INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

EEC-101
Programming with C++

Module-4:
Object Oriented Programming

About Subject

- Object Oriented Programming Concepts
  - Data hiding,
  - Abstract data types,
  - Classes and Access control;
  - Class Implementation-
    - constructors, default constructor, copy constructor,
    - destructor
  - Operator overloading
  - Friend functions
  - Introduction to Templates

Operator Overloading

- Operator overloading is the ability to tell the compiler how to perform a certain operation when its corresponding operator is used on one or more variables.
- For example, the compiler acts differently with regard to the subtraction operator "-" depending on how the operator is being used.
  - When it is placed on the left of a numeric value such as -99, the compiler considers the number a negative value.
  - When used between two integral values, such as 50-499, the compiler applies the subtraction operation.
  - When the - symbol is doubled and placed on one side of a variable, such as --Variable or Variable--, the value of the variable needs to be decremented; in other words, the value 1 shall be subtracted from it.
  - All of these operations work because the subtraction operator "-" has been reconfigured in various classes to act appropriately.

Operator Overloading

- a1 = a2 + a3;
- The above operation is valid, if a1, a2 and a3 are variables of in-built Data Types.

Operator Overloading

- But what if those are, say objects of a Class; is the operation valid?

- Yes, it is, if we overload the '+' Operator in the class, to which a1, a2 and a3 belong.

Operator Overloading

- Operator overloading is used to give special meaning to the commonly used operators (such as +, -, * etc.) with respect to a class.

- By overloading operators, we can control or define how an operator should operate on data **with respect to a class.**

- Operators are overloaded in C++ by creating operator functions either as a member or as a friend Function of a class.

Operator Overloading

- Operator functions are declared using the following general form:

  ret-type operator # (arg-list);

  and then defining it as a normal member function.

- Here, ret-type is commonly the name of the class itself as the operations would commonly return data (object) of that class type.

- # is replaced by any valid operator

  such as +, -, *, /, ++, -- etc.

- The C++ language recognizes 45 operators of various types.
- Arithmetic operators are used to perform numeric operations.
- The unary operators are used to control the sign or behavior of a variable.
- There are many other binary and C++ specific operators.

- When customized, the arithmetic and any other operators can be applied to varying circumstances.

- When overloaded, the operators are implemented as functions using the operator keyword. For example, the syntax of overloading the addition operator “+” would be operator+() .

- The output operator << used to display anything in C++. Also called as insertion operator.

- The input >> operator is used to retrieve data input. Also called extraction operator.

