Polymorphism

1. Compile time Polymorphism:-

1.1. Function Overloading is a feature in C++ where two or more functions can have the same name but different parameters.

It provides multiple definitions of the function by changing signature i.e changing number of parameters, changing datatype of parameters, return type doesn't play any role.

```
1. void print(int i)
2. {
3.      cout<<"Here is int "<<i<<endl;
4. }
5.
6. void print(float i)
7. {
8.      cout<<"Here is Float "<<i<<endl;
9. }
10.
11.  int main() {
12.     print(10);
13.     print(10.1);
14.     return 0;
15.  }
```

Functions that cannot be overloaded:-

1. Function declarations that differ only in the return type.
2. Parameter declarations that differ only in a pointer * versus an array [] are equivalent. That is, the array declaration is adjusted to become a pointer declaration.

3. Parameter declarations that differ only in that one is a function type and the other is a pointer to the same function type are equivalent.
4. Parameter declarations that differ only in the presence or absence of const are equivalent.
5. Two parameter declarations that differ only in their default arguments are equivalent.

1.2. Operator overloading in C++ have the ability to provide special meaning to the operator.

Syntax: returnType operator symbol(arguments)

```
1. class Complex {
2. private:
3.     int real, imag;
4. public:
5.     Complex(int r = 0, int i =0)  {real = r;   imag = i;}
6.
7.     Complex operator + (Complex const &obj) {
8.          Complex res;
9.          res.real = real + obj.real;
10.           res.imag = imag + obj.imag;
11.           return res;
12.      }
13.      void print() { cout << real << " + i" << imag <<
   endl; }
14.  };
15.
16.  int main()
17.  {
18.      Complex c1(10, 5), c2(2, 4);
19.      Complex c3 = c1 + c2;
```

```
20.      c3.print();
21.  }
```

Two operators = and & are already overloaded by default in C++.

Almost all operators can be overloaded except a few. Following is the list of operators that cannot be overloaded.

```
1.)  . (member selection)
2.)  :: (scope resolution)
3.)  ?: (ternary operator)
4.)  sizeof
```

Operator Overloading can't change the precedence and associativity of operators do not change.

New operators can't be created like &|.

The overloading of operators && and || lose short-circuit evaluation.

2. Runtime/Dynamic Polymorphism:-

Method Overriding:- It is the redefinition of base class function in its derived class, with same return type and same parameters.

```
1. class Animal {
2.    public:
3. void eat(){
4. cout<<"Eating...";
5.    }
6. };
7. class Dog: public Animal
8. {
9. public:
10.  void eat()
11.     {
12.         cout<<"Eating bread...";
```

```
    }
};
int main(void) {
    Dog d = Dog();
    d.eat();
    return 0;
}
```

Virtual Function

A virtual Function is a member function which is declared with a 'virtual' keyword in the base class and redeclared (overridden) in a derived class. When you refer to an object of derived class using a pointer to a base class , you can call a virtual function of that object and execute the derived class's version of the function.

->They are used to achieve Runtime Polymorphism.
->Virtual Function cannot be static and also cannot be a friend function of another class.

Compile -time (Early Binding) Vs Run-time (Late Binding)

```
class base {
public:
    virtual void print()
    {
        cout << "print base class" << endl;
    }

    void show()
    {
        cout << "show base class" << endl;
    }
```

```
12.  };
13.
14.  class derived : public base {
15.  public:
16.      void print()
17.      {
18.          cout << "print derived class" << endl;
19.      }
20.
21.      void show()
22.      {
23.          cout << "show derived class" << endl;
24.      }
25.  };
26.
27.  int main()
28.  {
29.      base* bptr;
30.      derived d;
31.      bptr = &d;
32.
33.      // virtual function, binded at runtime
34.      bptr->print();
35.
36.      // Non-virtual function, binded at compile time
37.      bptr->show();
```

Output:-

Derived print //Late Binding
Base show fun //Early Binding

As during compiler time bptr behaviour is judged on the base's of which class it belongs, so bptr represents base class.

If the function is not virtual then it will allow binding at compile time and print fun of base class will get binded with bptr representing base class.

But at run time bptr points to the object at class derived, so it will bind the function of derived at run time.

Override

Function overriding is redefinition of base class function in its derived class with same signature i.e return type and parameters

It helps to check if :

- There is a method with the same name in the parent class.
- The method in the parent class is declared as "virtual" which means it was intended to be rewritten.
- The method in the parent class has the same signature as the method in the subclass.

Pure virtual Function and Abstract Class

Sometimes implementation of all functions cannot be provided in a base class because we don't know the implementation. Such a class is called an abstract class.

For example, let Shape be a base class. We cannot provide implementation of function draw() in Shape, but we know every derived class must have implementation of draw().

```
//Abstract Class
Class test
{
```

```
    Public:
    Virtual Void fun()=0;        //pure virtual function.
}
```

1. A class is Abstract if it has at least one pure virtual function.
   we cannot declare objects of abstract class. It will show errors.
2. We can have pointers or references of abstract classes.
3. We can access the other function except virtual by object of its derived class.
4. If we don't override the pure virtual function in the derived class then it becomes abstract.
5. An abstract class can have constructor.

Working of virtual function(Vtable & Vptr)

If a class contains virtual function then compiler itself does two things:

1. A virtual Pointer (Vptr) is created every time obj is created for the class which contains virtual function.
2. Irrespective of whether an object is created or not, a static array of pointer called Vtable where each cell points to each virtual Function is created, in base class and derived class.