Pointers, Virtual Functions Polymorphism and workingwith files

Early Binding or Compile Time Polymorphism

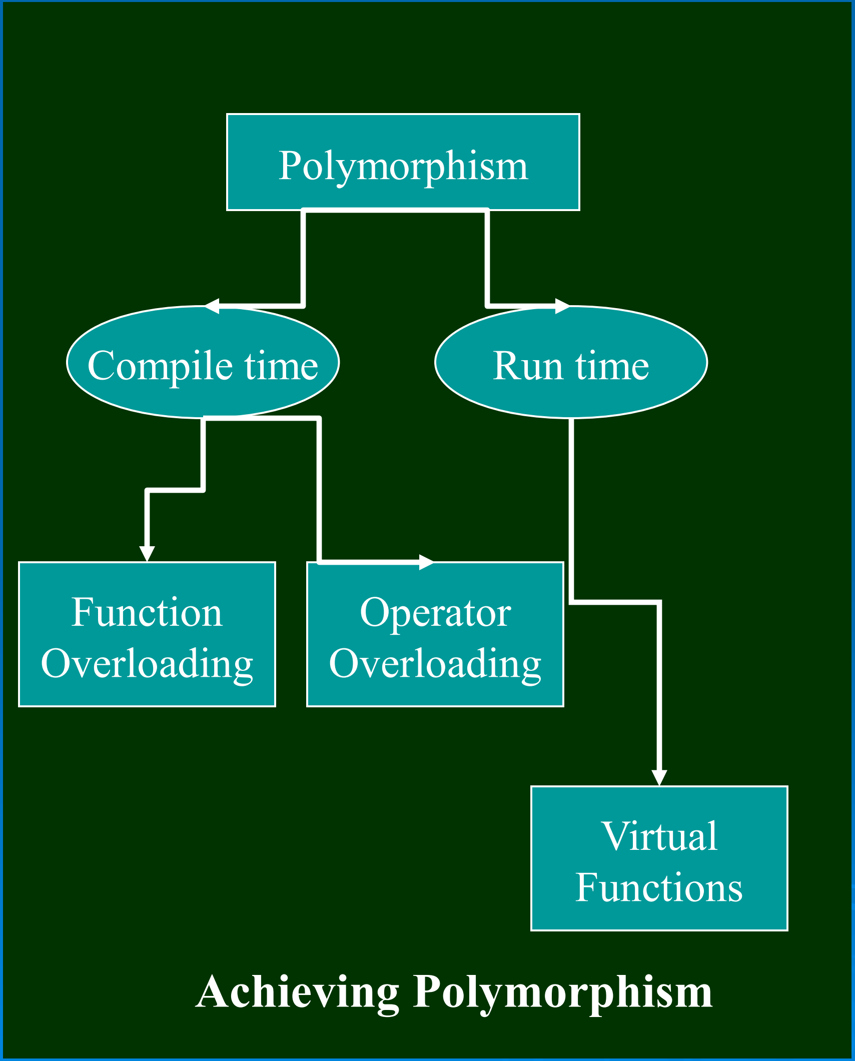
- The concept of polymorphism is implemented using overloaded functions and operators.
- The overloaded member functions are selected for invoking by matching arguments, both type and numbers.
- This information is known to the compiler at the compile time and, therefore, compiler is able to select the appropriate function for a particular call at the compile time itself.
- This is called early binding or static binding or static linking.

Late Binding or Run Time Polymorphism

```
class A
{
      int x;
   public:
      void show( ) {……}
};
class B : public A
{
      int y;
   public:
      void show( ) {……}
};
```

- Since the prototype of show( ) is the same in both the places, the function is not overloaded and therefore static binding does not apply.
- The class resolution operator is used to specify the class while invoking the functions with the derived class.
- Appropriate member function is selected while the program is running.

Late Binding or Run Time Polymorphism



Achieving Polymorphism

- The appropriate version of function will be invoked at runtime.
- Since the function is linked with a particular class much later after the compilation, this process is termed as late binding.
- This also known as dynamic binding, since the selection of appropriate function is done dynamically at runtime.

