Object Oriented Programming

Unit 3: Operator Overloading and Type Conversion

Operator Overloading and Type Conversion

CE: 3.1
Introduction to Operator Overloading

CE: 3.1. Introduction to Operator Overloading (Conti…)

- It allows the user to *program the problems* or to *develop the solution of problems* which can be observed in the real world.

- The operator overloading feature of C++ is one of the methods of realizing polymorphism.

polymorphism = poly + morphism

✓**"poly"** refers to many or multiple

✓**"morphism"** refers to actions

Therefore, polymorphism means ***performing many actions with a single operator***.

- **Operator overloading** is a type of polymorphism in which an operator is overloaded to give user defined meaning to it. It is used to perform operation on user-defined data type.

CE: 3.1. Introduction to Operator Overloading (Conti…)

- **Operator function** must be either member function(non-static) or friend function.

Functions	Unary Operator	Binary Operator
Member Function	No Argument	One Operator
Friend Function	One Argument	Two Argument

- This is because *the object used to invoke the member function*. And object is being passed with friend functions.

- Arguments can be passed either by value or by reference.

CE: 3.1. Introduction to Operator Overloading (Conti…)

- Overloaded operators are functions with special names with the keyword **operator** followed by the **symbol** for the operator being defined.

- Like any other function, an overloaded operator has a return type and parameter list.

- Syntax:

```
 return_type operator operator_symbol(argument_list)
        {
                // body
         }
 Return_type class_name :: operator operator_symbol(argument_list)
        {
                // body
         }
```

CE: 3.1. Introduction to Operator Overloading (Conti…)

- Operator functions are declared in the class as…
  - vector operator+(vector); // vector addition
  - vector operator-(); // unary minus

  - friend vector operator+(vector, vector); // vector addition
  - friend vector operator-(vector);// unary minus

  - vector operator – (vector &a);

  - int operator == (vector); // comparison
  - friend int operator == (vector, vector); //comparison

Demonstration

```
#include<iostream>
using namespace std;
class space
    {
                              int x,y,z;
               public:
                              void getdata(int a, int b, int c);
                              void display(void);
                              void operator-();  // declaration
     };
void space :: getdata(int a, int b, int c)
    {
               x=a; y=b; z=c;
     }
void space :: display(void)
    {
               cout<<"x="<<x<<" ";  cout<<"y="<<y<<" ";   cout<<"z="<<z<<" ";
     }
```

Demonstration (Conti…)

```
void space :: operator-()   //Definition
    {
            x =-x; y=-y; z=-z;
     }

int main()
    {
            space s;
            s.getdata(10,-20,30);
            cout<<"S: ";
            s.display();

            -s;
            cout<<"-S :";
            s.display();
     }
```

Rules for Overloading Operators:

1. Only existing operators can be overloaded. New operators cannot be created.
2. The overloaded operator must have at least one operand that is of user-defined type.
3. We cannot change the basic meaning of an operator. Like we cannot redefine the plus(+) operator to subtract one value from the other.
4. There are some operators that cannot be overloaded like…
   - Scope resolution operator - ::
   - Sizeof
   - Member selector - .
   - Member pointer selector - .*
   - Ternary operator - ?:
5. Two operators = and & are already overloaded by default in C++. For example: To copy objects of same class, you can directly use = operator. You do not need to create an operator function.

Rules for Overloading Operators: (Conti…)

- The reason, why we cannot overloaded these operators is, it may be attributed to the fact, that these operators take names(example class name) as their operand instead of values, as it is the case with other normal operators.

CE: 3.1. Introduction to Operator Overloading (Conti…)

- Operator overloading can also be applied for data conversion.
- C++ offers automatic conversion of primitive data types.
- For Example:

  x = a + b;
- Here, the complier indirectly converts the *integer result to floating result* and then *assign to the float variable* ***x***.
- But the conversion of user defined data types requires some effort on the part of the programmer.
- Therefore, the operator overloading concepts are applied on two principle area:
  1. Extending capability of operators to operate on user defined data.
  2. Data conversion.

CE: 3.1. Introduction to Operator Overloading (Conti…)

What is it used?

- You can write any C++ program without the knowledge of operator overloading.

- However, operator operating are greatly used by programmers to make program intuitive (based on feelings rather than facts or proof).

