

Chapter 7

Operator Overloading

Operator overloading is one of the exiting features of C++ language. It is an important technique that increases the power of C++. One of the aim of C++ is to operate the variables of user defined data type in the same way as that of the variables of basic data type.

C++ allows us to add the variables of user defined data type with the same syntax i.e. applied to built in data type variables. This means that C++ has the ability to provide a special meaning to the operators for user defined data types.

The mechanism of giving a special meaning to an operator is known as operator overloading. It is not possible to overload all the operators provided by C++. The operators which we can not overloaded are

- 1) Scope resolution operator (::)
- 2) Conditional operator (?)
- 3) Class member access operator (dot operator)
- 4) Size of operator.

Defining Operator Overloading

To give an additional task to an operator we must specify one function in relation to the class. This function is called as Operator Function. The operator function describes the task to be performed by the operator.

The general form of an operator function is as follows.

1) Inside Class

```
Return type operator Op(Parameter List)
{
    Operator Function Statement;
    :
    :
}
```

2) Outside Class

```
Return type Class name :: operator Op(Parameter List)
{
    Operator Function Statement;
    :
    :
}
```

Where,

- 1) Return type specifies the value return by the function.
- 2) Op is the operator which we want to overloaded.

The Op is preceded by keyword “operator” i.e. the “operator op” is function name.

The process of overloading involves the following steps.

- 1) First create a class that defines the data type i.e. to be used in the overloading operator.
- 2) Declare the operator function operator op() in the public part of the class. It may be either member function or friend function.
- 3) Define the operator function to implement the required operations.

Overloaded operator function can be called by expression such as.

- 1) For Unary Operator

- a) Op Operand e.g. - x , - y
- b) Operand Op e.g. x - , y -

- 2) For Binary Operator

Operand 1 Op Operand 2 e.g. a+b

The operators are always of two types

1) Unary Operator

Unary operators are those which require only one operand for their operation.

e.g. ++, --, - are the unary operators.

2) Binary Operator

The operators which require two operands for their operation are known as binary operators.

e.g. +, -, /, *, =, <=, >=

Note 1)

If the operator function is the member function & the overloaded operator is unary the operator function does not require any argument.

Note 2)

If the operator function is the member function & the overloaded operator is binary then the operator function receives one argument.

Write a Program to Overload a Unary Operator minus (-) Using Member Function Inside the Class.

```
#include<iostream.h>
#include<conio.h>

class no
{
private:
    int a,b;
public:
    void getdata()
    {
        cout<<"\nEnter two no = ";
        cin>>a>>b;
    }
    void putdata()
    {
        cout<<"No 1 = "<<a<<endl
            <<"No 2 = "<<b<<endl;
    }
    void operator -()
    {
        a=-a;
        b=-b;
    }
};

main()
{
no x,y;
clrscr();
x.getdata();
x.putdata();
-x;
cout<<"After changing sign = "<<endl;
x.putdata();
y.getdata();
y.putdata();
-y;
cout<<"After changing sign = "<<endl;
y.putdata();
getch();
return(0);
}
```

Write a Program to Overload a Unary Operator minus (-) Using Member Function Outside the Class.

```
#include<iostream.h>
#include<conio.h>
```

```

class no
{
    private:
        int a,b;
    public:
        void getdata()
        {
            cout<<"\nEnter two no = ";
            cin>>a>>b;
        }
        void putdata()
        {
            cout<<"No 1 = "<<a<<endl
                <<"No 2 = "<<b<<endl;
        }
        void operator -();
};

void no::operator -()
{
    a=-a;
    b=-b;
}

main()
{
no x,y;
clrscr();
x.getdata();
x.putdata();
-x;
cout<<"After changing sign = "<<endl;
x.putdata();
y.getdata();
y.putdata();
-y;
cout<<"After changing sign = "<<endl;
y.putdata();
getch();
return(0);
}

```

Write a Program to Overload a Unary Operator ++ to Increase the values of data members of object by 1.

```

#include<iostream.h>
#include<conio.h>

class value
{

```

```

private:
    int a,b;
public:
    void getdata()
    {
        cout<<"Enter two nos = ";
        cin>>a>>b;
    }
    void putdata()
    {
        cout<<"No 1 = "<<a<<endl
            <<"No 2 = "<<b<<endl;
    }
    void operator++()
    {
        a=++a;
        b=++b;
    }
};

main()
{
value x,y;
clrscr();
x.getdata();
++x;
x.putdata();
y.getdata();
++y;
y.putdata();
getch();
return(0);
}

```

Write a Program to Overload a Unary Operator -- to Decrease the values of data members of object by 1.

```

#include<iostream.h>
#include<conio.h>

class value
{
private:
    int a,b;
public:
    void getdata()
    {
        cout<<"Enter two no = ";
        cin>>a>>b;
    }
    void putdata()

```

```

    {
        cout<<"No 1 = "<<a<<endl
            <<"No 2 = "<<b<<endl;
    }
    void operator--();
};

void value::operator--()
{
    a=--a;
    b=--b;
}

main()
{
value x,y;
clrscr();
x.getdata();
x--;
x.putdata();
y.getdata();
y--;
y.putdata();
getch();
return(0);
}

