# C++ Programming

Trainer : Rohan Paramane

Email: rohan.paramane@sunbeaminfo.com

# Agenda

- Association
  - Composition
  - Aggregation
- Inheritance
- Mode of Inheritance
- Types of Inheritance
- Diamond Problem
- Virtual Base Class
- Virtual Keyword
- Virtual Function
- Abstract Class

# Association

- If has-a relationship exist between two types then we should use association.
- Example : Car has-a engine (OR engine is part-of car)
- If object is part-of / component of another object then it is called association.
- If we declare object of a class as a data member inside another class then it represents association.
- Example Association:

```
class Engine
{ };
class Car
{               private:
    Engine e;   //Association
};
int main( void )
{   Car car;
    return 0;
}
//Dependant Object : Car Object
//Dependancy Object : Engine Object
```

# Composition and aggregation are specialized form of association

**Composition**

- If dependency object do not exist without Dependant object then it represents composition.
- Composition represents tight coupling.
- Example: Human has-a heart.

```
class Heart
{ };
class Human
{   Heart hrt; //Association->Composition
};
int main( void )
{   Human h;
    return 0;
}
```

- //Dependant Object : Human Object
- //Dependancy Object : Heart Object

**Aggregation**

- If dependency object exist without Dependant object then it represents Aggregation.
- Aggregation represents loose coupling.

```
class Faculty
{ };
class Department
{
    Faculty f; //Association->Aggregation
};
int main( void )
{
    Department d;
    return 0;
}
```

- //Dependant Object : Department Object
- //Dependancy Object : Faculty Object

# Inheritance

- If "is-a" relationship exist between two types then we should use inheritance.
- Inheritance is also called as "Generalization".
- Example: Book is-a product
- During inheritance, members of base class inherit into derived class.
- If we create object of derived class then non static data members declared in base class get space inside it.
- Size of object = sum of size of non static data members declared in base class and derived class.
- If we use private/protected/public keyword to control visibility of members of class then it is called access Specifier.
- If we use private/protected/public keyword to extend the class then it is called mode of inheritance.
- Default mode of inheritance is private.
  - Example: class Employee : person //is treated as class Employee : private Person
- Example: class Employee:public Person
- In all types of mode, private members inherit into derived class but we can not access it inside member function of derived class.
- If we want to access private members inside derived class then:
  - Either we should use member function(getter/setter).
  - or we should declare derived class as a friend inside base class.

# Syntax of inheritance in C++

| | |
|---|---|
| **class Person //Parent class**<br>{ };<br>**class Employee : public Person //** Child class<br>{ }; | In C++ Parent class is called as Base class and child class is called as derived class. To create derived class we should use colon(:) operator. As shown in this code, public is mode of inheritance. |
| **class Person //Parent class**<br>{ **char name[ 30 ]; int age;** };<br>**class Employee : public Person //Child class**<br>{ **int empid; float salary;** };<br>**int main( void )**<br>{<br>Person p;<br>cout<<**sizeof( p )<<endl;**<br>Employee emp;<br>cout<<**sizeof( emp )<<endl;**<br>**return 0;**<br>} | If we create object of derived class, then all the non- static data member declared in base class & derived class get space inside it i.e. non-static static data members of base class inherit into the derived class. |

# Syntax of inheritance in C++

- Using derived class name, we can access static data member declared in base class i.e. static data member of base class inherit into derived class.

```
class Base{
protected:
static int number;
};
int Base::number = 10;
class Derived : public Base{
public:
static void print( void )
{ cout<<Base::number<<endl; }
};
int main( void ){
Derived::print();
return 0;
}
```

```
class Derived : public Base
{
int num3;
static int num4;
public:
void setNum3( int num3 )
{ this->num3 = num3; }
static void setNum4( int num4 )
{ Derived::num4 = num4;}
};
int Derived::num4;
```

```
int main( void )
{
Derived d;
d.setNum1(10);
d.setNum3(30);
Derived::setNum2(20);
Derived::setNum4(40);
return 0;
}
```

# Except following functions, including nested class, all the members of base class, inherit into the derived class

- Constructor
- Destructor
- Copy constructor
- Assignment operator
- Friend function.

# Mode of inheritance

- If we use private, protected and public keyword to manage visibility of the members of class then it is
- called as access specifier.
- But if we use these keywords to extends the class then it is called as mode of inheritance.
- C++ supports private, protected and public mode of inheritance. If we do not specify any mode, then default mode of inheritance is private.

# Mode Of inheritance

| Public Mode of inheritance | | | | | |
|---|---|---|---|---|---|
| **Access Specifier** | **Same class** | **Derived class** | **Indirect Derived class** | **Friend function** | **Non- member function** |
| Private | A | NA | NA | A | NA |
| Protected | A | A | A | A | NA |
| Public | A | A | A | A | A |
| **Private Mode of inheritance** | | | | | |
| **Access Specifier** | **Same class** | **Derived class** | **Indirect Derived class** | **Friend function** | **Non- member function** |
| Private | A | NA | NA | A | NA |
| Protected | A | A | NA | A | NA |
| Public | A | A | NA | A | A : Base<br>NA : Derived |

