Operator Overloading: The mechanism of adding special meaning to an operator is known as operator overloading. It provides a flexibility for the creation of new definitions for most C++ operators.

Using operator overloading, we can give additional meaning to basic C++ operators such as (+,-,<,== etc.) when they are applied to user defined data types.

Benefits/Use of Operator overloading: Generally, operations using operators can be performed only on basic data type.
Example: int a, b, c;
C = a + b;
But if we declare a class Complex { }; and create objects from it as:
Complex c1, c2, c3;
Then, c3=c1+c2; is not possible because objects are user-defined data types.

Operator overloading helps to define the usage of operator for user defined data types, i.e. objects. That means, c3 = c1+ c2 for objects is possible if the operator (+) is overloaded.

Operators that cannot be overloaded: All operators cannot be overloaded. The operators that are not overloaded are:

- Class member access operators (. and .*)
- Scope resolution Operator (::)
- Sizeof operator(sizeof)
- Conditional Operator(? :)

General form of operator function:

```
return_type operator op(arglist) {
    function body //task defined
}
```

Where, 'return_type' is the type of value returned by the specified operation.
'op' is the operator being overloaded.
'operator op' is function name, where operator is keyword.

Types of operators that are overloaded:
a) Unary operators: These are the operators that operate on single operand. Some of unary operators are: Increment operator (++), Decrement operator (--), Unary minus operator(-)
For example: ++a ; Here a is only one operand.

b) Binary operators: The operator which operates on two operands is known as binary operators. Arithematic operators (-, +, *, /), Comparision operators (>, <, < =, > =, = =), Assignment operators (+=, - =, =).
For example: c=a+b where a and b are two operands.

- Operator function must be either member function (non-static) or friend function.
- Member function has no arguments for unary operators and only one for binary operators.
  Eg. Abc operator + ( ) //unary
  Abc operator + (Abc) //binary
- Friend function will have only one argument for unary operators and two for binary operators. This is because the object used to invoke the member function is passed implicitly and therefore available for member function. This is not the case with friend functions.
  Eg. friend Abc operator + (Abc) //unary
  friend Abc operator - (Abc, Abc) //binary
- Arguments may be passed either by value or by reference. Example of prototype where the arguments are passed by reference:
  Eg. Abc operator + (Abc &a) //argument passed by reference

Mechanism for Operator overloading: Generally, the process of overloading involves the following steps:

- Create a class that defines the data type that is used in the overloading function.
- Declare the operator function operator op( ) in the public part of class. It can either be normal member function or a friend function.
- Define operator function to implement the required operations.
- Calling (invoking) the operator overloaded function.

Operator overloaded function can be invoked using expression such as:

	In case of member function	In case of friend function
For unary operators	op x or x op (eg.++x, or x++) *x.operator op();*	op x or x op (eg.++x, or x++) *operator op(x)*
For binary operators	x op y eg x+y *x.operator op(y);*	x op y eg x+y *operator op(x,y)*

- Here, op represents the operator being overloaded.
- x and y represents the objects.

A. Overloading unary operators:

1. Overloading unary (-) operator:

Example program: 6.1

```
#include <iostream>
using namespace std;

class A{

 int x;
 int y;

 public:
 A(int a, int b){
   x=a;
   y=b;
 }

 void display(){
  cout<<"Value of x: "<<x<<endl;
  cout<<"Value of y: "<<y<<endl;
 }
 void operator-(){     // - operator overloading
   x=-x;
   y=-y;
 }

};
int main()
{
  A object(2,5);
  object.display();
  -object;      //it calls the operator overloading function
  object.display();

  return 0;
}
```