```

Write a Program to Overload a Unary Operator ++, -- to Increase & Decrease the values of data members of object by 1.

```

#include<iostream.h>
#include<conio.h>

class value
{
private:
    int a,b;
public:
    void getdata()
    {
        cout<<"Enter two no = ";
        cin>>a>>b;
    }
    void putdata()
    {
        cout<<"No 1 = "<<a<<endl
            <<"No 2 = "<<b<<endl;
    }
    void operator++();
    void operator--();
};

```

```

void value::operator++()
{
    a=++a;
    b=++b;
}
void value::operator--()
{
    a=--a;
    b=--b;
}
main()
{
value p,q,r,s;
clrscr();
p.getdata();
++p;
p.putdata();
q.getdata();
++q;
q.putdata();
r.getdata();
--r;
r.putdata();
s.getdata();
--s;
s.putdata();
getch();
return(0);
}

```

Write a Program to perform Addition of Complex no of two Object of Class & Overload a Binary Operator + using Members Function.

```

#include<iostream.h>
#include<conio.h>

class complex
{
private:
    int r,i;
public:
    void getdata()
    {
        cout<<"Enter real and imaginary part = ";
        cin>>r>>i;
    }
    void putdata()
    {
        cout<<"Complex no = ";
        cout<<r<<"+"<<i<<"i"<<endl;
    }
}

```

```

        }
    complex complex::operator+(complex p);
};

complex complex::operator+(complex p)
{
    complex q;
    q.r=r+p.r;
    q.i=i+p.i;
    return(q);
}

main()
{
complex x,y,z;
clrscr();
x.getdata();
y.getdata();
x.putdata();
y.putdata();
cout<<"Addition of ";
z=x+y;
z.putdata();
getch();
return(0);
}

```

Write a Program to perform Subtraction of Complex no of two Object of Class & Overload a Binary Operator - using Members Function.

```

#include<iostream.h>
#include<conio.h>

class complex
{
private:
    int r,i;
public:
    void getdata()
    {
        cout<<"Enter real and imaginary part = ";
        cin>>r>>i;
    }
    void putdata()
    {
        cout<<"Complex no = ";
        cout<<r<<"+"<<i<<"i"<<endl;
    }
    complex complex::operator-(complex p);
};

```

```

complex complex::operator-(complex p)
{
    complex q;
    q.r=r-p.r;
    q.i=i-p.i;
    return(q);
}

main()
{
    complex x,y,z;
    clrscr();
    x.getdata();
    y.getdata();
    x.putdata();
    y.putdata();
    cout<<"Subtraction of ";
    z=x-y;
    z.putdata();
    getch();
    return(0);
}

```

Write a Program to perform Addition of Matrix of two Object of Class & Overload a Binary Operator + using Members Function.

```

#include<iostream.h>
#include<conio.h>
class matrix
{
private:
    int m[3][3];
public:
    void getdata();
    void putdata();
    matrix operator+(matrix);
};

void matrix::getdata()
{
cout<<"Enter matrix element =\n";
for(int a=0;a<3;a++)
{
    for(int b=0;b<3;b++)
        cin>>m[a][b];
}
}

void matrix::putdata()
{
cout<<"Matrix is =\n";

```

```

for(int a=0;a<3;a++)
{
    for(int b=0;b<3;b++)
        cout<<m[a][b]<<"\t";
    cout<<endl;
}
}

matrix matrix::operator+(matrix p)
{
matrix q;
for(int a=0;a<3;a++)
{
    for(int b=0;b<3;b++)
        q.m[a][b]=m[a][b]+p.m[a][b];
}
return(q);
}

main()
{
matrix x,y,z;
clrscr();
x.getdata();
y.getdata();
z=x+y;
cout<<"Addition of ";
z.putdata();
getch();
return(0);
}

```

Write a Program to perform subtraction of Matrix of two Object of Class & Overload a Binary Operator - using Members Function.

```

#include<iostream.h>
#include<conio.h>
class matrix
{
private:
    int m[3][3];
public:
    void getdata();
    void putdata();
    matrix operator-(matrix);
};

void matrix::getdata()
{
cout<<"Enter matrix element =\n";
for(int a=0;a<3;a++)

```

```

{
    for(int b=0;b<3;b++)
        cin>>m[a][b];
}
}

void matrix::putdata()
{
cout<<"Matrix is =\n";
for(int a=0;a<3;a++)
{
    for(int b=0;b<3;b++)
        cout<<m[a][b]<<"\t";
    cout<<endl;
}
}

matrix matrix::operator-(matrix p)
{
matrix q;
for(int a=0;a<3;a++)
{
    for(int b=0;b<3;b++)
        q.m[a][b]=m[a][b]-p.m[a][b];
}
return(q);
}

main()
{
matrix x,y,z;
clrscr();
x.getdata();
y.getdata();
z=x-y;
cout<<"Subtraction of ";
z.putdata();
getch();
return(0);
}