Pointer To Objects

- item * it_ptr ; *where item is a class and it_ptr is a pointer of type item.*

- Object pointers are useful in creats objects at run time.

- An object pointer can be used to access the public members of an object.

Pointer To Objects

continue…

```
class item
{
      int code;
      float  price;
   public:
      void getdata( int a, float b )
         { code =a; price = b;}
      void show( void )
      { cout << “Code :” << code<<endl
         << “Price :” << price << endl; }
};
item x;
item *ptr = &x;
```

- We can refer to the member functions of item in two ways:
  - Using dot operator and object.
    - x.getdata(100,75.50);
    - x.show( );
  - Using arrow operator and object pointer.
    - ptr -> getdata(100, 75.50);
    - ptr -> show( );
  - Since *ptr is an alias of x
    - (*ptr).show( );

Pointer To Objects

continue…

- We can also create the objects using pointers and new operator as:
  - item * ptr = new item ;
  - This statement allocates enough memory for the data members in the object structure and assigns the address of the memory space to ptr.
- We can also create an array of objects using pointers
  - item *ptr = new item[10];
  - Creates memory space for an array of 10 objects of item.

- Study question about pointers to objects and array of pointers to objects
- Page No : 225 and 227

this Pointer

- C++ uses a unique keyword called **this** to represent an object that invokes a member function.
- This unique pointer is automatically passed to a member function when it is called.
- The pointer **this** acts as an implicit argument to all the member functions.
- One important application of the pointer **this** is to return the object it points to.

```
 return *this;
```

- Refer program to implement this pointer in page No. 230-231

Pointer to Derived Classes

- We can use pointers not only to the base objects but also to the objects of derived classes.
- Pointers to objects of a base class are type-compatible with pointers to objects of a derived class.
- Therefore, single pointer variable can be made to point to objects belonging to different classes.

Pointer to Derived Classes

continue…

- If B is a base class and D is a derived class from B, then a pointer declared as a pointer to B can also be a pointer to D.

```
B * bptr ;
B b;
D d;
bptr = &b;
```

We can also make bptr = &d;

Pointer to Derived Classes

continue…

- Using bptr, we can access only those members which are inherited from B and not the members that originally belong to D.
- In case a member of D has the same name as one of the members of B, then any reference to that member by bptr will always access the base class member.

- Refer program to implement pointer to derived objects in Pg. 232-233

Virtual Functions

- Polymorphism refers to the property by which objects belonging to different classes are able to respond to the same message, but in different forms.
- An essential requirement of polymorphism is therefore the ability to refer to objects without any regard to their classes.
- This necessitates the use of a single pointer variable to refer to the objects of different classes.

Virtual Functions

continue…

- We use pointer to base class to refer to all the derived objects.
- When we use the same function name in both the base and derived classes, the function in base class is declared as virtual using the keyword **virtual** preceding its normal declaration.
- When a function made virtual, C++ determines which function to use at run time based on the type of object pointed to by the base pointer, rather than the type of the pointer.

Virtual Functions

continue…

- One important point to remember is that, we must access virtual functions through the use of a pointer declared as a pointer to the base class.
- Run time polymorphism is achieved only when a virtual function is accessed through a pointer to the base class.

Rules for Virtual Functions

- The virtual functions must be members of some class.
- They cannot be static members.
- They are accessed by using object pointers.
- A virtual function can be a friend of another class.
- A virtual function in a base class must be defined, even though it may not be used.

- Refer program to implement pure virtual function and run time polymorphism in page no. 234-235 and 236-238

Pure Virtual Functions

- A pure virtual function is a function declared in a base class that has no definition relative to the base class.
- A do-nothing function may be defined as follows:

  virtual void display( ) = 0;
- A class containing pure virtual functions cannot be used to declare any objects of its own. – abstract classes.

Pure Virtual Functions

- The main objective of an abstract base class is to provide some traits to the derived classes and to create a base pointer required for achieving run time polymorphism.