Output: Value of x: 2
Value of y: 5
Value of x: -2
Value of y: -5

 # Overloading unary (–) operator using friend function:

Example program: 6.2

```
class A{

 int x;
 int y;

 public:
 A(int a, int b){
   x=a;
   y=b;
 }

 void display(){
  cout<<"Value of x: "<<x<<endl;
  cout<<"Value of y: "<<y<<endl;
 }

 friend A operator-(A a);    // - operator overloading
                             // using friend function
};
A operator-(A a){      // - operator overloading
   A temp(2,5);
   temp.x=-a.x;
   temp.y=-a.y;
   return temp;  //returns object of type class A
 }

int main()
{
  A obj1(2,5);
  obj1.display();
  A obj2=-obj1;       //it calls the operator overloading function
                      //and saves the returned object in 'obj2'.
  obj2.display();

  return 0;
}
```

Output: Value of x: 2
Value of y: 5
Value of x: -2
Value of y: -5

2. Overloading the prefix (++) operator

Example program: 6.3

```
class A{
  int count;
   public:
    A(){    //Default constructor
    }
    A(int a){   //Parameterized constructor
     count=a;
    }
    void display(){
     cout<<"Value of count: "<<count<<endl;
    }
   A operator ++(){   //Operator function
     A temp;
     temp.count=++count;
     return temp;
    }
};
int main()
{
  A a1(4),b1;      //Creating two objects
  a1.display();
  b1=++a1;       //Operator overloading
  a1.display();
  b1.display();
  return 0;
}
```

Output: Value of count: 4
Value of count: 5
Value of count: 5

3. Overloading the postfix (++) operator

Example program: 6.4

```
class A{
  int count;
   public:
    A(){    //Default constructor
    }
    A(int a){   //Parameterized constructor
     count=a;
    }
    void display(){
     cout<<"Value of count: "<<count<<endl;
    }
   A operator ++(int){   //Operator function
     A temp;
     temp.count=count++;
     return temp;
    }
};
int main()
{
  A a1(4),b1;      //Creating two objects
  a1.display();
  b1=a1++;       //Operator overloading
  a1.display();
  b1.display();
  return 0;
}
```

Output: Value of count: 4
Value of count: 5
Value of count: 4

Note: For the overloading of postfix ++ operator, 'int' is placed in the parentheses ( ) of the operator function, so that the compiler doesn't get confused between the postfix and the prefix operator.

B. Overloading binary operators: The binary overloaded function takes the first object as an implicit operand and the second object must be passed explicitly. The data members of the first object are accessed without using the dot operator whereas the data members of the second object are accessed using the dot operator (if the argument is the object).

For example, if the 'op' operator is overloaded and the operator function is invoked as:

object1 op object2; then only object2 is passed explicitly as an argument

1. Overloading binary (+) operator:

Example program: 6.5

```
class Add
{
    int num;
    public:
    Add(){
    }
    Add(int x){
        num=x;
    }
    Add operator+(Add a){ //operator overloading function
     /** a is the copy of object 'a2'
         That means, object 'a' is same as object'a2'**/
      Add temp;  //creating an object to store the sum
      temp.num=num+a.num; /** function 'operator+( )' is
         invoked by 'a1' object, so no need to use dot(.)
         operator to access 'num' of 'a1' object.
      temp.num = num(this num is of 'a1' object)+a.num; **/

      return temp;/**returns 'temp' object which
          is stored in object 'a3' in main() funtion **/
    }
    void display(){
        cout<<"Value of num: "<<num<<endl;
    }
};
int main ()
{
    Add a1(5), a2(2); //Creating and initializing objects
    a1.display();
    a2.display();
    Add a3; // Creating third object
    a3=a1+a2; // binary operator '+' overloading //
    //operator function is invoked as a1.operator+(a2) //
    //operator function returns an object which
     // is stored in object 'a3'//
    a3.display();
    return 0;
}
```

 # Overloading binary (+) operator using friend function:

Example program: 6.6

```
class Complex{
  int real,imag;
   public:
    Complex(){    //Default constructor
    }
    Complex(int a, int b){   //Parameterized constructor
     real=a;
     imag=b;
    }
    friend Complex operator + (Complex obj1, Complex obj2);

    void display(){
     cout<<"The number: "<<real<<" + "<<imag<<"j"<<endl;
    }
};
 Complex operator + (Complex obj1, Complex obj2){
   Complex temp;
   temp.real=obj1.real+obj2.real;
   temp.imag=obj1.imag+obj2.imag;
   return temp;
 }


 int main()
 {
   Complex c1(3,4), c2(4,1);
   Complex c3=c1+c2;    //Operator overloading
   c3.display();
   return 0;
 }
```