```

Write a Program to perform Multiplication of Matrix of two Object of Class & Overload a Binary Operator * using Members Function.

```

#include<iostream.h>
#include<conio.h>
class matrix
{
private:
    int m[3][3];
public:

```

```

        void getdata();
        void putdata();
        matrix operator*(matrix);
    };

void matrix::getdata()
{
    cout<<"Enter matrix element =\n";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            cin>>m[a][b];
    }
}

void matrix::putdata()
{
    cout<<"Matrix is =\n";
    for(int a=0;a<3;a++)
        for(int b=0;b<3;b++)
            cout<<m[a][b]<<"\t";
        cout<<endl;
}

matrix matrix::operator*(matrix p)
{
    matrix q;
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
        {
            q.m[a][b]=0;
            for(int c=0;c<3;c++)
                q.m[a][b]=q.m[a][b]+m[a][c]*p.m[c][b];
        }
    }
    return(q);
}

main()
{
    matrix x,y,z;
    clrscr();
    x.getdata();
    y.getdata();
    z=x*y;
    cout<<"Multiplication of ";
    z.putdata();
    getch();
    return(0);
}

```

Write a Program to perform Transpose of Matrix of two Object of Class & Overload a Binary Operator ~ using Members Function.

```
#include<iostream.h>
#include<conio.h>
class matrix
{
    private:
        int m[3][3];
    public:
        void getdata();
        void putdata();
        matrix operator~();
};

void matrix::getdata()
{
    cout<<"Enter matrix element =\n";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            cin>>m[a][b];
    }
}

void matrix::putdata()
{
    cout<<"Matrix is =\n";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            cout<<m[a][b]<<"\t";
        cout<<endl;
    }
}

matrix matrix::operator~()
{
    matrix q;
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            q.m[a][b]=m[b][a];
    }
    return(q);
}

main()
{
    matrix x,z;
    clrscr();
```

```

x.getdata();
z=~x;
cout<<"Transpose of ";
z.putdata();
getch();
return(0);
}

```

Write a Program to perform Addition, Multiplication & Transpose of Matrix of two Object of Class & Overload a Binary Operator +,* & ~ using Members Function.

```

#include<iostream.h>
#include<conio.h>
class matrix
{
    private:
        int m[3][3];
    public:
        void getdata();
        void putdata();
        matrix operator+(matrix);
        matrix operator*(matrix);
        matrix operator~();
};

void matrix::getdata()
{
    cout<<"Enter matrix element =\n";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            cin>>m[a][b];
    }
}

void matrix::putdata()
{
    cout<<"Matrix is =\n";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            cout<<m[a][b]<<"\t";
        cout<<endl;
    }
}

matrix matrix::operator+(matrix p)
{
    matrix q;
    for(int a=0;a<3;a++)
    {

```

```

        for(int b=0;b<3;b++)
            q.m[a][b]=m[a][b]+p.m[a][b];
    }
    return(q);
}

matrix matrix::operator*(matrix p)
{
matrix q;
for(int a=0;a<3;a++)
{
    for(int b=0;b<3;b++)
    {
        q.m[a][b]=0;
        for(int c=0;c<3;c++)
            q.m[a][b]=q.m[a][b]+m[a][c]*p.m[c][b];
    }
}
return(q);
}

matrix matrix::operator~()
{
matrix q;
for(int a=0;a<3;a++)
{
    for(int b=0;b<3;b++)
        q.m[a][b]=m[b][a];
}
return(q);
}

main()
{
matrix x,y,z,g,h;
clrscr();
x.getdata();
y.getdata();
z=x+y;
g=x*y;
h=~x;
cout<<"Addition of ";
z.putdata();
cout<<"Multiplication of ";
g.putdata();
cout<<"Transpose of ";
h.putdata();
getch();
return(0);
}

```

Write a Program to Compare the data Members of 2 Object values using Overload Operator “<”.

```
#include<iostream.h>
#include<conio.h>

class value
{
    private:
        int no;
    public:
        void getdata()
        {
            cout<<"Enter the no = ";
            cin>>no;
        }
        int operator <(value);
};

int value :: operator <(value p)
{
    if(no<p.no)
        return(1);
    else
        return(0);
}

main()
{
value x,y;
clrscr();
x.getdata();
y.getdata();
if(x<y)
    cout<<"Y Object Value is Large";
else
    cout<<"X Object Value is Large";
getch();
return(0);
}
```

Write a Program to Compare the data Members of 2 Object values using Overload Operator “>”.

```
#include<iostream.h>
#include<conio.h>

class value
{
    private:
        int no;
```

```

public:
    void getdata()
    {
        cout<<"Enter the no = ";
        cin>>no;
    }
    int operator >(value);
};

int value :: operator >(value p)
{
    if(no>p.no)
        return(1);
    else
        return(0);
}

main()
{
value x,y;
clrscr();
x.getdata();
y.getdata();
if(x>y)
    cout<<"X Object Value is Large";
else
    cout<<"Y Object Value is Large";
getch();
return(0);
}

