CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

CE142: OBJECT ORIENTED PROGRAMMING WITH C++

December – April 2019

Unit – 08

Operator Overloading and Type Conversions

DEPSTAR

Devang Patel Institute of Advance Technology and Research

Introduction

- It is one of the many exciting features of C++.
- C++ has ability to provide the operators with a special meaning for a data types.
- We can overload (give additional meaning to) all the C++ operators except:
  - Class member access operators (. & .*)
  - Scope resolution operators (: :)
  - Size operator (sizeof)
  - Conditional operators (? :)
- When an operator is overloaded, its original meaning is not lost.

Defining Operator Overloading

- To define an additional task to an operator, we must specify what it means in relation to the class to which the operator is applied.
- This is done with the help of a special function called operator function.

```
return type class-name : :operator op (arg-list)
{
    Function body   //  task defined
}
```

Defining Operator Overloading

```
return type class-name : :operator op (arg-list)
{
    Function body   //  task defined
}
```

- return type is the type of value returned by the specified operation.
- op is the operator being overloaded.
- op is preceded by the keyword operator.
- operator op is the function name.

Defining Operator Overloading

- Operator Function must be either
  - member function (non-static) Or
  - friend function.
  - The basic difference :
    - A friend function will have only one argument for unary operators and two for binary operators.
    - A member function has no arguments for unary operators and one argument for binary operators.
    - This is because the object used to invoke the member function is passed implicitly and therefore is available for the member function.
    - Arguments may be passed either by value or by reference.

Process of Operator Overloading

- The process of overloading involves the following steps:
  - Create a class that defines the data type that is to be used in the overloading operation.
  - Declare the operator function operator op( ) in the public part of the class. It may be either a member function or a friend function.
  - Define the operator function to implement the required operations.

Process of Operator Overloading

- Overloaded operator functions can be invoked by expressions such as:
  - For unary operators: op x or x op
  - For binary operators: x op y
- op x or x op would be interpreted as
  - for a friend function: operator op (x)
  - for a member function: x.operator op ( )
- x op y would be interpreted as
  - for a friend function: operator op (x,y)
  - for a member function: x.operator op (y)

Overloading Unary Operators

- Consider a unary minus operator:
  - It takes just one operand.
  - It changes the sign of an operand when applied to a basic data item.
  - For ex: The unary minus when applied to an object should change the sign of each of its data items.

Example

```
#include <iostream>
class ComplexNumber
{
    int real;
    int imaginary;
public:
    ComplexNumber() :real(0), imaginary(0)
    {}
    ComplexNumber(int r, int i) :real(r), imaginary(i)
    {}
    void print()
    {
        int img = imaginary < 0 ? -imaginary : imaginary;
        std::cout << real << (imaginary < 0 ? " - ": " + ");
        std::cout << "i" << img  << std::endl;
    }
    // Overloaded unary minus operator
    ComplexNumber operator-() const;
};
```

```
ComplexNumber ComplexNumber::operator-() const
{
    return ComplexNumber(-(this->real), -(this->imaginary) );
}

int main()
{
    // Create a Complex Number Object
    ComplexNumber c1(2, -3);
    std::cout<<"c1 = ";
    c1.print();
    // Call the unary operator minus on c1 and
    // store the returned in a new object
    ComplexNumber c2 = -c1;

    std::cout<<"c2 = ";
    c2.print();

    return 0;
}
```

Output:

```
c1 = 2 - i3
c2 = -2 + i3
```

Overloading Binary Operators

- As a rule, in overloading binary operators,
  - the left-hand operand is used to invoke the operator function and
  - the right-hand operand is passed as an argument.

return complex((x+c.x), (y+c.y));

