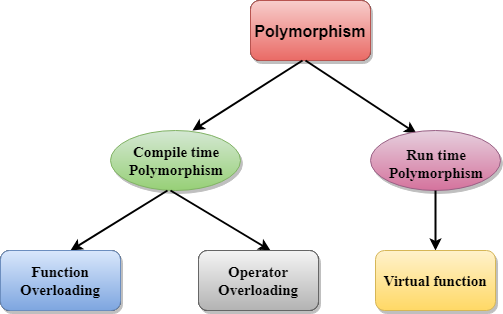
Polymorphism

* The term "*Polymorphism*" is the combination of "*poly*" + "*morphs*" which means many forms. It is a Greek word.
* There are two types of polymorphism in C++ -



* 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.

class A // base class declaration.

{

int a;

public:

void display ()

{

cout<< "Class A ";

}

};

class B : public A // derived class declaration.

{

int b;

public:

void display()

{

cout<<"Class B";

}

};

* 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.

#include<iostream.h>

class Animal

{

public:

void eat()

{

cout<<"Eating...”;

}

};

class Dog: public Animal

{

public:

void eat()

{

cout<<"Eating bread...";

}

};

void main()

{

Dog d = Dog();

d.eat();

}

* Runtime Polymorphism with Data Members -

Let's see an example where we are accessing the field by reference variable which refers to the instance of derived class.

#include <iostream>

class Animal

{

public:

string color = "Black";

};

class Dog: public Animal

{

public:

string color = "Grey";

};

void main()

{

Animal d= Dog();

cout<<d.color;

}

* Function Overriding -

If derived class defines same function as defined in its base class, it is known as function overriding in C++. It is used to achieve runtime polymorphism. It enables you to provide specific implementation of the function which is already provided by its base class.

#include<iostream.h>

class Animal

{

public:

void eat()

{

cout<<"Eating...";

}

};

class Dog: public Animal

{

public:

void eat()

{

cout<<"Eating bread...";

}

};

void main()

{

Dog d = Dog();

d.eat();

}

* Overloading –

If we create two or more members having the same name but different in number or type of parameter, it is known as C++ overloading. In C++, we can overload:

* + methods,
  + constructors, and
  + indexed properties

Types of overloading in C++ are -

* + Function overloading
  + Operator overloading

Function Overloading –

Function Overloading is defined as the process of having two or more function with the same name, but different in parameters is known as function overloading in C++.

In function overloading, the function is redefined by using either different types of arguments or a different number of arguments.

* The advantage of Function overloading is that it increases the readability of the program.

program of function overloading when number of arguments vary

#include<iostream.h>

class Cal {

public:

static int add(int a, int b)

{

return a + b;

}

static int add(int a, int b, int c)

{

return a + b + c;

}

};

void main()

{

Cal C;

cout<<C.add(10, 20)<<endl;

cout<<C.add(12, 20, 23);

}

Program of function overloading with different types of arguments

* C++ 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.

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

* Operator that cannot be overloaded are as follows -
  + Scope operator (::)
  + Sizeof
  + member selector(.)
  + member pointer selector(\*)
  + ternary operator(?:)
* Rules for Operator Overloading -
  + Existing operators can only be overloaded, but the new operators cannot be overloaded.
  + The overloaded operator contains atleast one operand of the user-defined data type.
  + We cannot use friend function to overload certain operators.
  + When unary operators are overloaded through a member function take no explicit arguments, but, if they are overloaded by a friend function, takes one argument.
  + When binary operators are overloaded through a member function takes one explicit argument, and if they are overloaded through a friend function takes two explicit arguments.

program to overload the unary operator ++

#include<iostream.h>

class Test

{

private:

int num;

public:

Test(): num(8)

{}

void operator ++()

{

num = num+2;

}

void Print()

{

cout<<"The Count is: "<<num;

}

};

void main()

{

Test tt;

++tt;

tt.Print();

}

program to overload the binary operators

#include<iostream.h>

class A

{

int x;

public:

A(){}

A(int i)

{

x=i;

}

void operator+(A);

void display();

};

void A :: operator+(A a)

{

int m = x+a.x;

cout<<"The result of the addition of two objects is : "<<m;

}

void main()

{

A a1(5);

A a2(4);

a1+a2;

}