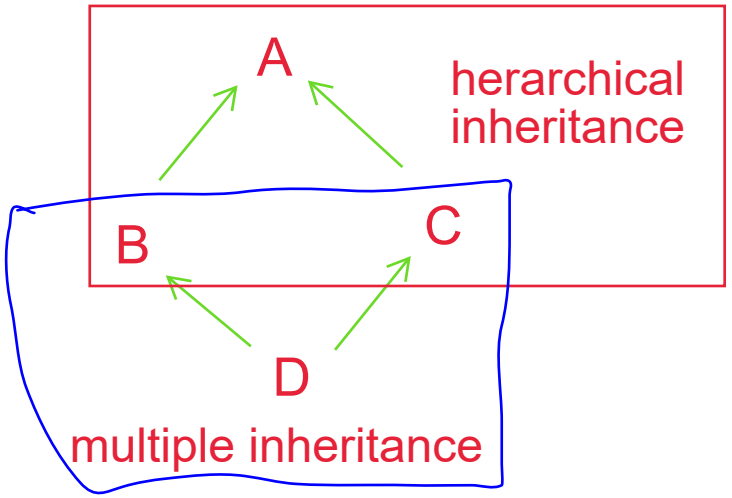
Diamond Problem

- As shown in diagram it is hybrid inheritance. Its shape is like diamond hence it is also called as diamond inheritance.
- Data members of indirect base class inherit into the indirect derived class multiple times. Hence it effects on size of object of indirect derived class.
- Member functions of indirect base class inherit into indirect derived class multiple times. If we try to call member function of indirect base class on object of indirect derived class, then compiler generates ambiguity error.
- If we create object of indirect derived class, then constructor and destructor of indirect base class gets called multiple times.
- All above problems generated by hybrid inheritance is called diamond problem.

A is parent of B and C
D is child of B and C
D is indirect child of A
A is indirect parent of D

constructor calling sequence =ABDACD
destructor calling sequence =DCABDBA





Solution to Diamond Problem– Virtual Base Class

- If we want to overcome diamond problem, then we should declare base class virtual i.e. we should derive class B & C from class A virtually. It is called virtual inheritance. In this case, members of class A will be inherited into B & C but it will not be inherited from B & C into class D.

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

constructor calling sequence =ABCD
destructor calling sequence =DCBA

Function overriding :

- Function defined in base class is once again redefine in derived class having same name as in it's base class and same signature is called Function overriding
- Function which takes part in overridding is called as overriding function
- For function overriding inheritance is important.

A function in derived class having same name and signature as parent class is called function overriding.

Inheritance is must for function overriding.

Virtual Keyword

- Virtual functions allow us to create a list of base class pointers and call methods of any of the derived classes without even knowing kind of derived class object.
- **Early Binding**
- When we use Base class's pointer to hold Derived class's object, base class pointer or reference will always call the base version of the function
- **Late Binding**
- **Using Virtual Keyword in C++**
- We can make base class's methods virtual by using **virtual** keyword while declaring them. Virtual keyword will lead to Late Binding of that method.
- On using Virtual keyword with Base class's function, Late Binding takes place and the derived version of function will be called, because base class pointer pointes to Derived class object.
- **Points to note**
  - **Only the Base class Method's declaration needs the Virtual Keyword, not the definition.**
  - If a function is declared as **virtual** in the base class, it will be virtual in all its derived classes.
  - The address of the virtual Function is placed in the **VTABLE** and the compiler uses **VPTR**(vpointer) to point to the Virtual Function

Program Demo

Early Binding

create a class Base and Derived (void print() in both classes)

create base *bptr;

bptr=&d;

bptr->print() //print will be called of Base class

Late Binding

create a class Base and Derived (void display() in both classes one as virtual in base class)

create base *bptr;

bptr=&d;

bptr->display() //print will be called of Derived class

RTTI

- RTTI means **Run Time Type Information**
- RTTI is the process of **getting information of object at runtime.**
- RTTI (Run-time type information) is a mechanism that **exposes information about an object's data type at runtime.**
- It 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.
- There are two main C++ language elements to run-time type information:
  - 1. The typeid operator.
  - 2. The type_info class