Output: The number: 7 + 5j

2. Overloading '+=' operator:

Example program: 6.7

```
class A{
  int x,y;
  public:
  A(){
  }
  A(int a, int b){
    x=a;
    y=b;
  }
  void display(){
    cout<<"Value of x: "<<x<<endl;
    cout<<"Value of y: "<<y<<endl;
  }
  void operator += (A obj){ //operator function
    x+=obj.x;
    y+=obj.y;
  }
};
int main ()
{
    A a1(3,4), a2(2,5); //Creating objects
    a1.display();
    a2.display();
    a1+=a2;   //operator overloading
    cout<<"After addition: "<<endl;
    a1.display();
    return 0;
}
```

Output:

Value x: 3
Value y: 4
Value x: 2
Value y: 5
After addition:
Value x: 5
Value y: 9

3. Overloading '>' operator:

Example program: 6.8

```
class Maximum{
  int x;
  public:
  Maximum(){
  }
  Maximum(int a){
    x=a;
  }
  Maximum operator >(Maximum obj){ //operator function
    Maximum temp;
    if(x>obj.x){
        temp.x=x;
    }
    else{
        temp.x=obj.x;
    }
    return temp;
  }
  void display(){
   cout<<"Maximum value = "<<x<<endl;
  }
};
int main ()
{
    Maximum m1(3), m2(5); //Creating objects
    Maximum m3=m1>m2;    //operator overloading
    m3.display();
    return 0;
}
```

Output: Maximum value = 5

4. Overloading '==' operator:

Example program: 6.9

```
class Equal{
  int x,y;
  public:
  Equal(){
  }
  Equal(int a, int b){
    x=a;
    y=b;
  }
  int operator ==(Equal obj){ //operator function
    if(x==obj.x && y==obj.y){
        return 1;
    }
    else{
        return 0;
    }
  }
};
int main()
{
    Equal e1(3,5), e2(2,5);
    if(e1==e2){    //operator overloading
      cout<<"They are equal"<<endl;
    }
    else{
      cout<<"They are not equal"<<endl;
    }
    return 0;
}
```

Output: They are not equal.

Type Conversion: It is the process of converting one type of data to another type.

- Compiler automatically converts basic to another basic data type (for eg. int to float, float to int etc.) by applying type conversion rule provided by the compiler.
- The type of data to the right of the assignment operator (=) is automatically converted to the type of variable on the left.

Example: int a;
float b = 2.157;
a= b;
It converts "b" to an integer before its value is assigned to "a". So, the fractional part is truncated.

However, compiler does not support automatic type conversion for user-defined data type. For example, if length1 and length2 are the objects of two different classes, then,
length1=length2 is not possible.

Similarly, user defined data type to primitive data types and vice-versa is not possible.
Eg. If length is an object of a class, and meter is an integer data, then
meter = length; // not possible.
And length =meter; //not possible

In order to realize such type of conversions, we must design conversion routines. There are three possible type conversions:

A. Conversion from basic type to class type
B. Conversion from class type to basic type
C. Conversion from one class type to another class type

A. Conversion from basic type to user-defined type/class type: To convert the data from a basic type to a user-defined type, the conversion function must be defined in the class in the form of constructor. The constructor function takes a single argument of basic data type.

General format:
Constructor (Basic type)
{
//converting statements
}

An example program that demonstrates the conversion from basic type to class type is given below. *[Example program: 6.10]*

This program converts the length 'meter' into 'cm'. The length 'meter' is used a basic data type, and the length 'cm' is used as the data member of the user-defined type (class-type). This program uses the formula as: 1 meter= 100 cm.

