Unit-5-Polymorphism, Virtual Function and Working with files

1 Explain pointer to objects with Example.

- Pointer is one of the key aspects of C++ language.
- Pointer refers to another data variable by its memory address.
- A pointer can point any object similar like as normal variable.
- variable that holds an address value is called a pointer variable or simply pointer.
- Pointer can point to objects as well as to simple data types and arrays.
- Sometimes we don't know, at the time that we write the program, how many objects we want to create
- When this is the case we can use new to create objects while the program is running.
- Create pointer object by following syntax:
  classname * pointerobjectname;
- Intialize with address of another object by following syntax:
  pointername=&objectname;

Example :

```
#include <iostream>
using namespace std;
class trial
{
        int a;
        public:
                void init(int x)
                {
                a=x;
                }
                void disp()
                {
                cout<<"\nvalue of a="<<a;
                }
};
int main()
{
        trial t;
        trial *ptr_t;
        ptr_t=&t;
        t.init(10);
        t.disp();
        ptr_t->init(20);
        ptr_t->disp();
        t.disp();
}
```

Output :

```
value of a=10
value of a=20
value of a=20
```

Example 2 :

```
#include <iostream>
```

```
using namespace std;
class student
{
private:
        int rollno;
        string name;
public:
        void get()
        {
                cout<<"enter roll no:";
                cin>>rollno;
                cout<<"enter  name:";
                cin>>name;
        }
        void print()
        {
                cout<<"roll no is:"<<rollno<<endl;
                cout<<"name is:"<<name<<endl;
        }
};
int main ()
{
        student *ps=new student;
        (*ps).get();
        (*ps).print();
        delete ps;
        getch();
        return 0;
}
```

Output :

```
enter roll no: 01
enter  name: abc
roll no is: 1
name is : abc
```

2 Explain 'this' pointer with example.

- C++ uses a unique keyword called this to represent an object that invokes a member function
- this is a pointer that points to the object for which this function was called.
- Every object has a special pointer "this" which points to the object itself. This pointer is accessible to all members of the class but not to any static members of the class.
- Can be used to find the address of the object in which the function is a member. Presence of this pointer is not included in the sizeof calculations.

Example :

```
#include<iostream>
using namespace std;
class MyClass
{
      int data;
      public:
```

```
    void SetData(int data)
        {
                this->data =data;
        }
        int GetData()
        {
                return data;
        }
};
int main()
{
   MyClass a;
   a.SetData(100);
   cout << a.GetData() << endl;
   return 0;
}
```

Output :

100

3 Explain pointer to derived classes.

- We can use pointers not only to the base objects but also to the objects of derived classes.
- Pointers of objects of a base class are type compatible pointers to objects of a derived class. Therefore, a single pointer variable can be made to point to objects belonging to different classes.
- For example if 'B' is a base class , 'D' is derived class from 'B'. Then a pointer declared as a pointer to 'B' can also a pointer to 'D'.
- However there is a problem in using p to access the public members of the derived class D.
- Using p 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 p will always access the base class member.
- Although C++ permits a base pointer to point to any object derived from that base, the pointer cannot be directly used to access all the members of the derived class.
- We may have to use another pointer declared as pointer to derived type.

Example :

```
#include <iostream>
using namespace std;
class B {
 int x;
public:
 void setx(int i) {
    x = i;
 }
 int getx() {
    return x;
 }
};
```

```
class D : public B {
  int y;
public:
  void sety(int i) {
     y = i;
  }
  int gety() {
     return y;
  }
};
int main()
{
  B *p;
  B base;
  D derived;
  p = &base;
  p->setx(10);
  cout << "Base object x: " << p->getx() << '\n';
  p = &derived;
  p->setx(99);
  derived.sety(88);
  cout << "Derived object x: " << p->getx() << '\n';
  cout << "Derived object y: " << derived.gety() << '\n';
  return 0;
}
```

Output :

Base object x:10

Derived object x:99

Derived object y:88

4 Explain virtual function with example.

- It is a run time polymorphism.
- Base class and derived class have same function name and base class pointer is assigned address of derived class object then also pointer will execute base class function.
- To execute function of derived class, we have to declare function of base class as virtual.
- To declare virtual function just uses keyword virtual preceding its normal function declaration.
- After making virtual function, the compiler will determine which function to execute at run time on the basis of assigned address to pointer of base class.
- Rules for virtual function
  1. The virtual functions must be member of any class.
  2. They cannot be static members.
  3. They are accessed by using object pointers.
  4. A virtual function can be a friend of another class.
  5. A virtual function in a base class must be defined, even though it may not be used.
  6. If two functions with the same name have different prototypes, C++ considers them as overloaded functions, and the virtual function mechanism is ignored.