- The compiler invokes an appropriate constructor, initializes an object with no name and returns the contents for copying into an object.
- Such an object is called a temporary object and goes out of space as soon as the contents are assigned to another object.

```
#include< iostream.h>
#include< conio.h>
class time
{
 int h,m,s;
 public:
  time()
  {
   h=0, m=0; s=0;
  }
  void getTime();
  void show()
  {
   cout<< h<< ":"<< m<< ":"<< s;
  }
 time operator+(time);    //overloading '+' operator
};
```

```
time time::operator+(time t1)    //operator function
{
 time t;
 int a,b;
 a=s+t1.s;
 t.s=a%60;
 b=(a/60)+m+t1.m;
 t.m=b%60;
 t.h=(b/60)+h+t1.h;
 t.h=t.h%12;
 return t;
}
void time::getTime()
{
   cout<<"\n Enter the hour(0-11) ";
   cin>>h;
   cout<<"\n Enter the minute(0-59) ";
   cin>>m;
   cout<<"\n Enter the second(0-59) ";
   cin>>s;
}
```

```
void main()
{
 clrscr();
 time t1,t2,t3;
 cout<<"\n Enter the first time ";
 t1.getTime();
 cout<<"\n Enter the second time ";
 t2.getTime();
 t3=t1+t2;  //adding of two time object using '+' operator
 cout<<"\n First time ";
 t1.show();
 cout<<"\n Second time ";
 t2.show();
 cout<<"\n Sum of times ";
 t3.show();
 getch();
}
```

Output

```
Enter the first time
Enter the hour(0-11) 4

Enter the minute(0-59) 15

Enter the second(0-59) 45

Enter the second time
Enter the hour(0-11) 5

Enter the minute(0-59) 23

Enter the second(0-59) 23

First time 4:15:45
Second time 5:23:23
Sum of times 9:39:8
```

Overloading Binary Operators Using Friends

- Friend function requires two arguments to be explicitly passes to it.
- Member function requires only one.

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

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

- We can use a friend function with built-in type data as the left-hand operand and an object as the right-hand operand.

```
#include<iostream.h>
#include<conio.h>
class time
{
 int hr,min,sec;
 public:
  time()
  {
   hr=0, min=0; sec=0;
  }
  time(int h,int m, int s)
  {
   hr=h, min=m; sec=s;
  }
 friend ostream& operator << (ostream &out, time &tm);
 //overloading '<<' operator
};
```

```
//operator function
ostream& operator<< (ostream &out, time &tm)
{
 out << "Time is " << tm.hr << "hour : "
 << tm.min << "min : " << tm.sec << "sec";
 return out;
}

void main()
{
 time tm(3,15,45);
 cout << tm;
}
```

Output

```
Time is 3hour : 15min : 45sec
```

Manipulation of Strings using Operators

- There are lot of limitations in string manipulation in C as well as in C++.
- Implementation of strings require character arrays, pointers and string functions.
- C++ permits us to create our own definitions of operators that can be used to manipulate the strings very much similar to other built-in data types.
- ANSI C++ committee has added a new class called string to the C++ class library that supports all kinds of string manipulations.

Manipulation of Strings using Operators

- Strings can be defined as class objects which can be then manipulated like the built-in types.
- Since the strings vary in size, we use new to allocate memory for each string and a pointer variable to point to the string array.

Manipulation of Strings using Operators

- We must create string objects that can hold two pieces of information:
    - Length
    - Location

```
class string
{
    char  *p;      // pointer to string
    int    len;    // length of string
  public :
    ------
    ------
};
```

```
#include<iostream>
#include<string.h>
using namespace std;
class String
{
    public:
            char str[20];
    public:
            void accept_string()
            {
                cout<<"\n Enter String                    :    ";
                cin>>str;
            }
            void display_string()
            {
                cout<<str;
            }
            String operator+(String x)  //Concatenating String
            {
                String s;
                strcat(str,x.str);
                strcpy(s.str,str);
                return s;
            }
};
```

```
int main()
{
    String str1, str2, str3;

    str1.accept_string();
    str2.accept_string();

    cout<<"\n ----------------------------------------";
    cout<<"\n\n First String is              :  ";
    str1.display_string();   //Displaying First String

    cout<<"\n\n Second String is             :  ";
    str2.display_string();  //Displaying Second String

    cout<<"\n ----------------------------------------";
    str3=str1+str2;          //String is concatenated. Overloaded '+' operator
    cout<<"\n\n Concatenated String is    :  ";
    str3.display_string();

    return 0;
}
```

Output:

```
Enter String                :   DEPSTAR_
Enter String                :   CHARUSAT
----------------------------------------
First String is             :  DEPSTAR_

Second String is            :  CHARUSAT
----------------------------------------
Concatenated String is      :  DEPSTAR_CHARUSAT
```

Rules For Overloading Operators

- Only existing operators can be overloaded. New operators cannot be created.
- The overloaded operator must have at least one operand that is of user-defined type.
- We cannot change the basic meaning of an operator.
- Overloaded operators follow the syntax rules of the original operators.