Example program: 6.10

```
class Length{
  float cm;
  public:
    Length(){

    }
    Length(float m){  //Constructor
      cm=m*100;   //using formula to convert meter to cm
    }
    void display(){
      cout<<"After conversion, length = "<<cm<<" cms."<<endl;
    }
};

int main()
{
    float meter=5.5;  //value of meter is known in advance
    Length obj;  //We have to convert meter into cm,
                 //which is the member of class Length
    obj=meter;    //it calls the constructor 'Length(float m)'
                  //where 'm' is the copy of 'meter'
    obj.display();
    return 0;
}
```

Output: After conversion, length = 550 cms.

B. User-defined type/class type to basic type: To convert the data from a user-defined type to a basic type, the conversion function must be defined in the class in the form of the casting operator function. The casting operator function is defined as an overloaded basic data type which takes no arguments. It converts the data members of an object to basic data type and returns it.

General format:

```
operator basic_data_type( ) {
  //conversion statements
   return entity (of basic_data_type);
}
```

An example program that demonstrates the conversion from class type to basic type is given below. *[Example program 6.11]*

This program converts the weight 'kg' to gm. The weight 'kg' is used as the data member of the user-defined type (class-type) and the weight 'gm' is used a basic data type.

Example program: 6.11

```
class Weight{
  float kg;
  public:
   void getData(float x){ //Member function to initialize kg
     kg=x;
   }
   operator float(){ //operator function
    //the type 'float' should be same as the destination type 'gm' in main()
    float g=kg*100; //converting kg to gm
    return g;
   }
};
int main()
{
    Weight obj; //Creating object
    obj.getData(2); //Initializing kg (value of kg is known in advance)
    float gm;  //We have to convert kg to gm,
               //which is the basic data type
    gm=obj; //it calls the operator function 'operator float()'
            //and the returned value is saved in 'gm'
            //destination is always kept in left side//
    cout<<"After conversion, weight = "<<gm<<" gm."<<endl;
    return 0;
}
```

Output: After conversion, weight = 200 gm.

C. User-defined data type (class type) to User-defined data type (class type):
When a data of one class type is converted into data of another class type, it is called conversion of one class to another class type.

For example: objA = objB;

Here, objA is an object of class A and objB is an object of class B. The data of type class B is converted to the data of type class A and converted value of objB is assigned to the objA. Since the conversion takes place from class B to class A, B is known as source class and A is known as destination class. This type of conversion is carried out by either:

1. constructor
   or
2. casting operator function

So, one of the two alternatives can be chosen for the data conversion. The choice depends on whether we put the conversion routine in the destination class or in the source class. If we want the conversion routine to be located in the source class then the operator function method is chosen. And if we want the conversion routine to be located in the destination class then the constructor method is chosen.

1. Conversion Routine in source object/Using casting operator function:

General format:

```
class Destination {
 };
class Source{
   public:
      operator Destination( ) {
        // code for conversion
          return (Destination type);
      }
};
```

Now, in main( ) function,

```
// d1 is the object of class Destination and s1 is the object of class Source
        d1=s1;    // above casting operator function is called
```

An example program that demonstrates the conversion routine using (casting) operator function is given below. *[Example program: 6.12]*

This program converts the 'kg' of one class type to 'gm' of another class type.

Example program: 6.12

```
class WeightD{
 float gm;
 public:
    void getData(float g){
     gm=g;
    }
    void display(){
     cout<<"After conversion, weight= "<<gm<<" gm."<<endl;
    }
};
```

```
class WeightS{
  float kg;
  public:
   void getData(float x){ //Member function to initialize kg
     kg=x;
   }
   operator WeightD(){ //operator function
    //the type 'WeightD' should be same as the
    //destination type 'wd' in main() function
    float gm=kg*1000; //converting kg to gm
    WeightD temp;//Creating object of destination class
    temp.getData(gm); //pass the converted value of gm
    return temp; //returns object 'temp' which is
                 //stored in object 'wd' in main() function
   }
};
int main()
{
    WeightS ws; //Creating object of Source class
    ws.getData(2.5);//Initializing kg (value of kg is known in advance)
    WeightD wd; //Creating object of Destination class
      //we have to convert kg to gm(which is the member of WeightD)
    wd=ws; //it calls the operator function 'operator WeightD()'
           //and the returned value is stored in 'wd'
           //destination object is always kept in left side//
    wd.display();
    return 0;
}
```

Output: After conversion, weight= 2500 gm.