```

Write a Program to Check 2 Object values are same or not using Overload Operator “!=”.

```

#include<iostream.h>
#include<conio.h>

class value
{
private:
    int no;
public:
    void getdata()
    {
        cout<<"Enter the no = ";
        cin>>no;
    }
    int operator !=(value);
};

```

```

int value :: operator !=(value p)
{
    if(no!=p.no)
        return(1);
    else
        return(0);
}

main()
{
value x,y;
clrscr();
x.getdata();
y.getdata();
if(x!=y)
    cout<<"Object Values are not Equal";
else
    cout<<"Object Values are Equal";
getch();
return(0);
}

```

Write a Program to Copy one Object values to another using Overload Operator “=”.

```

#include<iostream.h>
#include<conio.h>

class value
{
private:
    int a,b;
public:
    void getdata()
    {
        cout<<"Enter two no = ";
        cin>>a>>b;
    }
    void putdata()
    {
        cout<<"No 1 = "<<a<<endl
            <<"No 2 = "<<b<<endl;
    }
    void operator =(value);
};

void value :: operator =(value p)
{
    a=p.a;
    b=p.b;
}

```

```

main()
{
value x,y;
clrscr();
x.getdata();
y=x;
cout<<"Copy Value = "<<endl;
y.putdata();
getch();
return(0);
}

```

Overload Operator using Friend Function

Friend function may be used in the place of member function for overloading operator. The only difference being that a friend function requires two arguments to be explicitly passed.

Note 1)

If a operator function is friend function and overload operator is binary then there are two arguments.

Note 2)

If a operator function is friend function and overload operator is unary then there is one argument to the operator function.

Operator Type	Member Function	Friend Function
1) Unary	---	1 Argument
2) Binary	1 Argument	2 Argument

Rules for Overloading Operator

The following are the certain restriction and limitation in operator overloading.

- 1) Only existing operators can be overloaded. New operator can not be created.
- 2) The overloading operator must have at least one operand i.e. a user defined type.
- 3) We can not change the basic meaning of an operator i.e. we can not redefine the + operator to subtract one value from the other.
- 4) Overloaded operator follow the syntax rules of the original operators. That can not be overridden.
- 5) There are some operators can not be overloaded.
- 6) We can not use friend function to overload certain operators. However member function can be used to overload them.
- 7) Unary operators overloaded by means of a member function, take no explicit argument and return no explicit values. But in case of friend function take one reference argument.
- 8) Binary operator overloaded through member function take one explicit argument & those which are overloaded through a friend function take two explicit arguments.

- 9) When using binary operators overloaded through a member function, the left hand operand must be an object of the class.
- 10) Binary arithmetic operator such as + , - , * , / must explicit return a value. They must no attempt to change there own argument.

The following are the operator that can not be overloaded using member function.

Sizeof , . (Dot operator) , .* (Dot star) , :: (Scope resolution operator) , ? (Conditional Operator)

The following are the operator that can not be overloaded using friend function.

= , () , [] , ->

Write a Program to Overload a Unary Operator ++ to Increase the Values of data Members of Object by one using Friend Function.

```
#include<iostream.h>
#include<conio.h>

class value
{
private:
    int a,b;
public:
    void getdata()
    {
        cout<<"Enter two nos = ";
        cin>>a>>b;
    }
    void putdata()
    {
        cout<<"No 1 = "<<a<<endl
            <<"No 2 = "<<b<<endl;
    }
    friend void operator++(value &);
};

void operator ++(value &x)
{
    x.a++;
    x.b++;
}

main()
{
value x;
clrscr();
x.getdata();
++x;
x.putdata();
```

```

getch();
return(0);
}

Write a Program to Overload a Unary Operator -- to Decrease the Values of data Members of Object by one using Friend Function.

#include<iostream.h>
#include<conio.h>

class value
{
private:
    int a,b;
public:
    void getdata()
    {
        cout<<"Enter two no = ";
        cin>>a>>b;
    }
    void putdata()
    {
        cout<<"No 1 = "<<a<<endl
            <<"No 2 = "<<b<<endl;
    }
    friend void operator--(value &);
};

void operator--(value &x)
{
    x.a--;
    x.b--;
}

main()
{
value x;
clrscr();
x.getdata();
x--;
x.putdata();
getch();
return(0);
}

```

Write a Program to perform Addition of Complex no of two Object of Class & Overload a Binary Operator + using Friend Function.

```

#include<iostream.h>
#include<conio.h>

```

```

class complex
{
    private:
        int r,i;
    public:
        void getdata()
        {
            cout<<"Enter real & imag part = ";
            cin>>r>>i;
        }
        void putdata()
        {
            cout<<"Complex no = ";
            cout<<r<<"+"<<i<<"i"<<endl;
        }
        friend complex operator +(complex,complex);
};

complex operator +(complex a,complex b)
{
    complex c;
    c.r=a.r+b.r;
    c.i=a.i+b.i;
    return(c);
}

main()
{
    complex x,y,z;
    clrscr();
    x.getdata();
    y.getdata();
    x.putdata();
    y.putdata();
    z=x+y;
    cout<<"Addition of ";
    z.putdata();
    getch();
    return(0);
}