Rules For Overloading Operators

- The following operators that cannot be overloaded:
  - Size of Size of operator
  - . Membership operator
  - .* Pointer-to-member operator
  - : : Scope resolution operator
  - ? ; Conditional operator

Rules For Overloading Operators

- The following operators can be over loaded with the use of member functions and not by the use of friend functions:
  - Assignment operator =
  - Function call operator( )
  - Subscripting operator [ ]
  - Class member access operator ->
- Unary operators, overloaded by means of a member function, take no explicit arguments and return no explicit values, but, those overloaded by means of a friend function, take one reference argument.

Rules For Overloading Operators

- Binary operators overloaded through a member function take one explicit argument and those which are overloaded through a friend function take two explicit arguments.
- When using binary operators overloaded through a member function, the left hand operand must be an object of the relevant class.
- Binary arithmetic operators such as +, -, * and / must explicitly return a value. They must not attempt to change their own arguments.

Type Conversions

- The type conversions are automatic only when the data types involved are built-in types.
  - int m;
  - float x = 3.14159;
  - m = x; // convert x to integer before its value is assigned
  - // to m.
- For user defined data types, the compiler does not support automatic type conversions.
- We must design the conversion routines by ourselves.

Type Conversions

- Different situations of data conversion between incompatible types.
  - Conversion from basic type to class type.
  - Conversion from class type to basic type.
  - Conversion from one class type to another class type.

Basic to Class Type

- A constructor to build a string type object from a char * type variable.

```
string : : string(char *a)
{
  length = strlen(a);
  P = new char[length+1];
  strcpy(P,a);
}
```

- The variables length and p are data members of the class string.

Basic to Class Type

```
string s1, s2;

string name1 = "IBM PC";

string name2 = "Apple Computers";

s1 = string(name1);

s2 = name2;
```

First converts name1 from char* type to string type and then assigns the string type value to the object s1.

First converts name2 from char* type to string type and then assigns the string type value to the object s2.

Basic to Class Type

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

time T1;
int duration = 85;
T1 = duration;

```cpp
#include<iostream>
using namespace std;
class Time
{
    int hrs,min;
    public:
        void display();
        void operator=(int); // overloading function
};
void Time::display()
{
    cout<<hrs<< ": Hour(s) "<<endl ;
    cout<<min<<": Minutes"<<endl ;
}

void Time::operator=(int t)
{
    cout<<"Basic Type to ==> Class Type Conversion..."<<endl;
    hrs=t/60;
    min=t%60;
}
```

```
int main()
{

    Time t1;
    int duration;
    cout<<"Enter time duration in minutes";
        cin>>duration;
    cout<<"object t1 overloaded assignment..."<<endl;
    t1=duration;
    t1.display();
    cout<<"object t1 assignment operator 2nd method..."<<endl;
    t1.operator=(duration);
    t1.display();
    return 0;
}
```

Output:

```
Enter time duration in minutes410
object t1 overloaded assignment...
Basic Type to ==> Class Type Conversion...

6: Hour(s)

50: Minutes

object t1 assignment operator 2nd method...
Basic Type to ==> Class Type Conversion...

6: Hour(s)

50: Minutes
```

Class To Basic Type

- A constructor function do not support type conversion from a class type to a basic type.
- An overloaded casting operator is used to convert a class type data to a basic type.
- It is also referred to as conversion function.

```
operator typename( )
{
   ...
   ... ( function statements )
   ...
}
```

- This function converts a class type data to typename.

Class To Basic Type

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

- This function converts a vector to the square root of the sum of squares of its components.