2. Conversion routine in destination object/Using constructor:

General format:

```
class Source {
};
class Destination{
  public:
    Destination(Source objectS ) {
      // code for conversion
    }
};
```

Now, in main( ) function,

```
// d1 is the object of class Destination and s1 is the object of class Source
       d1=s1;    // constructor called
```

An example program that demonstrates the conversion routine using constructor is given below. *[Example program 6.13]*

This program converts the 'kg' of one class type to 'gm' of another class type.

Example program 6.13

```
class WeightS{
 float kg;
 public:
    void getData(float k){
     kg=k;
    }
    float getKg(){ //creating function to access kg from outside
     return kg;
    }
};
class WeightD{
  float gm;
  public:
   WeightD(){ //Default Constructor
   }
   WeightD(WeightS w){   //Parameterized Constructor
    //object 'w' is same as object 'ws' //copy of 'ws' object
    float kg=w.getKg(); //accessing the value of kg
    //since 'kg' is in private section of source class 'WeightS',
      //we cannot directly access kg as 'w.kg'
    gm=kg*1000; //converting kg to gm
   }
   void display(){
    cout<<"After conversion, weight= "<<gm<<" gm."<<endl;
   }
};
int main()
{
    WeightS ws; //Creating object of Source class
    ws.getData(2.5);//Initializing kg (value of kg is known in advance)
    WeightD wd; //Creating object of Destination class
      //we have to convert kg to gm(which is the member of WeightD)
    wd=ws; //it calls the constructor 'WeightD(WeightS w)'
           //constructor receives the object 'ws' as argument
            //destination object is always kept in left side//
    wd.display();
    return 0;
}
```

Output: After conversion, weight= 2500 gm

OLD /MODEL/IMPORTANT QUESTIONS:

1. Explain operator overloading in brief. How operator overloading is used to support polymorphism?

2. What is the benefit of overloading an operator? Write a program to overload unary minus (-) operator.

3. Write a program showing '+ +' and '- -' operator overloading.

4. Write a program to overload multiplication operator (*) showing the multiplication of two objects.

5. Write a program to overload '+' operator to concatenate two strings.

```
#include<string.h>
using namespace std;

class AddString{
  char name[20];
  public:
       AddString(){   //Default Constructor
         cout<<"Enter a string: "<<endl;
         cin>>name;
       }
       AddString operator +(AddString s){
        strcat(name,s.name); //concatenates (adds) two strings
           //and stores in name (data member of object 'str1')
        return *this;
       }
       void display(){
        cout<<"Concatenated String: "<<name<<endl;
       }
};
int main()
{
  AddString str1,str2; //Creating two objects
  AddString str3=str1+str2; //Operator overloading
        //operator function is invoked as 'str1.operator+(str2)'
                  //returned object is stored in object 'str3'
  str3.display();
  return 0;
}
```

Output: Enter a string:
Unbend
Enter a string:
Unwind
Concatenated String: UnbendUnwind

6. Write a program to generate Fibonacci series using ++ operator overloading.

```
#include<iostream>
using namespace std;

class Fibo{
 int a,b,c;
 public:
  Fibo(){ //Default constructor
    a=-1;
    b=1;
    c=0;
  }
  operator ++(){ //operator overloading function
    c=a+b;
    a=b;
    b=c;
  }
  void display(){
   cout<<c<<endl;
  }
};
int main()
{
    int n;
    cout<<"Enter the value of n: "<<endl;
    cin>>n;
    Fibo f; //Creating object
    cout<<"Fibonacci series upto "<<n<<"th term: "<<endl;
    for(int i=1; i<=n; i++){
     ++f;  //operator overloading
        //operator function is invoked as 'f.operator++()'
     f.display();//displaying the term 'c' in each iteration
    }
     return 0;
}
```

Output: Enter the value of n:
6
Fibonacci series upto 6th term:
0
1
1
2
3
5

7. Write a program to add two complex numbers. Your program should have three objects. Each object contains two attributes (i.e real and imaginary part). Now add each attribute and save them into third object separately. Use the concept of '+' operator overloading.