```

Write a Program to perform Subtraction of Complex no of two Object of Class & Overload a Binary Operator - using Friend Function.

```

#include<iostream.h>
#include<conio.h>

class complex
{
    private:
        int r,i;

```

```

public:
    void getdata()
    {
        cout<<"Enter real & imag part = ";
        cin>>r>>i;
    }
    void putdata()
    {
        cout<<"Complex no = ";
        cout<<r<<"+"<<i<<"i"<<endl;
    }
    friend complex operator -(complex,complex);
};

complex operator -(complex a,complex b)
{
    complex c;
    c.r=a.r-b.r;
    c.i=a.i-b.i;
    return(c);
}

main()
{
complex x,y,z;
clrscr();
x.getdata();
y.getdata();
x.putdata();
y.putdata();
z=x-y;
cout<<"Subtraction of ";
z.putdata();
getch();
return(0);
}

```

Write a Program to perform Addition of Matrix of two Object of Class & Overload a Binary Operator + using Friend Function.

```

#include<iostream.h>
#include<conio.h>

class matrix
{
private:
    int m[3][3];
public:
    void getdata();
    void putdata();
    friend matrix operator +(matrix,matrix);
};

```

```

void matrix::getdata()
{
    cout<<"Enter matrix element =\n";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            cin>>m[a][b];
    }
}

void matrix::putdata()
{
    cout<<"Matrix is =\n";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            cout<<m[a][b]<<"\t";
        cout<<endl;
    }
}

matrix operator +(matrix p,matrix q)
{
    matrix z;
    cout<<"Addition of ";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            z.m[a][b]=p.m[a][b]+q.m[a][b];
    }
    return(z);
}

main()
{
    matrix a,b,c;
    clrscr();
    a.getdata();
    b.getdata();
    c=a+b;
    c.putdata();
    getch();
    return(0);
}

```

Write a Program to perform subtraction of Matrix of two Object of Class & Overload a Binary Operator - using Friend Function.

```
#include<iostream.h>
#include<conio.h>
```

```

class matrix
{
    private:
        int m[3][3];
    public:
        void getdata();
        void putdata();
        friend matrix operator -(matrix,matrix);
};

void matrix::getdata()
{
    cout<<"Enter matrix element =\n";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            cin>>m[a][b];
    }
}

void matrix::putdata()
{
    cout<<"Matrix is =\n";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            cout<<m[a][b]<<"\t";
        cout<<endl;
    }
}

matrix operator -(matrix p,matrix q)
{
    matrix z;
    cout<<"Substaction of ";
    for(int a=0;a<3;a++)
        for(int b=0;b<3;b++)
            z.m[a][b]=p.m[a][b]-q.m[a][b];
    return(z);
}

main()
{
matrix a,b,c;
clrscr();
a.getdata();
b.getdata();
c=a-b;
c.putdata();
getch();
return(0);
}

```

Write a Program to perform Multiplication of Matrix of two Object of Class & Overload a Binary Operator * using Friend Function.

```
#include<iostream.h>
#include<conio.h>

class matrix
{
private:
    int m[3][3];
public:
    void getdata();
    void putdata();
    friend matrix operator *(matrix,matrix);
};

void matrix::getdata()
{
    cout<<"Enter matrix element =\n";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            cin>>m[a][b];
    }
}

void matrix::putdata()
{
    cout<<"Matrix is =\n";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            cout<<m[a][b]<<"\t";
        cout<<endl;
    }
}

matrix operator *(matrix p,matrix q)
{
    matrix z;
    cout<<"Multiplication of ";
    for(int a=0;a<3;a++)
        for(int b=0;b<3;b++)
        {
            z.m[a][b]=0;
            for(int c=0;c<3;c++)
                z.m[a][b]=z.m[a][b]+p.m[a][c]*q.m[c][b];
        }
    return(z);
}
```

```

main()
{
matrix a,b,c;
clrscr();
a.getdata();
b.getdata();
c=a*b;
c.putdata();
getch();
return(0);
}

```

Write a Program to perform Transpose of Matrix of two Object of Class & Overload a Binary Operator ~ using Friend Function.

```

#include<iostream.h>
#include<conio.h>

class matrix
{
private:
    int m[3][3];
public:
    void getdata();
    void putdata();
    friend matrix operator ~(matrix);
};

void matrix::getdata()
{
    cout<<"Enter matrix element =\n";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            cin>>m[a][b];
    }
}

void matrix::putdata()
{
    cout<<"Matrix is =\n";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            cout<<m[a][b]<<"\t";
        cout<<endl;
    }
}

```

```

matrix operator ~(matrix p)
{
    matrix z;
    cout<<"Transpose of ";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            z.m[a][b]=p.m[b][a];
    }
    return(z);
}

main()
{
    matrix a,b;
    clrscr();
    a.getdata();
    b=~a;
    b.putdata();
    getch();
    return(0);
}

