Lecture 2

Operator Overloading- Part2

University of Anbar
College of Computer Science and Information Technology
Department of Computer Science
Object Oriented Programming
Second Class
Dr. Ruqayah R. Al-Dahhan

Example1: An example of(++ operator and - - operator) functions (member functions) that operate on the object of **Check** class (object **m** in this case).

Sol:

```
#include <iostream>
using namespace std;
class Check
{ private:
     int count;
  public:
    Check(){count=5;}
    void operator ++()
       {   count = count+1;       }
     void Display()
       { cout<<"Count: "<<count<< endl; }
      void operator --()
       {       count= count-1;   }
main()
{    Check m;
     ++ m ;       // calls "operator ++()" function
     m.Display();
      -- m ;
     m.Display();
   }
```

Output:

Count: 6
Count: 5

Note:

++ operator operates on object **m** to increase the value of **data member count** by **1**.

- - operator operates on object **m** to decrease the value of **data member count** by 1.

Overloading Binary Operators Using Friend Functions

- *Must precede with friend keyword, and declare a function class scope.*

```
Friend returnType operator*(parameters);
```

any type *keyword* *operator symbol*

- *Keeping in mind, friend operator function takes **two parameters** in a **binary operator**, varies **one parameter** in a **unary operator**.*
- *All the working and implementation would same as binary operator function(Member Function) except this function will be implemented outside of the class scope.*