- For Example:
  You can replace the code like:

  calculation = add(multiply(a, b), divide(a, b));

  to

  calculation = (a*b)+(a/b);

CE: 3.2
Unary and Binary Operator Overloading

CE: 3.2. Unary Operator Overloading

- An *Unary operator* is an operator that ***operates on a single operand*** and returns a new value.

- In unary operator function, no arguments should be passed.

- Some of the unary operators are,
  - –   (Unary Minus operator)
  - !   (NOT Operator)
  - + + (Increment Operator)
  - - -   (Decrement operator)
  - etc.

- For primitive types like int, string, etc., these unary operators are already overloaded. Let’s see how to overload a unary operator for a User Defined class.

CE: 3.2. Unary Operator Overloading (Conti…)

- **The unary operators operate on the object** for which they were called.

- It works only with one class objects.

- Normally, this operator appears on the left side of the object as…

  !obj, -obj, and ++obj

  but sometime they can be used as postfix as well like obj++ or obj--.

CE: 3.2. Unary Operator Overloading (Conti…)

```
#include<iostream>
using namespace std;
class test
  {
            int a, b;
        public:
             void getdata(int x, int y)
                {
                        a=-x;
                        b=-y;
                }
            void operator-()
                {
                         cout<<a<<"\n " <<b;
                }
  };
```

```
int main()
  {
           test t;
           t.getdata(10,-20);
           -t;
           return 0;
  }
```

unaryoo.cpp

CE: 3.2. Binary Operator Overloading

- A *Binary operator* is an operator that ***operates on two operands*** and returns a new value.

- In binary operator overloading function, there should be one argument to be passed.

- Very frequently used binary operators are like addition (+) operator, subtraction (-) operator and division (/) operator.

CE: 3.2. Binary Operator Overloading (Conti…)

- Operator functions are declared in the class as…

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

CE: 3.2. Binary Operator Overloading (Conti…)

binaryoo.cpp

binaryoo2.cpp

CE: 3.2.2.
Unary and Binary Operator Overloading using Friend Function

- In this approach, all the working and implementation would be same as unary and binary operator member function; except this function will be implemented outside of the class scope.

- But a friend operator function takes two parameters in a binary operator; and one parameter in a unary operator.

CE: 3.2.2.
Unary and Binary Operator Overloading using Friend Function

For example: Overloading of binary operator using friend function

- Declaration

```
friend complex operator + (complex c1, complex c2);
```

- Definition

```
friend complex operator + (complex c1, complex c2)
  {
         return complex((a.x+b.x),(a.y+b.y));
  }
```

- Function call

```
C3= C1+C2;
```

CE: 3.3
Overloading Special Operators

CE: 3.3.1. Increment and Decrement Operators

- The increment (++) and decrement (--) operators are two important unary operators available in C++.

idoo.cpp

CE: 3.3.2. Subscript Operator

- The subscript operator [] is normally used to access array elements.
- This operator can be overloaded to enhance the existing functionality of C++ arrays.

Demonstration

CE: 3.3.2. Subscript Operator (Conti…)

```
#include<iostream>
using namespace std;
class marks
   {        int subject[3];
        public:
           marks(int sub1, int sub2, int sub3)
             {
                  subject[0] = sub1;
                  subject[1] = sub2;
                  subject[2] = sub3;
              }
           int operator [](int position)
             {
                  return subject[position];
             }
   };
```

```
int main()
  {
       marks bhavik(22, 33, 55);
       cout<<bhavik[1];
       cout<<bhavik[0];
       cout<<bhavik[2];
       return 0;
  }
```

subscriptoo.cpp

CE: 3.3.3. Memory Management Operators

- The **new** and **delete** operators can also be overloaded like other operators in C++.

- Both the operators can be overloaded **globally** or they can be overloaded for specific classes.

- If these operators are overloaded using member function for a class, it means that these operators are overloaded only for that specific class.

- If thee overloading is done outside a class, then the overloaded ‘new’ and ‘delete’ will be called anytime (i.e. that means it will be called within the classes or outside classes). This mechanism is global overloading.

CE: 3.3.3. Memory Management Operators (Conti…)

- Syntax for overloading the **new** operator:

```
void* operator new(size_t size);
```

- The overloaded new operator receives **size of type** **size_t**, which specifies the number of bytes of memory to be allocated.
- The return type of the overloaded new must be….**void***.
- The overloaded function **returns a pointer** to the beginning of the block of memory allocated.