```

Write a Program to perform Addition, Multiplication & Transpose of Matrix of two Object of Class & Overload a Binary Operator +,* & ~ using Friend Function.

```

#include<iostream.h>
#include<conio.h>

class matrix
{
private:
    int m[3][3];
public:
    void getdata();
    void putdata();
    friend matrix operator +(matrix,matrix);
    friend matrix operator -(matrix,matrix);
    friend matrix operator *(matrix,matrix);
    friend matrix operator ~(matrix);
};

void matrix::getdata()
{
    cout<<"Enter matrix element =\n";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            cin>>m[a][b];
    }
}

```

```

void matrix::putdata()
{
    cout<<"Matrix is =\n";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
        cout<<m[a][b]<<"\t";
        cout<<endl;
    }
}

matrix operator +(matrix p,matrix q)
{
    matrix z;
    cout<<"Addition of ";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
        z.m[a][b]=p.m[a][b]+q.m[a][b];
    }
    return(z);
}

matrix operator -(matrix p,matrix q)
{
    matrix z;
    cout<<"Substaction of ";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
        z.m[a][b]=p.m[a][b]-q.m[a][b];
    }
    return(z);
}

matrix operator *(matrix p,matrix q)
{
    matrix z;
    cout<<"Multiplication of ";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
        {
            z.m[a][b]=0;
            for(int c=0;c<3;c++)
            {
                z.m[a][b]=z.m[a][b]+p.m[a][c]*q.m[c][b];
            }
        }
    }
    return(z);
}

```

```

matrix operator ~(matrix p)
{
    matrix z;
    cout<<"Transpose of ";
    for(int a=0;a<3;a++)
    {
        for(int b=0;b<3;b++)
            z.m[a][b]=p.m[b][a];
    }
    return(z);
}

main()
{
    matrix a,b,c;
    clrscr();
    a.getdata();
    b.getdata();
    c=a+b;
    c=a-b;
    c=a*b;
    c=a^b;
    c=~a;
    c.putdata();
    getch();
    return(0);
}

```

Write a Program to Overload Binary Operator “<” using Friend Function.

```

#include<iostream.h>
#include<conio.h>

class value
{
private:
    int no;
public:
    void getdata()
    {
        cout<<"Enter the no = ";
        cin>>no;
    }
    friend int operator <(value,value);
};

int operator <(value p,value q)
{
    if(p.no<q.no)

```

```

        return(1);
    else
        return(0);
}

main()
{
value x,y;
clrscr();
x.getdata();
y.getdata();
if(x<y)
    cout<<"2nd Object Value is Large";
else
    cout<<"1st Object Value is Large";
getch();
return(0);
}

```

Write a Program to Overload Binary Operator “>” using Friend Function.

```

#include<iostream.h>
#include<conio.h>

class value
{
private:
    int no;
public:
    void getdata()
    {
        cout<<"Enter the no = ";
        cin>>no;
    }
    friend int operator >(value,value);
};

int operator >(value p,value q)
{
    if(p.no>q.no)
        return(1);
    else
        return(0);
}

main()
{
value x,y;
clrscr();
x.getdata();
y.getdata();

```

```

if(x>y)
    cout<<"1st Object Value is Large";
else
    cout<<"2nd Object Value is Large";
getch();
return(0);
}

```

Write a Program to Compare the data Members of 2 Object values using Overload Operator “ != ” & Friend Function.

```

#include<iostream.h>
#include<conio.h>

class value
{
private:
    int no;
public:
    void getdata()
    {
        cout<<"Enter the no = ";
        cin>>no;
    }
    friend int operator !=(value,value);
};

int operator !=(value p,value q)
{
    if(p.no!=q.no)
        return(1);
    else
        return(0);
}

main()
{
value x,y;
clrscr();
x.getdata();
y.getdata();
if(x!=y)
    cout<<"Object Values are not Equal";
else
    cout<<"Object Values are Equal";
getch();
return(0);
}

```

Manipulation of String using Operators

C implements strings using character arrays, pointer and string functions. There are no operators for manipulating the string. One of the main drawbacks of string manipulations in C is that whenever a string is to be copied, the programmer must first determine its length and allocate the required amount of memory.

