C++ Programming

Trainer : Akshita Chanchlani

Email: akshita.chanchlani@sunbeaminfo.com

C++ friend Function and friend Classes

- It is a mechanism built in C++ programming to access private or protected data from non-member functions.
- This is done using a friend function or/and a friend class.
- If a function is defined as a friend function then, the private and protected data of a class can be accessed using the function.
- The complier knows a given function is a friend function by the use of the keyword **friend**.
- For accessing the data, the declaration of a friend function should be made inside the body of the class (can be anywhere inside class either in private or public section) starting with keyword friend.
- **Declaration of friend function in C++**
  - Syntax : class class_name { ... . friend return_type function_name(argument/s); ... . }
  - Now, you can define the friend function as a normal function to access the data of the class.
    No friend keyword is used in the definition.

    class className { ... . friend return_type functionName(argument/s); ... . }

    return_type functionName(argument/s) { ... . // Private and protected data of className can be accessed from // this function because it is a friend function of className. ... . }

Example Friend function

- If we want to access private members inside derived class
  - Either we should use member function(getter/setter).
  - Or we should declare a facilitator function as a friend function.
  - Or we should declare derived class as a friend inside base class.

We can declare global function (including main function) as well as member function as a friend inside class. We can write friend declaration statement inside any section(private/ protected/public) of the class. Consider this code snippet:	**class Test**
	{
	private: int number;
	public:
	Test(void)
	{ **this->number = 10** }
	friend void print(void);
	};
	void print(void)
	{ Test t; cout<<"Number : "<<t.number<<endl; }
	int main(void)
	{ print();
	return 0;
	}

friend Class in C++ Programming

- like a friend function, a class can also be made a friend of another class using keyword friend.

```
class B;
class A
{
        // class B is a friend class of class A
        friend class B;
         ... .. ...
}
class B
 {
... .. ...
}
```

In this example , all member functions of class B will be friend functions of class A. Thus, any member function of class B can access the private and protected data of class A.
But, member functions of class A cannot access the data of class B.

- When a class is made a friend class, all the member functions of that class becomes friend functions.

C++ Inheritance

- Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application.
- It provides an opportunity to reuse the code functionality and fast implementation time.
- When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base** class, and the new class is referred to as the **derived** class.
- Syntax:
  - class derived-class: access-specifier base-class
- 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.

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;
```

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;
```

