Department of Commerce (CA)

Object Oriented Programming with C++

Sem: II Sub Code: 18MM23C

Unit - V I. M. Com CA

Virtual functions and polymorphism - need for virtual function - pointers to derived class objects. Pure virtual functions - Abstract classes. Rules for virtual functions. - Data file operations - opening of file - closing of file. Stream state member functions - Reading / writing a character from a file - Structure and file operations.

Reference Books:

1. Programming with C++ — D. Ravichandran
2. Mastering C++ — K. R. Venugopal, Rajkumar, T. Ravishankar.
3. Object Oriented programming with C++ — E. Balaguruswamy

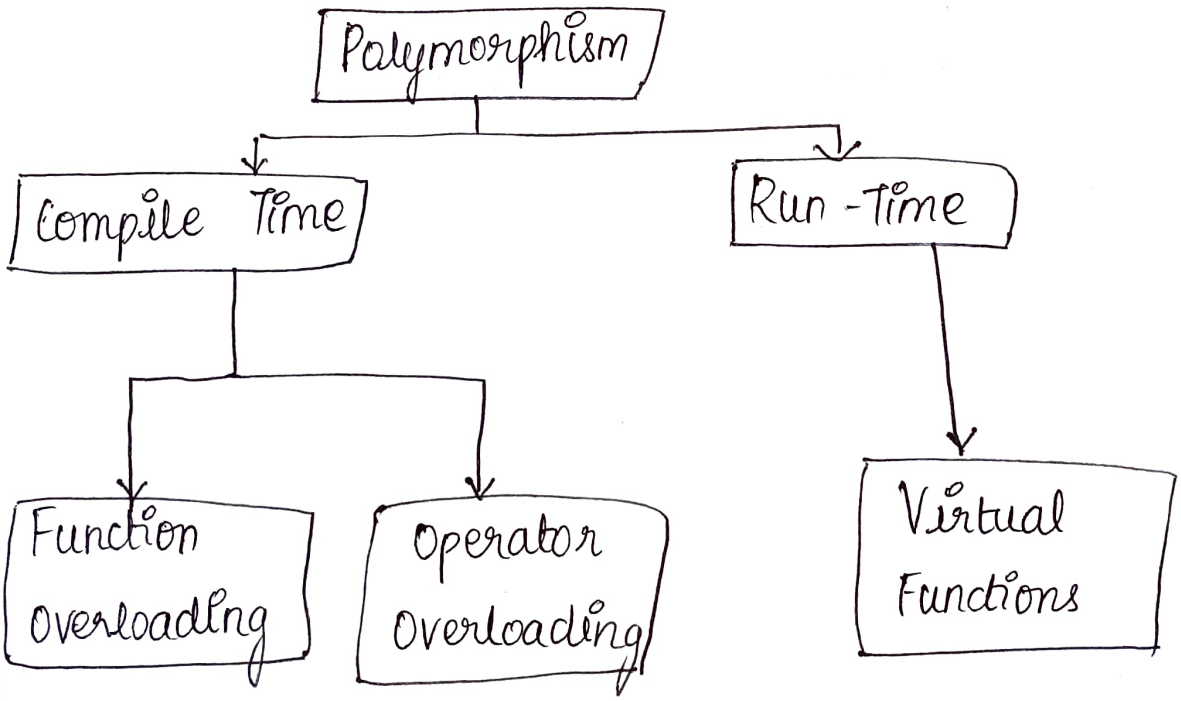
prepared by

D. S. VASANTHA

Assist professor.