8. Define type conversion. Explain with example the conversion from one class type to another class type.

9. Write a program to read the length of a white board in feet and inches and convert it into meter and cm using user-defined to user-defined type conversion. (1 meter=3.3 feet, 1 feet= 12 inch).

Solution:

Approach-1: (Using casting operator function method)

```
#include<iostream>
using namespace std;

class LengthD{
  int meter;
  float cm;
  public:
  void getData(int m, float c){
   meter=m;
   cm=c;
  }
  void display(){
   cout<<"After conversion,length = "<<meter<<" meter and "<<cm<<" cm."<<endl;
  }
};
class LengthS{
  int feet;
  float inch;
  public:
  void getData(){
   cout<<"Enter the length in feet and inches: "<<endl;
   cin>>feet>>inch;
  }
  operator LengthD(){
   float f=feet+inch/12;//converting feet and inches to total feet
   float m=f/3.3; //converting feet to meter
   int meter= int(m); //ignoring the decimal part
     //for example, if m=5.5 then meter=int(m), that will be 5
   float cm= (m-meter)*100; //converting the fractional part into cm
     //for example, if m=5.5 then cm=(5.5-5)*100, that will be 50

   LengthD temp; //Creating object of Destination class
                 //to store the converted values
   temp.getData(meter,cm);
   return temp; //returns the object temp
                //which is saved in object 'ld' in main() function
  }
};
```

```
int main()
{
  LengthS ls;    //Creating object of source class
  ls.getData(); //initializing the value of feet and inch.
  LengthD ld;  //Creating object of destination class
  ld=ls;  //type conversion // operator function is called
  ld.display(); //displaying the values after conversion
  return 0;
}
```

Output: Enter the length in feet and inches:
18 1.8 (entered by user)
After conversion, length = 5 meter and 50 cm.

Approach- 2: (Using constructor method)

```
#include<iostream>
using namespace std;

class LengthS{
  int feet;
  float inch;
  public:
  void getData(){
   cout<<"Enter the length in feet and inches: "<<endl;
   cin>>feet>>inch;
  }
  int getFeet(){  //function to access feet from outside
    return feet;
  }
  float getInch(){ //function to access inch from outside
    return inch;
  }
};
class LengthD{
  int meter;
  float cm;
  public:
  LengthD(){   //Default constructor
  }
  LengthD(LengthS ls){  //Parameterized constructor
    int feet=ls.getFeet(); //getting the value of feet of source object
    float inch=ls.getInch();//getting the value of inch of source object
    float f=feet+inch/12;
    float m=f/3.3;
    meter= int(m);
    cm= (m-meter)*100;
  }
  void display(){
   cout<<"After conversion,length = "<<meter<<" meter and "<<cm<<" cm."<<endl;
  }
};
```

```
int main()
{
  LengthS ls;    //Creating object of source class
  ls.getData(); //initializing the value of feet and inch.
  LengthD ld;  //Creating object of destination class
  ld=ls;  //type conversion // parameterized constructor is called
  ld.display(); //displaying the values after conversion
  return 0;
}
```

Output: Enter the length in feet and inches:

18 1.8 (entered by user)

After conversion, length = 5 meter and 50 cm.

10. What is type casting? Write a program to read a height of a person in Feet and Inches and convert it into Meter using user-defined to class type conversion method.

11. Write a program to convert an object of Polar class into the object of Rectangle class by using type conversion routine. (hint: polar co-ordinates (radius, angle) and rectangular co-ordinates(x,y), where, x=r*cos(angle) and y=r*sin(angle).