Although these limitation exist in C++ as well, it permits us to create our own definitions of operators that can be used to manipulate the strings very much similar to the numbers.

e.g.

```
string3 = string1 + string2;
if(string1>=string2)
    string=string1;
```

Strings can be defined as class objects which can be then manipulated like the built-in types. Since the strings vary greatly in size, we use new to allocate memory for each string and a pointer variable to point to the string array. Thus, we must create string objects that can hold these two pieces of information, namely, length and location which are necessary for string manipulations. A typical string class will look as follows:

```
class string
{
    char *p;
    int len;
public :
    .....           // member function
    .....           // to initialize and
    .....           // manipulate strings
};
```

Write a Program to Concatenating 2 String and Compare 2 String using Overload Operator “+” & “<= ”.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>

class string
{
private:
    char *p;
    int len;
public:
    string()
    {
        len=0;
        p=0;
    }
    string (char *s);
    string operator +(string);
```

```

        int operator <=(string);
        void show()
        {
            cout<<"String = "<<p<<endl;
        }
    };

string :: string(char *s)
{
    len=strlen(s);
    p=new char[len+1];
    strcpy(p,s);
}

string string :: operator +(string s)
{
    string t;
    t.len=len+s.len;
    t.p=new char[t.len+1];
    strcpy(t.p,p);
    strcat(t.p,s.p);
    return(t);
}

int string :: operator <=(string s)
{
    int m=strlen(p);
    int n=strlen(s.p);
    if(m<=n)
        return(1);
    else
        return(0);
}

main()
{
    string s1="Amol ",s2="Patil",s3="Aim",t1,t2;
    t1=s1+s2;
    t2=s1+s3;
    clrscr();
    s1.show();
    s2.show();
    s3.show();
    t1.show();
    t2.show();
    if(t1<=t2)
    {
        t1.show();
        cout<<" smaller than ";
        t2.show();
    }
}

```

```

else
{
    t2.show();
    cout<<" smaller than ";
    t1.show();
}
getch();
return(0);
}

```

Write a Program to Concatenating 2 String using Overload Operator “+”.

```

#include<iostream.h>
#include<conio.h>
#include<stdio.h>

class string
{
private:
    char str[30];
public:
    void getdata()
    {
        cout<<"Enter the String = ";
        gets(str);
    }
    void putdata()
    {
        cout<<"String = ";
        puts(str);
    }
    string operator +(string);
};

string string :: operator +(string x)
{
string z;
int a=0,b=0;
    while(str[a]!='\0')
    {
        z.str[a]=str[a];
        a++;
    }
z.str[a]=' ';
a++;
    while(x.str[b]!='\0')
    {
        z.str[a]=x.str[b];
        a++;
        b++;
    }
z.str[a]='\0';
}

```

```

        return(z);
    }

main()
{
    string p,q,r;
    clrscr();
    p.getdata();
    q.getdata();
    r=p+q;
    p.putdata();
    q.putdata();
    r.putdata();
    getch();
    return(0);
}

```

Write a Program for Multiple Overloading Comparison Operator.

```

#include<iostream.h>
#include<conio.h>

class value
{
private:
    int no;
public:
    void getdata()
    {
        cout<<"Enter the no = ";
        cin>>no;
    }
    friend int operator <(value,value);
    friend int operator >(value,value);
};

int operator <(value p,value q)
{
    if(p.no<q.no)
        return(1);
    else
        return(0);
}

int operator >(value p,value q)
{
    if(p.no>q.no)
        return(1);
    else
        return(0);
}

```

```

main()
{
value x,y;
clrscr();
x.getdata();
y.getdata();
if(x<y)
    cout<<"2nd Object Value is Large";
else
    cout<<"1st Object Value is Large";
value p,q;
cout<<endl;
p.getdata();
q.getdata();
if(p>q)
    cout<<"1st Object Value is Large";
else
    cout<<"2nd Object Value is Large";
getch();
return(0);
}

```

Type Conversion

We know that when constants and variables of different types are mixed in an expression. C applies automatic type conversion to the operands as per certain rules. Similarly, an assignment operation also causes automatic type conversion. The type of data to the right of an assignment operator is automatically converted to the type of the variable on the left.

The type conversion are automatic as long as the data types involved are built-in types. But the type conversion is not possible for user defined data type. We have seen, in the case of class objects, the values of all the data members of the right-hand object are simply copied into the corresponding members of the object on the left-hand. What if one of the operands is an object and the other is a built-in type variable or what if they belong to 2 different classes.

Since the user-defined data type are designed by us to suit our requirements, the compiler does not support automatic type conversions for such data types. We, must therefore, design the conversion routines by ourselves.

Three types of situation might arise in the data conversion between incompatible types:

1. Conversion from built-in type to class type.
2. Conversion from class type to built-in type.
3. Conversion from one class type to another class type.

1) Basic to Class Type

The conversion from basic type to class type is easy to accomplish. It may be recalled that the use of constructors was illustrated in a number of examples to initialize objects.

Let us consider example of converting an int type to a class type.

```

class time
{
    int hour;
    int mins;
public :
    .....
    .....
    time(int t)
    {
        hour = t / 60;
        mins = t % 60;
    }
};

```

The following conversion statement can be used in a function.

```

time t1;
int duration = 85;
t1 = duration;

```

Note

The constructors used for the type conversion take a single argument whose type is to be converted. In this conversion, the left hand operand of = operator is always a class object. Therefore, we can also accomplish this conversion using an overloaded = operator.

Write a Program to Demonstration of Type Conversion form Basic to Class type.