Class To Basic Type

- The casting operator function should satisfy the following conditions:
  - It must be a class member.
  - It must not specify a return type.
  - It must not have any arguments.

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

Class To Basic Type

- Conversion functions are member functions and it is invoked with objects.
- 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.

```cpp
#include <iostream>
using namespace std;
class Time
{
    int hrs,min;
    public:
        Time(int ,int);    // constructor
        operator int();    // casting operator function
        ~Time()            // destructor
        {
            cout<<"Destructor called..."<<endl;
        }
};

Time::Time(int a,int b)
{
    cout<<"Constructor called with two parameters..."<<endl;
    hrs=a;
    min=b;
}

Time :: operator int()
{
    cout<<"Class Type to Basic Type Conversion..."<<endl;
    return(hrs*60+min);
}
```

```
int main()
{
    int h,m,duration;
    cout<<"Enter Hours ";
    cin>>h;
    cout<<"Enter Minutes ";
    cin>>m;
    Time t(h,m);          // construct object
    duration = t;         // casting conversion OR duration = (int)t
    cout<<"Total Minutes are "<<duration;
    cout<<"2nd method operator overloading "<<endl;
    duration = t.operator int();
    cout<<"Total Minutes are "<<duration;
    return 0;
}
```

Output:

```
Enter Hours 12

Enter Minutes 12
 Constructor called with two parameters...
 Class Type to Basic Type Conversion...

Total Minutes are 732

2nd method operator overloading
 Class Type to Basic Type Conversion...
Total Minutes are 732

 Destructor called...
```

One Class To Another Class Type

objX = objY ; // objects of different types

- 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.
- Conversion is takes place from class Y to class X.
- Y is known as source class.
- X is known as destination class.

One Class To Another Class Type

- Conversion between objects of different classes can be carried out by either a constructor or a conversion function.
- Choosing of constructor or the conversion function depends upon where we want the type-conversion function to be located in the source class or in the destination class.

One Class To Another Class Type

operator typename( )

- Converts the class object of which it is a member to typename.
- The typename may be a built-in type or a user-defined one.
- In the case of conversions between objects, typename refers to the destination class.
- When a class needs to be converted, a casting operator function can be used at the source class.
- The conversion takes place in the source class and the result is given to the destination class object.

One Class To Another Class Type

- Consider a constructor function with a single argument
  - Construction function will be a member of the destination class.
  - The argument belongs to the source class and is passed to the destination class for conversion.
  - The conversion constructor be placed in the destination class.

```
#include <iostream>
using namespace std;
class Time
{
    int hrs,min;
    public:
        Time(int h,int m)
        {
            hrs=h;
            min=m;
        }
        Time()
        {
            cout<<"\n Time's Object Created";
        }
        int getMinutes()
        {
            int tot_min = ( hrs * 60 ) + min ;
                        return tot_min;
        }
        void display()
        {
            cout<<"\n Hours: "<<hrs<<endl ;
            cout<<" Minutes : "<<min <<endl ;
        }
};
```

```
class Minute
{
    int min;
    public:
        Minute(){
            min = 0;
        }
    void operator=(Time T){
        min=T.getMinutes();
    }
    void display(){
        cout<<"\n Total Minutes : " <<min<<endl;
    }
};
int main()
{
    Time t1(2,30);
        t1.display();
    Minute m1;
        m1.display();
    m1 = t1;     // conversion from T
    t1.display();
    m1.display();
    return 0;
}
```

Output:

```
Hours: 2
Minutes : 30

Total Minutes : 0

Hours: 2
Minutes : 30

Total Minutes : 150
```

Presentation Prepared By:

Ms. Khushi Patel

Contact us:

dweepnagarg.ce@charusat.ac.in
parthgoel.ce@charusat.ac.in
hardikjayswal.it@charusat.ac.in
dipakramoliya.ce@charusat.ac.in
krishnapatel.ce@charusat.ac.in
khushipatel.ce@charusat.ac.in

Subject Teachers:

Ms. Dweepna Garg
Subject Coordinator

Mr. Parth Goel
https://parthgoelblog.wordpress.com

Mr. Hardik Jayswal

Ms. Krishna Patel

Thank You