CE: 3.3.3. Memory Management Operators (Conti…)

- Syntax for overloading the **delete** operator:

  void operator delete(void*);

- The function receives a parameter of type void* which has to be deleted. Function should not return anything.

- NOTE: Both overloaded new and delete operator functions are static members by default. Therefore, they don’t have access to this pointer .

CE: 3.3.3. Memory Management Operators (Conti…)

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

class student
  {
               int enro;
               string name;
          public:
              student()
                {
                        cout<< "Constructor is called\n" ;
                }
              student(int e, string n)
                  {
                          enro = e;
                          name = n;        }
```

CE: 3.3.3. Memory Management Operators (Conti…)

Conti…

```
void whoareyou()
    {
        cout<<"\nHi my name is "<<name <<" and enrollment no. is "<<enro;
    }
void* operator new(size_t size)
    {
            cout<<size; // size of overloading new operator
            void* pointer = ::new student();
            //pointer = malloc(size);
            return pointer;
     }
void operator delete(void* pointer)
    {
             cout<< "\nOverloading delete operator " << endl;
             free(pointer);
    }    };
```

CE: 3.3.3. Memory Management Operators (Conti…)

Conti…

```
int main()
  {
        student* s = new student(56, "Bhavik");
        s->whoareyou();
        delete s;
        return 0;
  }
```

CE: 3.4
Overloading << and >> Operators

CE: 3.4. Overloading << and >> Operators

- In C++, stream insertion operator << is used for output and
  stream extraction operator >> is used for input.

- cout is an object of ostream class and cin is an object istream class, which is a compiler defined class.

- When we do cout<<obj, where obj is an object of our class, the compiler first looks for an operator function in ostream, then it looks for a global function.

- One way to overload insertion operator is to modify ostream class which may not be a good idea.
- So the operators must be overloaded as a global function; and if we want to allow them to access private data members of class, we must make them friend.

CE: 3.4. Overloading << and >> Operators (Conti…)

- Therefore, these operators are overloaded as global functions with two parameters, cout and object of user defined class.
- Overloading operator << and operator >> make it extremely easy to output your class to screen and accept user input from the console.

CE: 3.4. Overloading << and >> Operators (Conti…)

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

class student
    {
                int enro;
                char name[20];
            public:
                friend void operator >>(istream &in, student &s)
                    {
                        cout<<"Enter the enro : ";
                        in>>s.enro;
                        cout<<"Enter the name: ";
                        in>>s.name;
                    }
```

CE: 3.4. Overloading << and >> Operators (Conti…)

Conti…

```
                friend void operator <<(ostream &out, student &s)
                      {
                                  cout<<"Enro : ";
                                  out<<s.enro;
                                  cout<<"Name: ";
                                  out<<s.name;
                       }
  };

int main()
   {
               student s1;
               cin>>s1;
               cout<<s1;
               return 0;
   }
```

International Certification Question

Q1.	**Which of the following is not an operator overloaded by the C++ language?**
A.	**Pow**
B.	>>
C.	+
D.	<<

Ans: A

International Certification Question

Q2.	**The correct function name for overloading the addition (+) operator is:**
A.	**operator_+**
B.	**operator:+**
C.	**operator+**
D.	**operator(+)**

Ans: C

International Certification Question

Q3.	Which of the following operators cannot be overloaded?
A.	The . operator
B.	The -> operator
C.	The [ ] operator
D.	The & operator

Ans: A and D

International Certification Question

Q4.	**Which statement about operator overloading is false?**
A.	**New operators can never be created.**
B.	**Certain overloaded operators can change the number of arguments they take.**
C.	**The precedence of an operator cannot be changed by overloading.**
D.	**Overloading cannot change how an operator works on built-in types.**

Ans: A

International Certification Question

Q5.	To implicitly overload the += operator:
A.	Only the = operator needs to be overloaded.
B.	Only the + operator needs to be overloaded.
C.	The += operator cannot be overloaded implicitly.
D.	Both the + and = operators need to be overloaded.

Ans: D

International Certification Question

Q6.	Which of the following operators can be overloaded as a global function?
A.	()
B.	= =
C.	+ =
D.	[]

Ans: A

International Certification Question

Q7.	Which situation would require the operator to be overloaded as a global function?
A.	The left most operand must be a class object (or a reference to a class object).
B.	The left operand is an int.
C.	The operator returns a reference.
D.	The overloaded operator is =.

