Polymorphism

Polymorphism:

The word polymorphism means having many forms.In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. A real-life example of polymorphism, a person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person posses different behavior in different situations.

This is called polymorphism. Polymorphism is considered as one of the important
features of Object Oriented Programming.
In C++ polymorphism is mainly divided into two types:

1) Compile time Polymorphism
2) Runtime Polymorphism

i) Compile time polymorphism:

* The overloaded functions are invoked by matching the type and number of arguments.

* This information is available at the compile time and, therefore, compiler selects the appropriate function at the compile time.

* It is achieved by function overloading and operator overloading which is also known as static binding or early binding.

* Now, let's consider the case where function name and prototype is same.

Function overloading

Function overloading is a **feature of object oriented programming where two or more functions can have the same name but different parameters**. When a function name is overloaded with different jobs it is called Function Overloading.

Example:

```
#include <bits/stdc++.h>
using namespace std;
class Answer
{
   public:

   void Students_answer(int x)
   {
      cout << "John answer of x value
is " << x << endl;
   }
```

```
   void Students_answer(double x)
   {
      cout << "Alex answer of x value
is " << x << endl;
   }

};

int main() {

   Answer value;
   value.Students_answer(7);
   value.Students_answer(9.132);
   return 0;
}
```

Output:

```
John answer of x value is 7
Alex answer of x value is
9.132
```

Operators Overloading

Operator overloading is a compile-time polymorphism in which the operator is overloaded to provide the special meaning to the user-defined data type.

Operator overloading is used to overload or redefines most of the operators available in C++.

It is used to perform the operation on the user-defined data type. For example, C++ provides the ability to add the variables of the user-defined data type that is applied to the built-in data types.

The advantage of Operators overloading is to perform different operations on the same operand.

Example:

```
#include<bits/stdc++.h>
using namespace std;
class math{
    private:

     int x = 10,y = 20;

    public:

      void operator ++();
};

void math :: operator++(){
        cout<<"value of x : "<<
x<<endl<<"value of y : "<<y<<endl;
      }
int main(){
    math k;
    ++k;
}
```

Output:

value of x : 10
value of y : 20

Run time polymorphism:

* Run time polymorphism is achieved when the object's method is invoked at
the run time instead of compile time.

* It is achieved by method overriding which is also known as dynamic binding or late binding.

Virtual function:

A virtual function is a member function which is declared within a base class and is re-defined (overridden) by a derived class.

When you refer to a derived class object using a pointer or a reference to the base class,
you can call a virtual function for that object and execute the derived class's version
of the function.

*) Virtual functions ensure that the correct function is called for an object,regardless of the type of reference (or pointer) used for function call.

*) They are mainly used to achieve Runtime polymorphism

*) Functions are declared with a virtual keyword in base class.

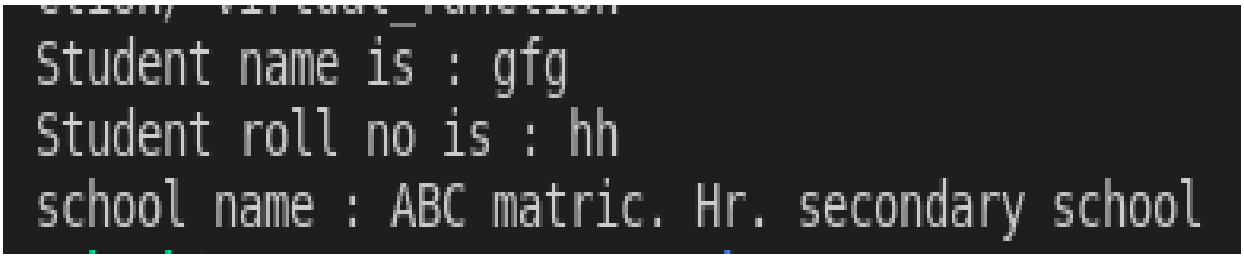
*) The resolving of function call is done at runtime.

Example:

```
#include<bits/stdc++.h>
#include<string.h>
using namespace std;
class Student
{
    public:
    string name;
    int roll_no;
    string school_name = "ABC matric. Hr.
secondary school";

    public:

    void scl_name(){
        cout<<"School name :
"<<school_name<<endl;
    }

    virtual void print(){
```

```
        cout<<"student details";
    }
};

class student1 : public Student{
public:
    void print(){
        cout<<"Student name is : ";
        cin>>name;
        cout<<"Student roll no is : "<<" ";
        cin>>roll_no;
    }
};

int main(){
    Student *s;
    student1 s1;
    s = &s1;
    s->print();
    s->scl_name();
}
```

Output:


Student name is : gfg
Student roll no is : hh
school name : ABC matric. Hr. secondary school