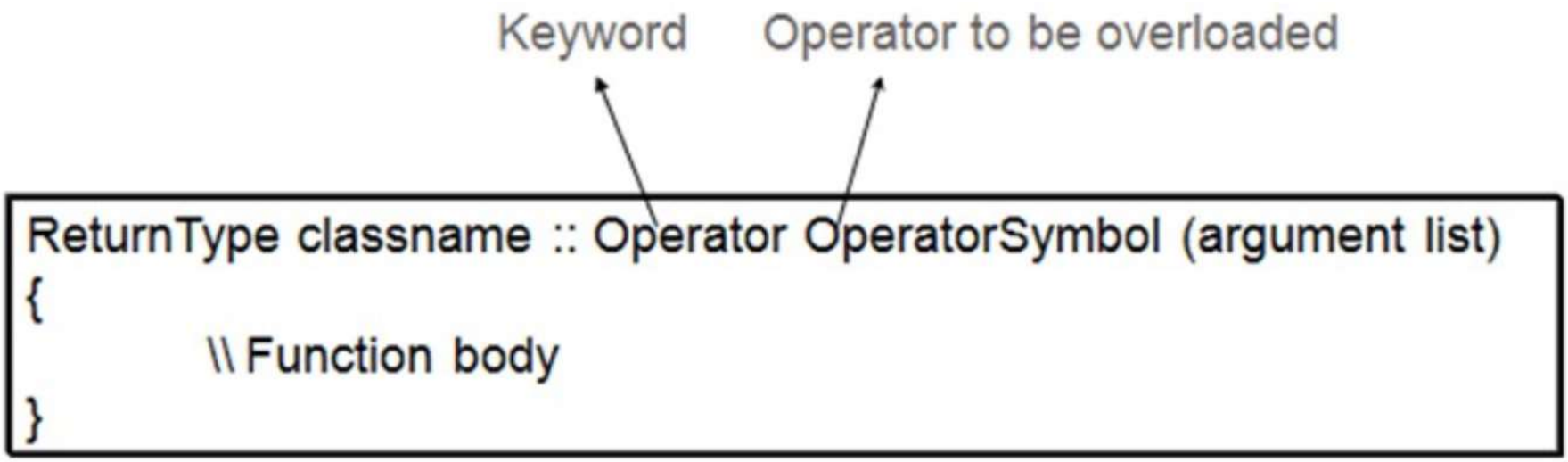
C++ operators that can be overloaded

Operators that can be overloaded	Examples
Binary Arithmetic	+, -, *, /, %
Unary Arithmetic	+, -, ++, —
Assignment	=, +=,*=, /=,-=, %=
Bitwise	& , | , << , >> , ~ , ^
De-referencing	(->)
Dynamic memory allocation, De-allocation	New, delete
Subscript	[ ]
Function call	()
Logical	&, ||, !
Relational	>, < , = =, <=, >=

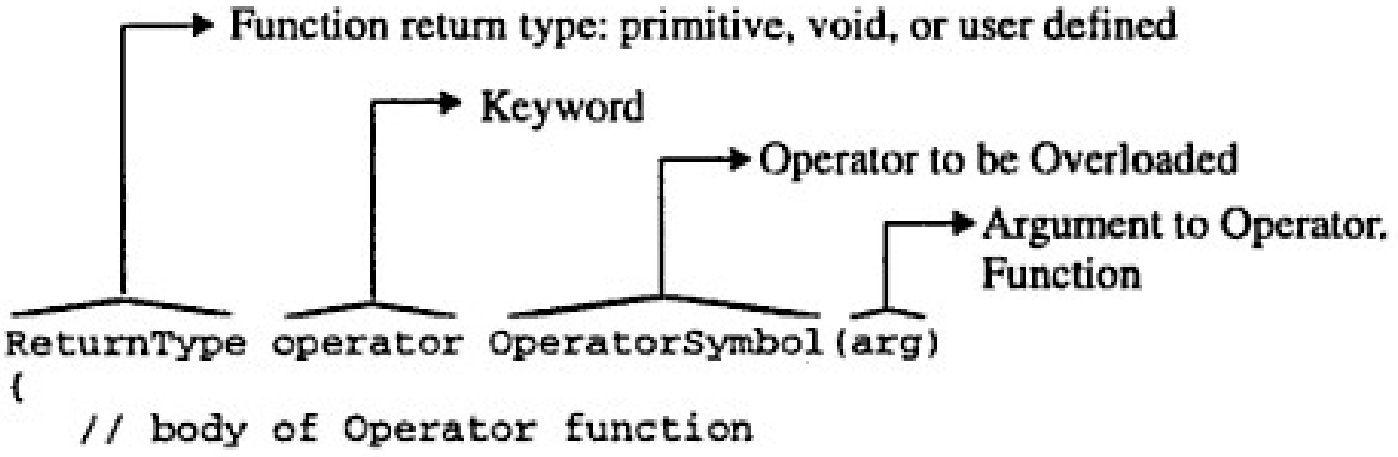
C++ operators that cannot be overloaded

However, there are a few **exceptions**. Operators that cannot be overloaded are:

::	**scope resolution**
.	**Direct member access**
sizeof	**size, in bytes, of an object**
.*	**direct member pointer access**
?:	**conditional**



Syntax of operator overloading



Syntax for Binary operator overloading

The following examples illustrate the overloading of binary operators:

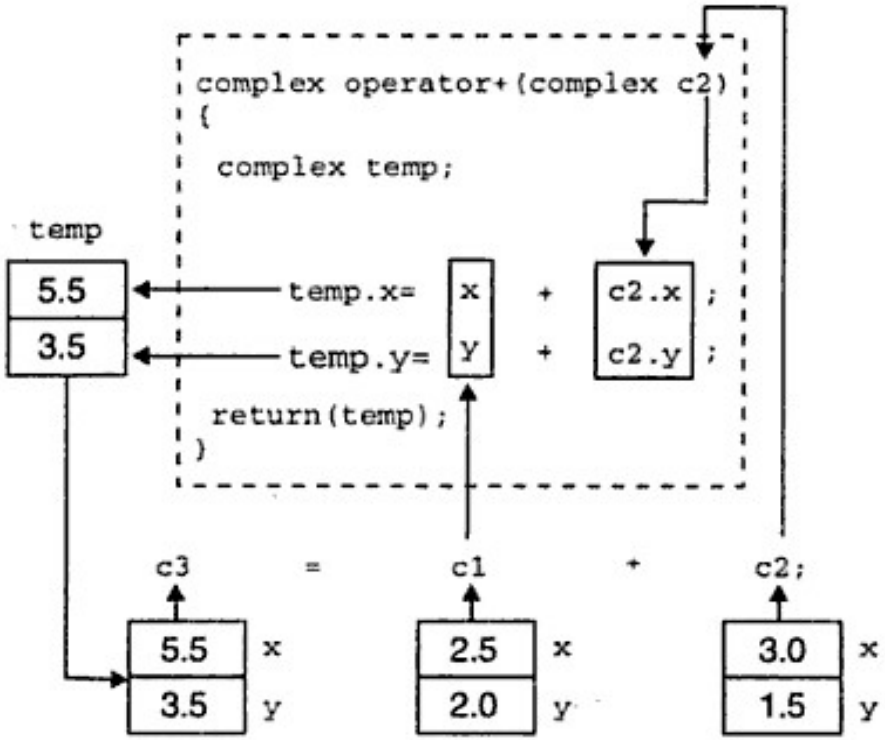
```
complex operator + ( complex c1 );
int operator - ( int a );
void operator * ( complex c1 );
void operator / ( complex c1 );
complex operator += ( complex c1 );
```

Sum of complex numbers without operator overloading

```
1  //Sum of two complex numbers without operator overloading
2  #include <iostream>
3  using namespace std;
4  class Complex
5  {
6     float x,y;
7     public:
8             void getcomplexnumber(float, float);
9             void dispcomplexnumber();
10            Complex sum(Complex, Complex); // declaration with objects as arguments
11 } C1, C2, C3;
12 void Complex::getcomplexnumber(float real, float imag)
13 { x = real; y=imag;}
14 void Complex::dispcomplexnumber()
15 {cout << x <<" + i " << y <<endl ;}
16 Complex Complex::sum(Complex c1, Complex c2)  // c1, c2 are objects
17 {     Complex c3;
18       c3.x = c1.x + c2.x;
19       c3.y = c1.y + c2.y;
20       return (c3);}
21 int main()
22 {
23      C1.getcomplexnumber(2.5, 2.0);
24      C2.getcomplexnumber(3.0, 1.5);
25      C3 = C3.sum(C1,C2);
26      cout << "C1 = "; C1.dispcomplexnumber();
27      cout << "C2 = "; C2.dispcomplexnumber();
28      cout << "C3 = "; C3.dispcomplexnumber();
29    system("pause");
30    return 0;
31    }
```

Sum of complex numbers with operator overloading

```
1  //Sum of two complex numbers through overloaded operator function
2  #include <iostream>
3  using namespace std;
4  class Complex
5  {
6     float x,y;
7     public:
8             void getcomplexnumber(float, float);
9             void dispcomplexnumber();
10            Complex operator+(Complex); // declaration with objects as arguments
11 };
12 void Complex::getcomplexnumber(float real, float imag)
13 { x = real; y=imag;}
14 void Complex::dispcomplexnumber()
15 {cout << x <<" + i " << y <<endl ;}
16 //Overloading of + operator
17 Complex Complex :: operator+(Complex C2)// C2 is passed and invoking object is passed implicitly
18  { Complex temp;
19    temp.x = x + C2.x;   // x is the data member of implicitly passed object
20    temp.y = y + C2.y;   // y is the data memeber of implicitly passed object
21   return temp;}
22 int main(){
23     Complex C1, C2, C3;
24     C1.getcomplexnumber(2.5, 2.0);
25     C2.getcomplexnumber(3.0, 1.5);
26     C3 = C1 + C2; // Sum of two complex numbers through overloaded operator function
27     //C3 = C1.operator+(C2); // C1 is invoking object(hence implicit) and C2 is passed
28     cout << "C1 = "; C1.dispcomplexnumber();
29     cout << "C2 = "; C2.dispcomplexnumber();
30     cout << "C3 = "; C3.dispcomplexnumber();
31   system("pause"); return 0;}
```



