Polymorphism

* + - Ability to take more than one form
    - Single name many forms

Two Types

* + - Compile Time Polymorphism –
    - (Function Overloading, Operator Overloading)
      1. Which function is to be called is to be decided at the time of compilation.

Function overloading- We can define multiple functions with the same name but having different no. and type of arguments.

Add(int x, int y)

Add(Complex c1, complex c2)

Add(char a, char b)

Add(int x, int y, int z)

Operator Overloading:

+, - , \*, /

These operators can work with integer and float values.

2+2

2-2

2\*2

2/2

These are already defined by complier and we are using them directly.

+(int,int)

/(float,float)

+(float,float)

Complex

{

Complex + (Complex C2)

}

C3=c1.add(c2);

C3=c1+c2;

C3=c1-c2; subtract

C3=c1\*c2; Multiplication

C3=c1/c2;

Matrix

{

Matrix \* (matrix m2)

}

Matrix m1,m2,m3;

M3=m1.multiplication(m3);

M3=m1\*m2;

A+b+c;

2+3+4

5+4

Purpose of operator Overloading:

* + - Provide additional meaning and usage to already existing operators
    - Type conversions also- Basic to basic conversion
      1. Basic to class
      2. Class to basic
      3. Class to class
    - Run Time Polymorphism
      1. Virtual Functions

C++ Overloadable Operators

1. Arithmetic operators - +,-,\*,/
2. Bit-Wise- &,I,~,^
3. Logical- &&, II,!
4. Relational - <,>,==,!=,<=,>=
5. Assignment or initialization =
6. Arithmetic Assignment +=,-=,\*=,/=,%=,&=,
7. Shift <<.>>, <<=,>>=
8. Unary ++,--
9. Subscripting [ ]
10. Function Call ( )
11. Dereferencing Operator ->
12. Unary sign prefix operator -
13. Allocate and free operator – new and delete

Operators cannot be overloaded:

1. Class member access operator - ., .\*
2. Scope resolution operator - ::
3. Size of operator – sizeof
4. Conditional operator - ?:

#include<iostream>

Class index

{

Private:

Int value;

Public:

Index()

{ value = 0;}

Int getindex()

{ return value;}

Void operator ++()

{

Value=value+1;

}

};

Void main()

{

Index idx1,idx2,idx3; // Idx1.value= 0, idx2.value=0

Cout<<idx1.getindex(); 0

Cout<<idx2.getindex(); 0

Idx1.nextindex(); // idx1.value = 0+1= 1 // ++idx1

Idx2.nextindex(); // idx2.value= 1 // idx2++

Idx2.nextindex();// idx2.value = 2

Cout<<idx1.getindex();

Cout<<idx2.getindex();

}

Returntype operator operatorsymbol ([arg1, [arg2]])

{

}

Return type : Primitive, void or user defined

Operator: keyword

Operatorsymbol: operator to be overloaded

**Arguments to** operator function

**Unary operator – One Operand**

**++object; // Object is used to call ++ operator function if operator function is defined as member function.**

**Int a;**

**++a;**

**++idx1; // idx1.++();**

**++ - operator member function , which is called by idx1 object.**

**Unary operator function as a member function**

**Returntype operator operatorsymbol()**

**{**

**} // idx1++;**

**Unary operator function as a friend function**

**Returntype operator operatorsymbol(arg1)**

**{ } // arg1 is object //idx1++ // idx1 is passed as an argument to ++ operator function**

**Returntype operator opsymbol ( arg1, arg2)**

**{**

**}**

**Unary operator –**

1. **Member function - no argument to be passed to operator function**
2. **Friend Function – single argument , which is of type object of the class to which it is declared as a friend.**

**Binary Operator – two operands ( o1+o2)**

1. **Member Function- one argument**
2. **Friend Function – two arguments, both the objects on which operator is to be applied is passed an argument.**

**C1+C2 // +(c1,c2)**

**// c1.+(c2)**

**Friend function - Void operator + (complex c1, complex c2)**

**Member - Void operator + (complex c2)**

**Friend Function**

**Add (complex c1, complex c2)**

**Add(c1,c2);**

**C1.add(c2)**