RTTI

- **1. The typeid operator.**
- Used for identifying the exact type of an object.
- It can be used to get the runtime information of an object.
- This operator returns a type_info instance. The std::type_info is a class that holds implementation specific information of a type, such as type name, and means to compare the equality of two types.
- **2. The type_info class**
- Used to hold the type information returned by the typeid operator.
- You cannot instantiate objects of the type_info class directly, because the class has only a private copy constructor.
- The only way to construct a (temporary) type_info object is to use the typeid operator.
- The header file must be included in order to use the type_info class.
- The operators == and != can be used to compare for equality and inequality with other type_info objects, respectively.

RTTI

- #include<iostream>
- using namespace std;
- int main()
- {

```
int number;
const type_info& type =typeid(number);
srting typeNamae=type.name();
cout<<“Type:”<<typeName<<endl;
return 0;
```

- }

Pure virtual function and Abstract class

- Virtual fun which is **equated to zero** such function is called as **Pure virtual function**
- Pure virtual function **does not have body.**
- A class which contains **at least one Pure virtual function** such class is called as "Abstract class".
- If class is Abstract **we can not create object of that class** but **we can create pointer or reference** of that class .
- It is not compulsory to override virtual function but It is **compulsory to override Pure virtual function**
- If we not override pure virtual function in derived class at that time derived class can be **treated as abstract class.**

```
virtual void accept()=0;// pure virtual function
```

Abstract class: a class having atleast 1 pure virtual function.

Abstract Class

Sometimes implementation of all function cannot be provided in a base class because we don't know the implementation. Such a class is called abstract class

```
class Shape{
public:
    virtual void accept() = 0; // Pure virtual function
    virtual void CalculateArea() = 0;  // Pure virtual function
};
class Circle : public Shape
{
private: int radius;
public:
    void accept() {
        cout << "Enter radius  = ";
        cin >> this->radius;
    }
    void CalculateArea() {
        cout << "Area of circle = " << 3.14 * radius * radius;
    }};
```

```
int main(){
    int choice = 0;
    Circle c;
    Rectangle r;
    Shape *s;
    do{
        cout << "0.Exit" << endl;
        cout << "1.Area of Circle";
        cout << "2.Area of Rectangle";
         cin >> choice;
        switch (choice)
        {
        case 1:  s = &c;   break;
        case 2:  s = &r;   break;
        }
        s->accept();
        s->CalculateArea();
    } while (choice != 0);
    return 0;}
```

Interface

- If class contains only pure virtual functions (no data members & other functions), then it is called as “interface”.
- They force (all) pure virtual functions to be overridden in derived class.
- These functions are called on base pointer or reference to achieve runtime polymorphism.
- Typically interfaces are used to design specifications/standards

INTERFACE : contains only pure virtual function nothing else.

Upcasting and downcasting

- Upcasting and downcasting :-

Upcasting - It is the process to create the derived class's pointer or reference from the base class's pointer or reference, and the process is called *Upcasting*.

eg : Person *p=new student(); //Ok

child class ke object ke address ko
Parent class ke pointer me store krwaye
tb UPCASTING

Downcasting - The *Downcasting* is an opposite process to the upcasting, which converts the base class's pointer or reference to the derived class's pointer or reference.

eg : Student *s=new Person();// not ok

Parent class ke object ke address ko
Child class ke pointer me store krwaye
tb UPCASTING

Object Slicing

- If derived class object is assigned to base class object, only base class part present in the derived class object is assigned to the base class object.
- If derived class object's address is assigned to base class pointer, only address of base class part present in the derived class object is assigned to that base class pointer.
- With such pointer/reference only base class members of derived class object can be accessed

What we have covered in this module?

- 1 : Introduction to C++
- 2 : Function features
- 3 : class and object
- 4 : namespaces
- 5 : static, const, friend
- 6 : memory management
- 7 : Object oriented concept
- 8 : composition
- 9 : Inheritance
- 10: virtual function
- 11: Advance C++ feature

Thank You