Ans: A

International Certification Question

Q8.	If we overload unary operator by means of making function friend, ________________ arguments required.
A.	Two
B.	One
C.	No arguments
D.	None

Ans: B

International Certification Question

Q9.	Which of the following operators can be overloaded?
A.	sizeof
B.	.*
C.	::
D.	+

Ans: D

Industry Interview Questions

1. What is operator overloading?
2. Why operator overloading is needed?
3. What is the significance of operator overloading?
4. What is unary operator?

Home Work

1. What is operator overloading? [1 Mark]
2. How any operator can be overloaded? Give example.[2 Marks]
3. How many arguments are required in the definition of member function and friend function to an overloaded unary operator? [1 Mark]
4. Explain the overloading of unary operator with example. [5 Marks]
5. Explain binary operator overloading with example. [5 Marks]
6. Explain binary operator overloading using friend function with example. [5 Marks]
7. What the rules of operator overloading? [5 Marks]

CE: 3.5
Type Conversion

CE: 3.5. Type Conversion

- A type conversion is basically a conversion from one type to another.
- There are two types of type conversion:
  1. Implicit Type Conversion
  2. Explicit Type Conversion

1. Implicit Type Conversion:

- It is also known as 'automatic type conversion'.
- It is done by the compiler on its own, without any external trigger from the user.
- This conversion takes place when in an expression, there are more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.

CE: 3.5. Type Conversion (Conti…)

1. Implicit Type Conversion: (Conti…)

- All the data types of the variables are upgraded to the data type of the variable with largest data type.

- bool -> char -> short int -> int -> unsigned int -> long -> unsigned -> long long -> float -> double -> long double

- It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

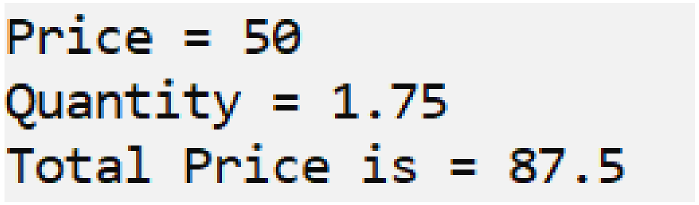
CE: 3.5. Type Conversion (Conti…)

```
//Implicit conversion
#include <iostream>
using namespace std;

int main()
 {
     int price = 50;
     float qty = 1.75;

     float totalprice = price * qty;

     cout << "Price = " << price;
     cout << "\nQuantity = " << qty;
     cout << "\nTotal Price is = " << totalprice;
     return 0;
 }
```



```
Price = 50
Quantity = 1.75
Total Price is = 87.5
```

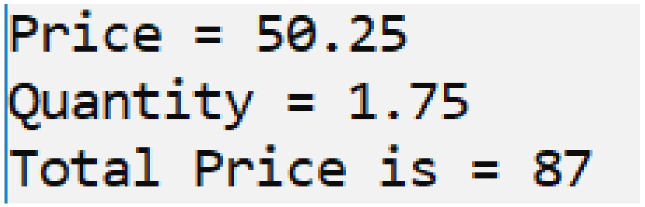
CE: 3.5. Type Conversion (Conti…)

```
//Implicit conversion
#include <iostream>
using namespace std;

int main()
 {
      float price = 50.25;
      float qty = 1.75;

      int totalprice = price * qty;

      cout << "Price = " << price;
      cout << "\nQuantity = " << qty;
      cout << "\nTotal Price is = " << totalprice;
      return 0;
  }
```



```
Price = 50.25
Quantity = 1.75
Total Price is = 87
```

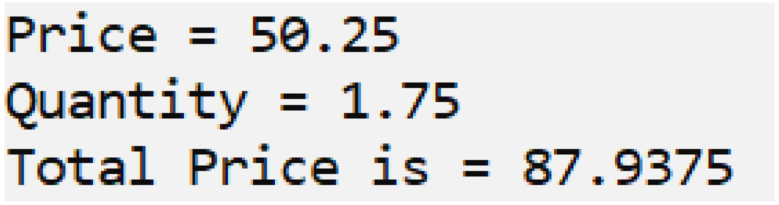
CE: 3.5. Type Conversion (Conti…)