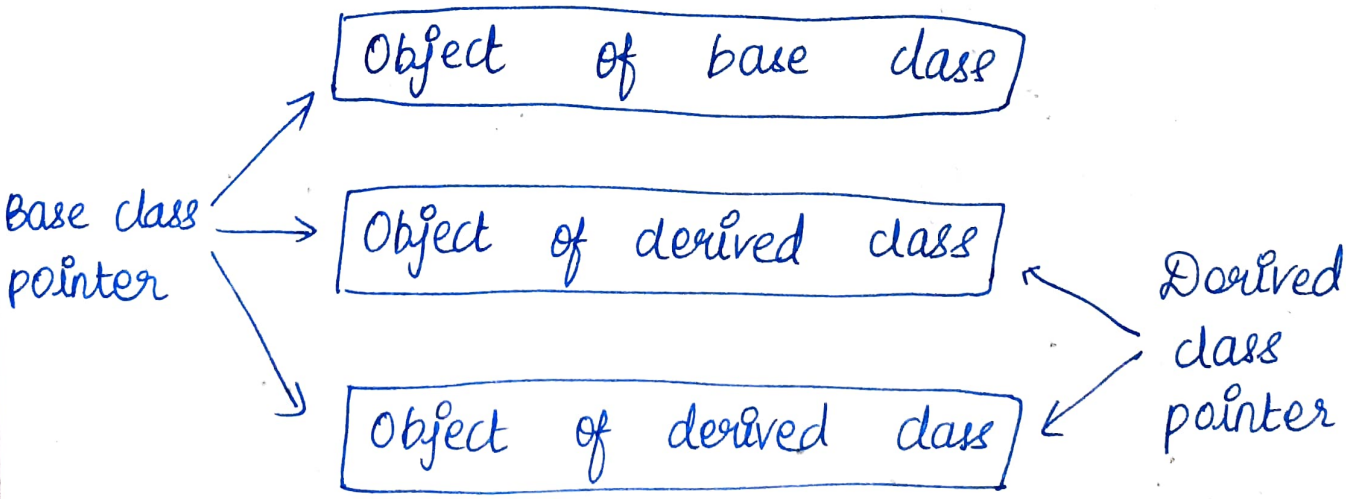
VIRTUAL FUNCTIONS

Polymorphism in biology means the ability of an organism to assume a variety of forms. In C++, it indicates the form of a member function that can be changed at runtime. Such member functions are virtual functions and the corresponding class is called polymorphic class. The objects of the polymorphic class, addressed by pointers, change at runtime and respond differently for the same message. Such a mechanism requires postponement of binding of a function call to the member function until runtime.



Pointer to Derived class Objects:

Pointers can be used with the objects of base classes or derived classes. Pointer to objects of a base class are type-compatible with pointers to objects of a derived class, thus allowing a single pointer variable to be used as pointer to objects of a base class and its derived classes. C++ makes polymorphism possible through a rule that one should memorize: a base class pointer may address an object of its own class or an object of any class derived from the base class.



Definition of Virtual Functions:

Virtual functions should be defined in the public section of a class to realize its full potential benefits. When such a declaration is made, it allows to decide which function to be used at runtime, based on the type of object, pointed to by the base pointer, rather than the type of the pointer. The program family 2.cpp illustrates the use of base pointer to point to different objects for executing different implementations of the virtual functions.

Syntax of virtual function

```
class Myclass
{
    public :
    ......
    ......
    virtual ReturnType FunctionName(arguments)
    {
        ......
        ......
    }
    ......
};
```

Pure Virtual Functions:

A pure virtual function declared in a base class has no implementation as far as the base class is concerned. The classes derived from a base class having a pure virtual function have to define such a function or redeclare it as a pure virtual function. It must be noted that, a class containing pure virtual functions cannot be used to define any objects of its own and hence such classes are called pure abstract classes or simply abstract classes. whereas all other classes without pure virtual functions and which are instantiated are called as concrete classes.

A pure virtual function is an unfinished placeholder that the derived class is expected to complete. The following are the properties of pure virtual functions:

* A pure virtual function has no implementation in the base class hence, a class with pure virtual function cannot be instantiated.

* It acts as an empty bucket (virtual function is a partially filled bucket) that the derived class is supposed to fill.
* A pure virtual member function can be invoked by its derived class.

Syntax of Pure Virtual Function

```
class Mydass
{
    public
    . . . . . .
    . . . . . .
    virtual Returntype FunctionName (arguments) = 0;
    . . . . . .
    . . . . . .
};
```

keyword

Null function body

Rules for Virtual Functions:

* When a virtual function in a base class is created, there must be definition of the virtual function in the base class even if base class version of the function is never

actually called. However pure virtual functions are exceptions.

* They cannot be static members.
* They can be a friend function to another class.
* They are accessed using object pointers.
* A base pointer can serve as a pointer to a derived object since it is type-compatible whereas a derived object pointer variable cannot serve as a pointer to base objects.
* Its prototype in a base class and derived class must be identical for the ~~future~~ virtual function to work properly.
* The class cannot have virtual constructors, but can contain virtual destructor. In fact, virtual destructors are essential to the solutions of some problems. It is also possible to have virtual operator overloading.
* Most importantly, to realize the potential benefits of virtual functions supporting runtime polymorphism, they should be declared in the public section of a class.