Operator Overloading

PART-5 (25)

(I) It is mechanism to give special meaning to an operator of language is known as operator overloading.

(II) For eg → + For basic data type int, float

c = a+b ⇒ add^n of int type (a, b, c are basic data typ)

if a, b, c are object of class then compiler gives errors

(III) so this problem can be solved using operators overloading.

C = a+b (a, b, c are object of class)

(IV) Defining operator overloading.

return_type Class_Name :: Operator op(arg_list)

{

function body

}

op is operator to be overloaded. This is also known as operator function.

(V) It can be unary operator or binary operator

↓ ↓

which apply operation on single operand — which apply operation on two operand

a++, -a, +a, a-- — a+b, a-b, a>b

(VI) bt operator fn must either be member fn or friend fn.

if It is member fn then no argument for unary operator

only one argu — binary oper

because object is used to invoke the member fn.

if It is friend fn then only one argu — unary oper

two argu — binary operator

for eg ① Member fn ← { void operator -() // overload unary minus
void operator ++() // ——— increament

Friend fn ← { friend void operator -(complex)
friend complex operator ++(complex)

```
void main()
{ Class_Name Obj ;
}
```

Member fn -Obj; obj ++;

Friend fn operator -(x);
operator ++(x);

Overloading unary operator → which takes only one operand. for eg -

① +, ② -, ③ ++, ④ --

Return type of operator

for eg

```
class complex
{
 int r, i;
 Public: void get data()
        {
        cin>>r; cin>>i;
        }
        void showdata()
        {
        cout<< r <<"+i"<<i;
        }
        void operator ++()
        {
        ++r;
        ++i;
        }
};
void main()
{
 complex C1; C1++;
 C1.getdata();
 C1.showdata();
}
```

```
complex operator ++()
{ complex C;
 C.r = ++r;
 C.i = ++i;
 return(C);
}

C++;
C2 = C1++;
C2.showdata();
```

Overloading of Binary operator → which use two operand.

Such as (i) arithmatic (+, −, *, /)
(ii) comparison
(iii) assignment

using friend

(i) addn of Two Complex No.

```
Complex operator +(Complex C)
{
    Complex C4;
    C4.real = real + C.real;
    C4.img = img + C.img;
    return(C4);
}
void main()
{
    Complex C1(5,4);
           C2(6,5);
    Complex C3;
    C3 = C1 + C2;        // binary + operator
        C1.operator +(C2);
}
```

```
friend Complex operator +(Complex, Complex);
then
Complex operator +(Complex a, Complex b)
{
Return((a.x+b.x),(a.y+b.y));
}
```

C3 = operator +(C1, C2);

(ii) addn of two string i.e concatenation of string

```
class string
{
    char Str[80];
    Public: String()
            {
            Strcpy(Str," ");
            }
            String(char S[])
            {
             strcpy(Str, S);
            }
```

```
void display()
{
cout << str << endl;
}
String operator +(String s4)
  {
  if((strlen(str) + strlen(s4.str))<80)
  {
   String temp
   strcpy(temp.str, str);
   strcat(temp.str, s4.str);
   return(temp);
   }
   else
   cout<<"String overlflow";
  }};
void main()
  {
   String s1(Akash);
   String s2(kumar); ⇒ String s2 = "kumar";
   String s3;
    s3 = s1+s2; ⇒ s1.operator +(s2)
    s3.display();
    getch();
  }
```

Overloading of comparison operator

① <, >, ==, <=, >=

② If returns true ~~and~~ false then we have to use enumerated data type.

Prog **Comparison of two No.**

```
enum boolean {false, true};
class sample
{
int d;
Public: sample()
        {
          d=0;
        }
        sample (int a )
        {
          d=a;
        }
        void show()
        {
        cout<<d;
        }
        boolean operator <(sample s) // comparison of two operand then 2nd is member fn so only one operand is passed.
        {
          if (d< s.d)
            {
              return (true);
            }
          else
            {
              return (false);
            }
  };
void main()
  {
  sample S1;
  Sample S2(4);
  Sample S3(6);
  if (S1< S2)
    cout<<" S1 is less than S2";
  else
  cout<<" S1 is not less than S2";
cout<< S1.show;
cout<< S2. show; }
cout<< S3. show;
getch();
  }
```

Overloading of assignment operator →

① =, +=, -=, *=, /= etc

```
class sample
{
  int d;
Public: sample()
        {
          d=1;
        }
        sample(int a)
        {
          d=a;
        }
void    operator=(sample S1)
        {
          d= S1.d;          // d = 10;  S3 value
        }
void    operator +=(sample S2)
        {
          d= d + S2.d;      // S5 = 6 + 10
                            // d = 16   S5 value
        }
};
void main()
        {
        sample S3;
        sample S4(10);
        sample S5(6);
        S3 = S4;            // = operator overload
        S5+= S4;            // += operator overload
```

(I) ?
(II) ::
(III) sizeof()
(IV) . , .*
(V)

Notes (I) It is called Compile time polymorphism

(II) It is used to add two user defined data types such as object just as basic data type

(III) operator function must be either friend function or member function (non-static) & overloading operator must have at least one operand that is of user defined type.

(IV) Sum of two complex No.

```
C = Sum(A, B); // Functional
C = A + B;     // operator overloading
```