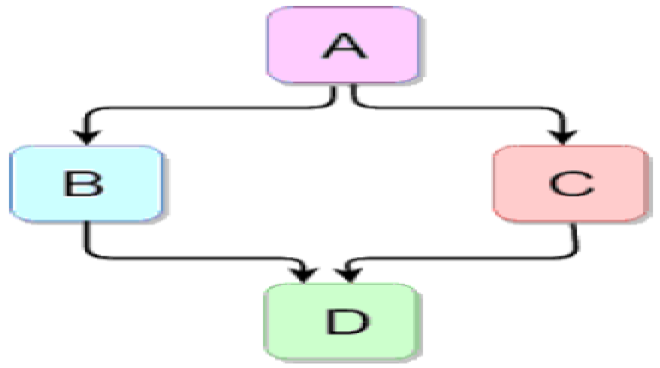
C++ Programming

Trainer : Pradnyaa S. Dindorkar

Email: pradnya@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.



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
{ };
```

Object slicing

- When a derived class object is assigned to a base class object in C++, the derived class object’s extra attributes are sliced off (not considered) to generate the base class object; and this whole process is termed **object slicing**.

- when extra components of a derived class are sliced or not used and the priority is given to the base class’s object this is termed object slicing.

- Class base{};
- Class derived :public base {};

```
Main()
 {
            base b ;      derived d ;
            b=d;         //object slicing.
}
```

Virtual Keyword

- **Virtual function** = It is the function which is called depending on type of object rather than type of pointer

- If class contains at least one virtual function then such class is called **polymorphic class**.

- If signature of base class and derived class member function is same and if function in base class is virtual then derived class member function is by default considered as virtual.

Function overriding.

Process of redefining, virtual function of base class, inside derived class with same signature is called function overriding.

- Virtual function redefined inside derived class is called overrided function.
- For function overriding:
    1. Functions must be exist inside base class and derived class.
    2. Signature of base class and derived function must be same.
    3. Function in base class must be 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 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.

Upcasting and downcasting

- Upcasting and downcasting :-

  Upcasting - process of converting derived class pointer into base class pointer

  or Storing address of derived class object into base class pointer.

  eg : Person *p=new student();

  Downcasting - process of converting base class pointer into derived class pointer

  or storing address of base class object into derived class pointer

  eg : Student *s=new person();

Static Variable

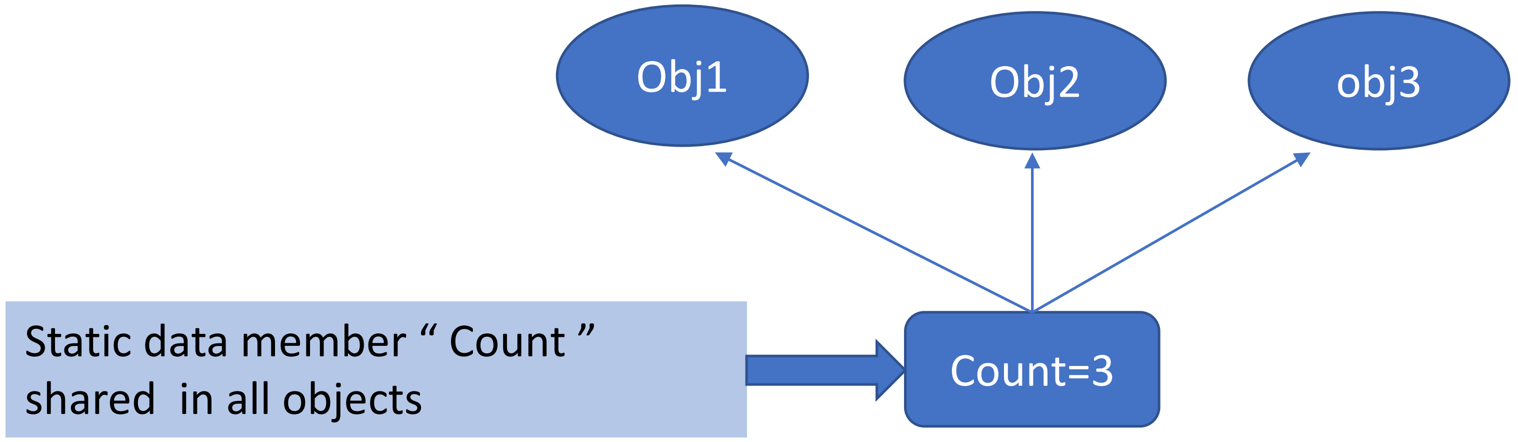
- All the static and global variables get space only once during program loading / before starting execution of main function
- Static variable is also called as shared variable.
- Initialized static and global variable get space on Data segment.
- Default value of static and global variable is zero.
- Static variables are same as global variables but it is having limited scope.

Static Data member

- If we want to share value of data member between all objects of same class then we should declare that data member as static data member.

- It is mandatory to provide global definition of static data member otherwise linker generates error.

- Static data member get space during class loading per class so it is called as **class-level variable**



Static Member Function

- Except main function, we can declare global function as well as member function static.
- static members of the class we should declare member function **static**.
- static member function is also called as **class level method.**
- To access class level method we should use classname and ::(scope resolution) operator.
- This pointer is not available in static member function .

Design pattern

- ➢ A design patterns are **well-proved solution** for solving the specific problem/task.
- ➢ Design patterns are programming language independent for solving the common object-oriented design problems.
- ➢ Design pattern represents an idea, not a particular implementation.
- ➢ Using design patterns you can make your code more flexible, reusable, and maintainable.
- ➢ To become a professional software developer, you must know at least some popular solutions (i.e. design patterns) to the coding problems.
- ➢ They are well-proved and testified solutions since they have been built upon the knowledge and experience of expert software developers.

Examples

1) Singleton Design pattern
2) Factory Design pattern
3) Builder Design pattern
4) Adapter Design pattern
5) Iterator Design pattern

Singleton Design pattern

- Singleton pattern is a software design pattern that restricts the instantiation of a class to one "single" instance/object. This is useful when exactly one object is needed to coordinate actions across the system.
- For example a single DB connection shared by multiple objects as creating a separate DB connection for every object may be costly.

```
class singleton{
    static singleton *ptr;
    singleton(){
        cout<<"in singleton()";
    }
    public:
    static singleton* getObject(){
        cout<<"\n in getObj()";
        if(ptr==NULL)
            ptr=new singleton();
        return ptr;
    }
};
```

```
singleton* singleton::ptr=NULL;

int main()
{
    singleton *ptr=singleton::getObject();
    return 0;
}
```

Friend :-

- A non member function of a class which designed to access private data of a class is called friend function.

- To declare a function as a friend of a class, precede the function prototype in the class definition with keyword **friend**

```
class MyData
{
private:
		int pin;
		int pass;

public:
		MyData(int pin,int pass);
		void PrintMyAccDetails();
		friend void anyFunction();
};
```

```
void anyFunction()
{
		MyData d1;
		d1.pass=9898;
		d1.pin=9999;
		d1.PrintMyAccDetails();
}
```

C++ is not a pure Object Oriented Language Because

- You can write code without creating a class in C++, and main() is a global function.
- Support primitive data types, e.g., int, char, etc. Instances(variable) of these primitive types are NOT objects.
- C++ provides "Friend" which is absolute corruption to

  the OO-Principle of encapsulation.

Thank You