Operator Overloading

Definition:

- ➢ The operator overloading feature of C++ is one of the methods of realizing polymorphism.
- ➢ C++ has the ability to provide the operators with a special meaning to an operator is known as operator overloading.
- ➢ Different behavior at different instances.

Syntax:

```
returntype operator operatorsymbol (arglist)
{
        //operator function
}
```

The following Operators cannot be overloaded

- ➢ Size of operator(sizeof)
- ➢ Scope resolution operator(::)
- ➢ Conditional operator(?:)
- ➢ Class member access operator(. , .* , ->*)
- ➢ Pointer to member declarator(::*)

The following Operators cannot be overloaded when a friend function is used

- ➢ Assignment operator(=)
- ➢ Function call operator (( ))
- ➢ Subscripting operator([ ])
- ➢ Class member access operator(->)

Operator overloading can be represented as following ways

- ✓ Unary operator overloading
- ✓ Binary operator overloading

Example program for unary operator overloading

```
#include<iostream.h>
#include<conio.h>
class sample
{       private:
                        int x,y,z;
        public:
                        void getdata()
                        {       cin>>x>>y>>z;
                        }
                        void operator +()               // operator overloading
                        {
                                x=-x;y=-y;z=-z;
                        }

                        void print()
                        {
                                cout<<x<<endl<<y<<endl<<z;
                        }
};

void main()
{       sample ob;
        ob.getdata();
        ob.print();
        +ob;                            // activates operator + () function
        ob.print();
}
```

Example program for binary operator overloading

```
#include<iostream.h>
#include<conio.h>
class complex
{
        private:
                int real,imag;
        public:
                complex()
                {
                     real=imag=0;
                }
                complex(int x,int y)
                {
                  real=x;
                  imag=y;
                }
                void display()
                {
                        cout<<real<<"+j"<<imag<<endl;
                }
```

```
                    complex operator +(complex c)
                    {
                      complex temp;
                      temp.real=real+c.real;
                      temp.imag=imag+c.imag;
                      return(temp);
                    }
};
void main()
{
 clrscr();
 complex c1,c2,c3;
 c1=complex(2,1);
 c2=complex(3,4);
 c1.display();
 c2.display();
 c3=c1+c2;
 c3.display();
 getch();
}
```

Overloading through Friend Function

Friend functions may be used in the place of member functions for overloading a binary operator, the only difference being that a friend function requires two arguments to be explicitly passed to it, while a member function requires only one.

Example program for operator overloading through Friend function

```
#include<iostream.h>
#include<conio.h>
class complex
{

          private:
                    int real,imag;
          public:
                    complex()
                    {
                         real=imag=0;
                    }
                    complex(int x,int y)
                    {
                      real=x;
                      imag=y;

                    }
                    void display()
                    {
                              cout<<real<<"+j"<<imag<<endl;
                    }
                    friend complex operator +(complex c1,complex c2);
};
```

```
complex operator +(complex c1,complex c2)
{

  complex temp;
   temp.real=c1.real+c2.real;
   temp.imag=c1.imag+c2.imag;
   return(temp);
}
void main()
{
 clrscr();
 complex c1,c2,c3;
 c1=complex(2,1);
 c2=complex(3,4);
 c1.display();
 c2.display();
 c3=c1+c2;
 c3.display();
 getch();
}
```

<u>Overloading the assignment operator</u>

Assignment operator can be overloaded in following way,

<u>Example program for overloading the assignment operator</u>

```
#include<iostream.h>
#include<conio.h>
class complex
{
        private:
                int real,imag;
        public:
                complex()
                {
                       real=imag=0;
                }
                complex(int x,int y)
                {
                   real=x;
                   imag=y;
                }
                void display()
                {
                        cout<<real<<"+j"<<imag<<endl;
                }
                void operator +=(complex c)
                {
                   real=real+c.real;
                   imag=imag+c.imag;
                }
};
```

```
void main()
{

clrscr();
 complex c1,c2,c3;
 c1=complex(2,1);
 c2=complex(3,4);
 c1.display();
 c2.display();
 c3=c1;
 c3+=c2;
 c3.display();
 getch();
}
```

Type Conversion

Conversion between Objects and Basic types

Three types of data conversion exits they are

- Conversion from basic type to class type
- Conversion from class type to basic type
- Conversion from one class type to another class type.

Conversion from basic type to class type

The constructors used for the type conversion take a simple argument whose type is to be converted.

Example program for Conversion from basic type to class type

```
#include<iostream.h>
#include<conio.h>
class example
{
        private:
                int y;
        public:
                example()
                {
                        y=0;            }
                example(int a)          // x value is stored in a <====(ob.a)
                {
                        y=a*10;
                }
                void print()
                {
                        cout<<y;
                }
};
void main()
{
 clrscr();
 example ob;
 int x=15;
 ob=x;          //conversion of basic type to class type
 ob.print();
 getch();
}
```

Conversion from class type to basic type

- C++ allows us to define an overload casting operator that should be used to convert a class type data to a basic type.
- The general form of an overloaded casting operator function usually reffered to as a conversion function is

```
        operator typename
        {
                //function body
        }
```

Example program for Conversion from class type to basic type

```
#include<iostream.h>
#include<conio.h>
class example
{
        private:
                int a;
        public:
                void getdata()
                {
                        cin>>a;
                }
                operator int()
                {
                        int y;
                        y=a*100;
                        return(y);
                }
};
void main()
{
 clrscr();
 example ob;
 int x;
 ob.getdata();
 x=ob;          //conversion of class type to basic type
 cout<<x;
 getch();
}
```

Conversion from one class type to another class type

Conversions between objects of different classes can be carried out by either a constructor or a conversion function.

Example program for Conversion from one class type to another class type

```
#include<iostream.h>
#include<conio.h>
class example
{
        private:
                int length;
        public:
                void getdata()
                {
                        cin>>length;
                }
                void print()
                {
                        cout<<length;
                }
};
void main()
{
 clrscr();
 example ob1,ob2;
 ob1.getdata();
 ob2=ob1;        //conversion of one class type to another class type
 ob2.print();
 getch();
}
```