```

#include<iostream.h>
#include<conio.h>

class time
{
private:
    int hour;
    int mins;
public:
    time(int t)
    {
        hour=t/60;
        mins=t%60;
    }
    void putdata()
    {
        cout<<hour<<" Hour ";
        cout<<mins<<" Mintues";
    }
};

main()
{
    time x(85);
}

```

```

clrscr();
cout<<"Duration = ";
x.putdata();
getch();
return(0);
}

```

2) Class to Basic Type

The constructors did a fine job in type conversion from a basic to class type. The constructor functions do not support this operation for conversion from class to basic type. C++ allows us to define a overloaded casting operator that could be used to convert a class type data to a basic type. The general form of an overloaded casting operator function, usually referred to as a conversion function, is:

```

operator type name( )
{
    ....
    ..... (Function Statement)
    ....
}

```

e.g.

```

vector :: operator double( )
{
    double sum = 0;
    for(int i=0;i<size;i++)
        sum = sum + v[i]*v[i];
    return sqrt(sum);
}

```

The above operator function invoked the following statement.

```

double length = double(v1);
or
double length = v1;

```

Where v1 is object of type vector.

When the compiler encounters a statement that requires the conversion of a class type to a basic type, it quietly calls the casting operator function to do the job. The casting operator function should satisfy the following condition.

1. It must be a class member.
2. It must not specify a return type.
3. It must not have any arguments.

Since it is a member function, it is invoked by the object and therefore, the values used for conversion inside the function belong to the object that invoked the function. This means that the function does not need an argument.

3) One Class to Another Class Type

We have just seen data conversion techniques from a basic to class type and a class to basic type. But there are situations where we would like to convert one class type data to another class type.

e.g.

Obj X = Obj Y;

ObjX is an object of class X and objY is an object of class Y. The class Y type data is converted to the class X type data and the converted value is assigned to the objX. Since the conversion takes place from class Y to class X, Y is known as the source class and X is known as the destination class.

Such conversion between objects of different classes can be carried out by either a constructor or a conversion function. The compiler treats them the same way. It depends upon where we want the type-conversion function to be located, in the source class or in the destination class. We know that the casting operator function.

Operator type name()

Converts the class object of which it is a member to type name. The type name may be a built-in type or a user-defined one(another class type). In the case of conversion between objects, type name refers to the destination class. Therefore, when a class needs to be converted, a casting operator function can be used (i.e. source class).

Now consider a single-argument constructor function which serves as an instruction for converting the argument's type to the class type of which it is a member. This implies that the argument belongs to the source class and is passed to the destination class for conversion. This makes it necessary that the conversion constructor be placed in the destination class.

Write a Program to Demonstration of Type Conversion.

```
#include<iostream.h>
#include<conio.h>

class invent1
{
    private:
        int code;
        int item;
        float price;
    public:
        invent1(int a,int b,float c)
        {
            code=a;
            item=b;
            price=c;
        }
}
```

```

void putdata()
{
    cout<<"Code = "<<code<<endl
        <<"Items = "<<item<<endl
        <<"Value = "<<price<<endl;
}
int getcode()
{
    return code;
}
int getitem()
{
    return item;
}
float getprice()
{
    return price;
}
operator float()
{
    return(item*price);
}
};

class invent2
{
private:
    int code;
    float value;
public:
    invent2()
    {
        code=0;
        value=0;
    }
    invent2(int x,float y)
    {
        code=x;
        value=y;
    }
    void putdata()
    {
        cout<<"Code = "<<code<<endl
            <<"Value = "<<value<<endl;
    }
    invent2(invent1 p)
    {
        code=p.getcode();
        value=p.getitem()*p.getprice();
    }
};

```

```

main()
{
clrscr();
invent1 s1(100,5,140.0);
invent2 d1;
float total;
total=s1;
d1=s1;
cout<<"Product Detail of Invent1 "<<endl;
s1.putdata();
cout<<"Stock Value = "<<total<<endl;
cout<<"Product Detail of Invent2 "<<endl;
d1.putdata();
getch();
return(0);
}

```

Write a Program in C++ by using overloaded operator to compare distance in feet and inch.

e.g. **dist1 = 5' 11.5"**
dist2 = 6' 2.5"
o/p -> dist1 is less than dist2.

```

#include<iostream.h>
#include<conio.h>

class dist
{
private:
    float feet,inch;
public:
    void getdata()
    {
        cout<<"Enter the Distance in Feet Inch. = ";
        cin>>feet>>inch;
    }
    void putdata()
    {
        cout<<"Distance = "<<feet<<"' "<<inch<<"\""<<endl;
    }
    int operator <(dist);
};

int dist :: operator <(dist p)
{
    if(feet==p.feet)
    {
        if(inch<p.inch)
        return(1);
        else
        return(0);
    }
}

```

```
    }
    if(feet<p.feet)
        return(1);
    else
        return(0);
}

main()
{
dist x,y;
clrscr();
x.getdata();
y.getdata();
x.putdata();
y.putdata();
if(x<y)
    cout<<"Dist1 is less than Dist2";
else
    cout<<"Dist2 is less than Dist1";
getch();
return(0);
}
```