Operator overloading in class complex

```
1  //Sum of two complex numbers through overloaded operator function
2  #include <iostream>
3  using namespace std;
4  class Complex
5  {
6     float x,y;
7     public:
8            void getcomplexnumber(float, float);
9            void dispcomplexnumber();
10           Complex operator+(Complex); // declaration with objects as arguments
11 };
12 void Complex::getcomplexnumber(float real, float imag)
13 { x = real; y=imag;}
14 void Complex::dispcomplexnumber()
15 {cout << x <<" + i " << y <<endl ;}
16 //Overloading of + operator
17 Complex Complex :: operator+(Complex C2)// C2 is passed and invoking object is passed implicitly
18  { Complex temp;
19    temp.x = x + C2.x;  // x is the data member of implicitly passed object
20    temp.y = y + C2.y;  // y is the data memeber of implicitly passed object
21   return temp;}
22 int main(){
23     Complex C1, C2, C3;
24     C1.getcomplexnumber(2.5, 2.0);
25     C2.getcomplexnumber(3.0, 1.5);
26     //C3 = C1 + C2; // Sum of two complex numbers through overloaded operator function
27     C3 = C1.operator+(C2); // C1 is invoking object(hence implicit) and C2 is passed
28     cout << "C1 = "; C1.dispcomplexnumber();
29     cout << "C2 = "; C2.dispcomplexnumber();
30     cout << "C3 = "; C3.dispcomplexnumber();
31   system("pause"); return 0;}
```

Operator overloading

- the purpose is to avoid calling of member functions for achieving some operator related tasks on an object – thus providing more concise notation
- through overloading, the built-in operators are used to specify object manipulations
- C++ uses overloading e.g. ++, --, + and – have a different meaning when using pointer arithmetic
- Similarly C++ advises users to attach appropriate meaning to operators for their objects
- Caution - misuse can make the programs difficult to understand

Overloading methodology

- For operator overloading, public member functions have to be introduced in the class
- The name of such a function is the keyword “operator” followed by operator symbol e.g. **operator+**
- Sometimes operators are overloaded through friend functions or non-member functions also
- Operator precedence or associativity or the arity cannot be changed by overloading
- Overloading of, say, + operator will allow writing:
- object2 = object1 + object2;
- which is equivalent to:
- object1.operator+(object2);
- If an operator is overloaded as a member function then in an expression, the left operand of a binary operator (or the only operand of a unary operator) must be a class object

Restrictions on operator overloading

- The precedence of an operator cannot be changed by overloading (parenthesis can be used to force the order of evaluation of overloaded operators in an expression)
- The associativity of an operator cannot be changed by overloading
- It is not possible to change the “arity” of an operator (i.e., the number of operands an operator takes)
- It is not possible to create new operators; only existing operators can be overloaded

- Operator precedence determines which operator will be performed first in a group of operators with different precedences.
- For instance 5 + 3 * 2 is calculated as 5 + (3 * 2), giving 11, and not as (5 + 3) * 2, giving 16.
- The operator associativity rules define the order in which adjacent operators **with the same precedence level** are evaluated.
- For instance the expression 8 - 3 - 2 is calculated as (8 - 3) - 2, giving 3, and and not as 8 - (3 - 2), giving 7.
- In this case we say that subtraction is left associative meaning that the left most subtraction must be done first.
- a = b = c to be interpreted as a = (b = c), thereby setting both a and b to the value of c. The alternative (a = b) = c does not make sense because a = b is not an lvalue.