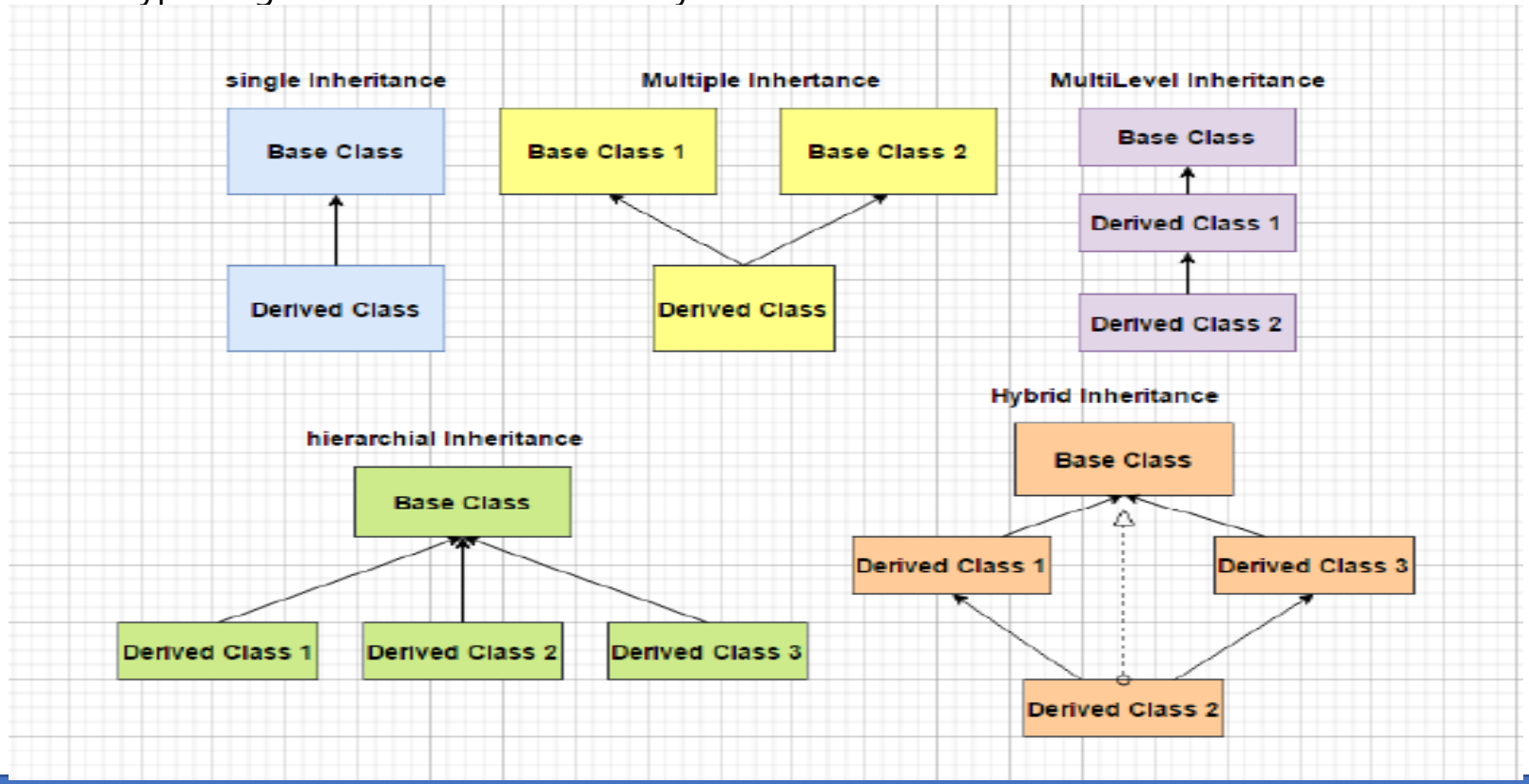
# Mode Of inheritance Cont....

| Protected Mode of inheritance | | | | | |
|---|---|---|---|---|---|
| **Access Specifier** | **Same class** | **Derived class** | **Indirect Derived class** | **Friend function** | **Non- member function** |
| Private | A | NA | NA | A | NA |
| Protected | A | A | A | A | NA |
| Public | A | A | A | A | A: Base<br>NA: Derived |

# Types of Inheritance

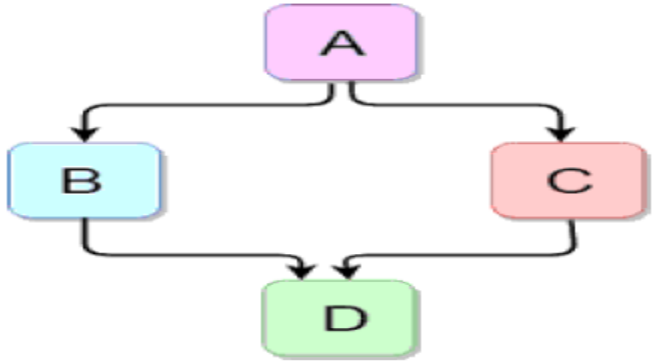
- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance

If we combine any two or more types together then it is called as hybrid inheritance.



# 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.



# 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
{ };
```

# 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 show() in both classes)

create base *bptr;

bptr=&d;

bptr->show()

**Late Binding**

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

create base *bptr;

bptr=&d;

bptr->show()

# 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.

```
classShape {
public:
virtualintArea() = 0; // Pure virtual function is declared as follows.
// Function to set width.
voidsetWidth(int w) {
width = w;
}
// Function to set height.
voidsetHeight(int h) {
height = h;
}

protected:
intwidth;
intheight;
};
 // A rectangle is a shape; it inherits shape.
// A triangle is a shape too; it inherits shape.
```

```
intmain() {
Rectangle R;
Triangle T;

R.setWidth(5);
R.setHeight(10);

T.setWidth(20);
T.setHeight(8);

cout <<"The area of the rectangle is: "<<
R.Area() <<endl;
cout <<"The area of the triangle is: "<< T.Area()
<<endl;
}
```

# Thank You