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

int main()
  {
      float price = 50.25;
      float qty = 1.75;

      float totalprice = price * qty;

      cout << "Price = " << price;
      cout << "\nQuantity = " << qty;
      cout << "\nTotal Price is = " << totalprice;
      return 0;
   }
```



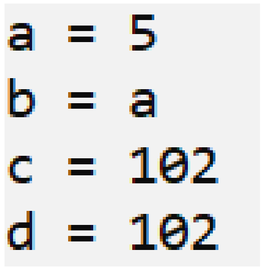
```
Price = 50.25
Quantity = 1.75
Total Price is = 87.9375
```

CE: 3.5. Type Conversion (Conti…)

```
//Implicit conversion
#include <iostream>
using namespace std;

int main()
  {
       int a = 5;
       char b = 'a';

       int c = a + b; // b implicitly converted to int ASCII value of 'a' is 97
       float d = a + b; // a is implicitly converted to float
       cout << "a = " << a;
       cout << "\nb = " << b;
       cout << "\nc = " << c;
       cout << "\nd = " << d;
       return 0;
   }
```



```
a = 5
b = a
c = 102
d = 102
```

CE: 3.5. Type Conversion (Conti…)

2. Explicit Type Conversion: (Conti…)

- It is also known as ‘type casting’.
- It is an user-defined process.
- Here, the user can typecast the result to make it of a particular data type.
- Syntax:

  (type) expression

  ✓ where type indicates the data type to which the final result is converted.

CE: 3.5. Type Conversion (Conti…)

```
//Explicit Type Casting
#include <iostream>
using namespace std;

int main()
  {
        double x = 1.5;

        int sum = (int)x + 1;   // Explicit conversion from double to int

        cout << "Sum = " << sum;
        return 0;
  }
```

```
Sum = 2
```

CE: 3.6
User Defined Conversion

CE: 3.6.1. Basic Type to Class Type

How data from basic data type is converted to class type?

- The assignment operator (=) is required for both of its operand with some of its basic type or objects.
- With “=“ operator the value of right side operand is assigned to left side operand.
- By writing both operands of operator of different data types one is basic and other is object which is know as *type conversion*.
- We can ***convert data type to object*** by writing ***one argument constructor***. And ***the argument of constructor will be the basic data type***.
- This constructor is invoked, when we assign some basic data to object type.

CE: 3.6.1. Basic Type to Class Type (Conti…)

```
#include<iostream>
using namespace std;
class length
   {
          int meter, cm;
       public:
          length()
            {     meter=cm=0;     }
          length(int a) // one argument constructor
            {
                 meter = a/100;
                 cm = a % 100;
             }
          void display()
            {      cout<<"\nMeter : "<<meter;
                   cout<<"\nCentimeters: "<<cm;
             }    };
```

```
int main()
   {
         length l1;
         int x=560;
         length l2=x;
         length l3=1050;
         l1.display();
         l2.display();
         l3.display();
         return 0;
   }
```

Type Conversion

typetoclass.cpp

CE: 3.6.2. Class Type to Basic Type

How data from class type is converted to basic data type?

- Here, left operand of operator is basic data type and right operand is object.

- The function through which this conversation occurs is known as conversion function. This conversion function is invoked, when we assign object to basic data type.

- There are few rules for conversion function…
  1. It must be a member function.
  2. It must not specify return type.
  3. It must not take any argument.

Demonstration

CE: 3.6.2. Class Type to Basic Type (Conti…)

```
#include<iostream>
using namespace std;
class length
   {
            int meter, cm;
        public:
            length()
               {     meter=cm=0;     }
            length(int m, int c)
               {
                       meter = m;
                       cm = c;
                }
            operator int() // conversion function
               {
                     int c;
                     c = meter*100;
                     c = c + cm;
                     return c;
                  }
               };
int main()
   {
          length l1(5, 50);
          int x = l1;
          cout<<x;
          return 0;
   }
```

classtotype.cpp

CE: 3.6.3. One Class Type to Another Class Type

How data from one class type is converted to another class type?

- It can be done by two ways:

1. The conversion routine is written into source class from which we want to convert. i.e. it's object is written on right side of = operator.

2. The conversion routine is written in destination class in form of one argument and that will be object of source class.

CE: 3.6.3. One Class Type to Another Class Type (Conti…)

classtoclass.cpp

Thank you