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 )
{
```

Access Control and Inheritance / Mode of inheritance

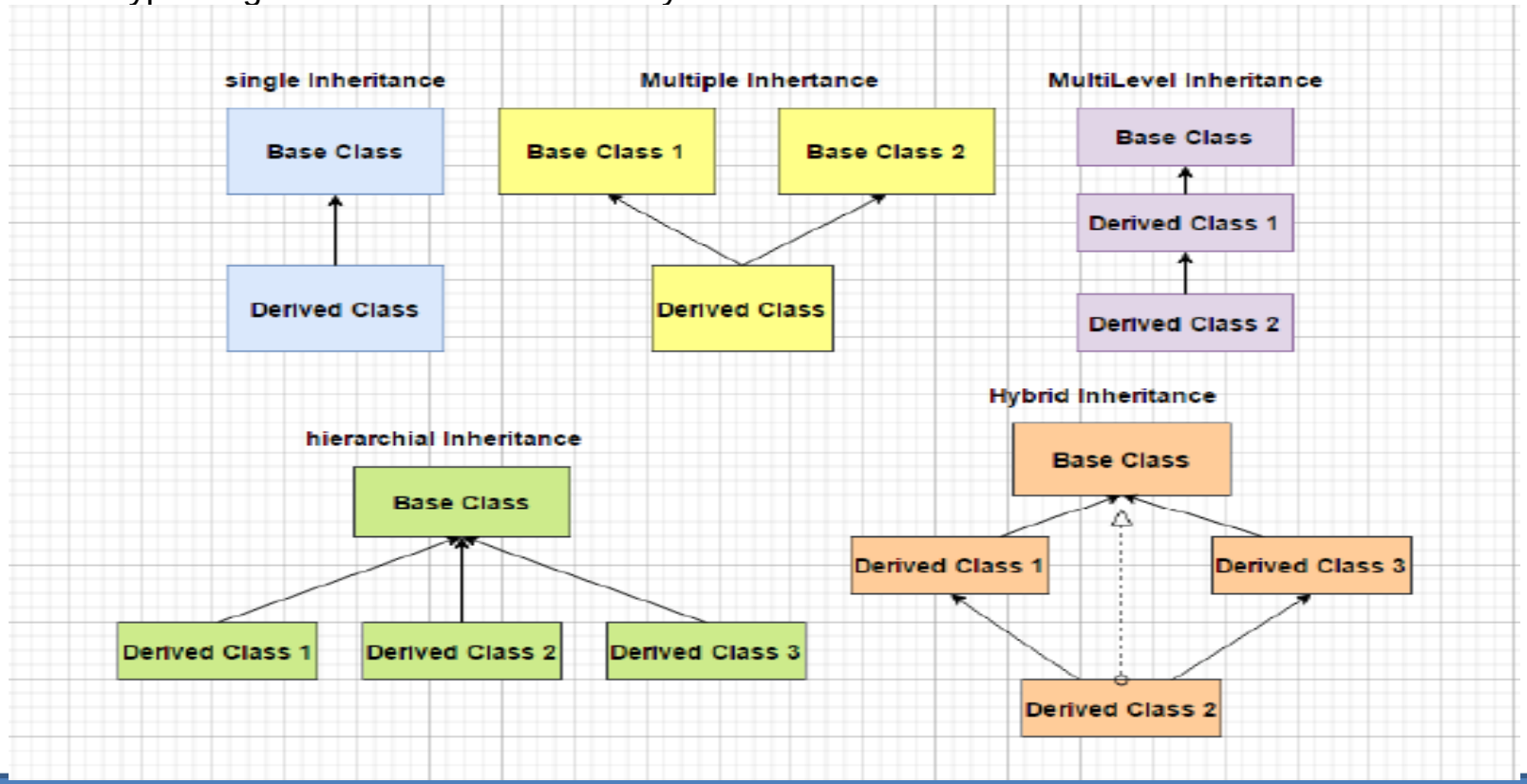
Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

- 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.

Types of Inheritance

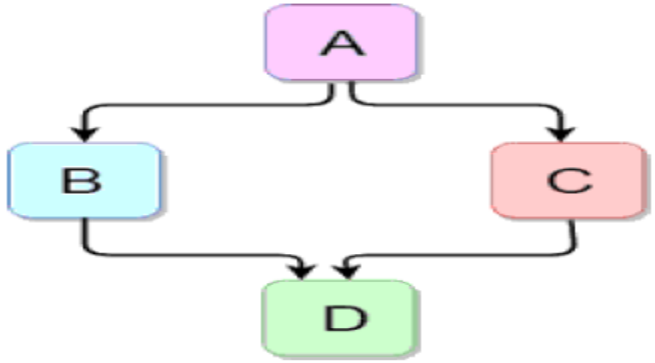
- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance

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 Function

- 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

Virtual Function calling

- virtual function is designed to call using base class pointer/reference.
- In C++, constructor is not designed to call on object, pointer or reference explicitly hence we cannot declare constructor virtual but we can declare destructor virtual.
- Virtual function implementation is implicitly based on virtual function pointer and virtual function table.
- If we declare member function virtual then compiler implicitly maintain one table to store address of declared virtual member function. It is called as virtual function table.
- To store address of virtual function table compiler implicitly declare one pointer as a data member of class. It is called as virtual function pointer.
- Due to virtual function pointer size of the object gets increased either by 2/4/8 bytes, depending on type of compiler.

Function Overriding

- Process of redefining virtual member function of base class inside derived class with same signature is called function overriding.
- According to client's requirement, if implementation of base class member function is logically 100% complete then it should be non-virtual.
- According to client's requirement, if implementation of base class member function is partially complete then it should be virtual.
- According to client's requirement, if implementation of base class member function is logically 100% incomplete then it should be pure virtual.

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()

Pure Virtual Function and Abstract Classs

- If we equate virtual function to zero then such virtual function is called pure virtual function.
- If class contains at least one pure virtual function, then it is called as abstract class.
- In this code class Shape contains pure virtual function hence it is considered as abstract class.
- If class contains all pure virtual function, then it is called as pure abstract class / interface.

```
class Shape    //Abstract class
{
protected:
        float area;
public:
        Shape( void ) : area( 0 )
        {       }
        virtual void acceptRecord( ) = 0;

        virtual void calculateArea( ) = 0

        void printRecord( void )const
        {
                cout<<"Area : "<<this->area<<endl;
        }
};
```

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;
}
```

```
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