7. We cannot have virtual constructors, but we can have virtual destructors.
8. The derived class pointer cannot point to the object of base class.
9. When a base pointer points to a derived class, then also it is incremented or decremented only relative to its base type. Therefore we should not use this method to move the pointer to the next object.
10. If a virtual function is defined in base class, it need not be necessarily redefined in the derived class. In such cases, call will invoke the base class.

- We can better understand virtual function by following example:

Example :

```
#include<iostream>
using namespace std;
class base
{
	public:
		void display()
		{
			cout<<"Display base"<<endl;
		}
		virtual void show()
		{
			cout<<"Show base"<<endl;
		}
};
class Derived : public base
{
	public:
		void display()
		{
			cout<<"Dispaly Derived"<<endl;
		}
		void show()
		{
			cout<<"Show Derived"<<endl;
		}
};
int main()
{
	base b;
	Derived d;
	base *p;
	cout<<"Base Class Called"<<endl;
	p = &b;
	p->display();
	p->show();
	cout<<"Virtual Class Called"<<endl;
	p = &d;
	p->display();
	p->show();
	return 0;
}
```

Output :

Base Class Called
Display base
Show base
Virtual Class Called
Display base
Show Derived

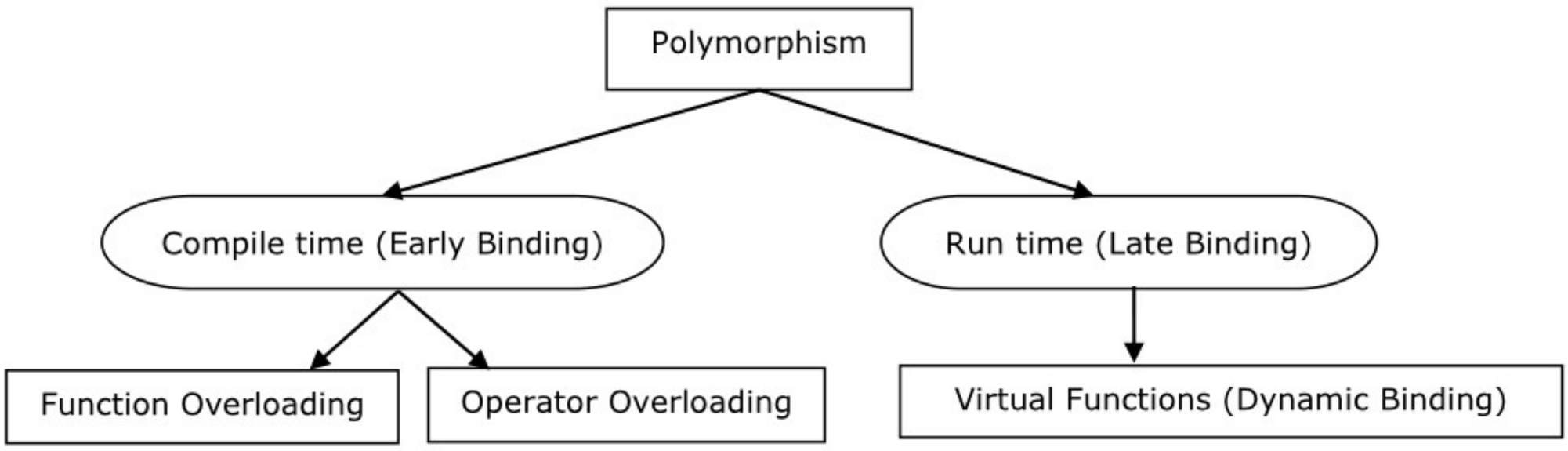
5 Explain Pointer to Virtual Functions with example.

- When we are creating a pointer, and that created pointer is pointing to a virtual function then it is known as the pointer to virtual functions.
- **NOTE: Example is same as in the above topic**

6 Write a short note on Polymorphism.

- Polymorphism means the ability to take more than one form.
- It allows a single name to be used for more than one related purpose.
- It means ability of operators and functions to act differently in different situations.

Different types of polymorphism are



Compile time:

- Compile time polymorphism is function and operator overloading.

Function Overloading:

- Function overloading is the practice of declaring the same function with different signatures.
- The same function name will be used with different number of parameters and parameters of different type.

Operator Overloading:

- Operator overloading is the ability to tell the compiler how to perform a certain operation based on its corresponding operator's data type.
- Like **+** performs addition of two integer numbers, concatenation of two string variables and works totally different when used with objects of class.

Dynamic Binding (Late Binding):

- Dynamic binding is the linking of a routine or object at runtime based on the conditions at that moment.
- It means that the code associated with a given procedure call is not known until the time of the call.
- At run-time, the code matching the object under current reference will be called.

Virtual Function:

- Virtual function is a member function of a class, whose functionality can be over-ridden in its derived classes.
- The whole function body can be replaced with a new set of implementation in the derived class.
- It is declared as virtual in the base class using the virtual keyword.