Example2: An example of (+,-,*,/,==,>,<,unary -,>>,<< operator) functions (**Friend** functions) that operate on objects of **Money** class .

```
#include <cmath>
#include <iostream>
using namespace std;
class Money                    // Class for amounts of money in US currency
{
  friend const Money operator+(const Money& amount1, const Money& amount2);
  friend const Money operator-(const Money& amount1, const Money& amount2);
  friend const Money operator*(const Money& amount1, const Money& amount2);
  friend const Money operator/(const Money& amount1, const Money& amount2);
  friend bool operator==(const Money& amount1, const Money& amount2);
 friend bool operator>(const Money& amount1, const Money& amount2);
  friend bool operator<(const Money& amount1, const Money& amount2);
  friend const Money operator-(const Money& amount);
  friend istream& operator>>(istream& istr, Money& amount);
  friend ostream& operator<<(ostream& ostr, const Money& amount);
public:
  Money();
  Money(int theDollars);
  Money(double amount);
  Money(int theDollars, int theCents);
private:
  int dollars;
  int cents;
  int dollarsPart(double amount) const; //private member function
  int   centsPart(double amount) const; //private member function
 int      round(double number) const; //private member function

};
```

```
//Addition operator
const Money operator+(const Money& amount1, const Money& amount2)
{                int finalDollars= amount1.dollars+ amount2.dollars;
                 int finalCents= amount1.cents+ amount2.cents;
                return Money(finalDollars, finalCents);
}
//Subtraction operator
const Money operator-(const Money& amount1, const Money& amount2)
{   int finalDollars= amount1.dollars- amount2.dollars;
    int finalCents= amount1.cents- amount2.cents;
    return Money(finalDollars, finalCents);
}
//Multiplication operator
const Money operator*(const Money& amount1, const Money& amount2)
{    int finalDollars= amount1.dollars * amount2.dollars;
      int finalCents= amount1.cents * amount2.cents;
      return Money(finalDollars, finalCents);
}
```

```
//Division operator
const Money operator/(const Money& amount1, const Money& amount2)
{  int finalDollars= amount1.dollars / amount2.dollars;
   int finalCents= amount1.cents / amount2.cents;
   return Money(finalDollars, finalCents);
}
//Equal to operator
bool operator==(const Money& amount1, const Money& amount2)
{  return ( (amount1.dollars  == amount2.dollars) && (amount1.cents == amount2.cents) );
}
//Greater than operator
bool operator>(const Money& amount1, const Money& amount2)
{   return ( (amount1.dollars  > amount2.dollars)&& (amount1.cents > amount2.cents) );


}
```

```
//Less than operator
bool operator<(const Money& amount1, const Money& amount2)
{   return ( (amount1.dollars   < amount2.dollars)&& (amount1.cents < amount2.cents) );}
//Unary subtraction operator
const Money operator-(const Money& amount)
{   return Money(-amount.dollars, -amount.cents);
}
istream& operator>>(istream& istr, Money& amount)
{   char dollarSign;
  istr >> dollarSign;
  if (dollarSign != '$') // if (strcmp(dollarSign, '$') == 0)
    {     cout << "No dollar sign in Money input. \n";
        exit(1);    }
  double amountAsDouble;
  istr >>  amountAsDouble;
  amount.dollars = amount.dollarsPart(amountAsDouble);
  amount.cents    = amount.centsPart(amountAsDouble);
  return istr;  }
```

```
 ostream& operator<<(ostream& ostr, const Money& amount)
{   int absDollars = abs(amount.dollars);
    int absCents    = abs(amount.cents);
    ostr << "Account balance: ";
   if (amount.dollars < 0 || amount.cents < 0)
    ostr << "$";
   else
    ostr << '$' << amount.dollars;
    if (absCents >= 10)
    ostr << "." << absCents << endl;
   else
    ostr << "." << '0' << absCents << endl;
  return ostr;
}
// Constructors
Money :: Money()
{   dollars = 0.0;
    cents    = 0.0;}
```

```
Money :: Money(double amount)
{  dollars = dollarsPart(amount);
   cents   = centsPart(amount);
}
Money :: Money(int theDollars)
{  dollars = theDollars;
   cents   = 0.0;
}
Money :: Money(int theDollars, int theCents)
{  if ( (theDollars < 0 && theCents > 0) ||(theDollars > 0 && theCents < 0) )
   {    cout << "Inconsistent money data.\n";
        exit(1);
   }
 dollars = theDollars;
 cents   = theCents;
}
// Private member functions
int Money :: dollarsPart(double amount) const
{  return static_cast<int> (amount);            //Use <cmath>            }
```

```
 int Money :: centsPart(double amount) const
{  double doubleCents = amount * 100;
   int intCents       =  (round(fabs(doubleCents)) );
   if (amount < 0)  intCents = -intCents;
   return (intCents % 100);  //Return the amount in 'cents'
}
int Money :: round(double number) const
{ return static_cast<int> (floor (number+0.5)); //Use <cmath>
}
int main()
{  Money baseAmount(100, 60), fullAmount;
   fullAmount = baseAmount + 25;
    cout << fullAmount << endl;
   //fullAmount = 25 + baseAmount;
    Money yourAmount, myAmount(10,9);
    cout << "Enter an amount of money, (use the dollar sign in front): ";
```

```
 cin >> yourAmount;
 cout << "Your amount is: " << yourAmount << endl;
 fullAmount = yourAmount + 25.34;
 cout << "Your amount + 25.34 is: " << fullAmount << endl;
 fullAmount = yourAmount - 70.78;
 cout << "Your amount - 70.78 is: " << fullAmount << endl;
 fullAmount=baseAmount+fullAmount;
 cout << "fullAmount: " << fullAmount << endl;
  return 0;
}
```

Output

When the previous code (i.e Example2) is compiled and executed, it produces the following results:

Account balance: $125.60

Enter an amount of money, (use the dollar sign in front): $567.93

Your amount is: Account balance: $567.93

Your amount + 25.34 is: Account balance: $592.127

Your amount - 70.78 is: Account balance: $497.15

fullAmount: Account balance: $597.75

Notes

- *C++ is able to **input and output** the **built-in data types** using the stream **extraction operator >>** and the **stream insertion operator <<.** The stream insertion and stream extraction operators also can be **overloaded** to perform input and output for **user-defined types** like an object.*
- *Here, it is important to make operator overloading function a friend of the class because it would be called without creating an object.*

- ***istream** and **ostream** serves the base classes for **iostream** class. The class **istream** is used for input and **ostream** for the output.*

Notes

- ***Static Cast:*** *This is the simplest type of cast which can be used. It is a* ***compile time cast****. It does things like implicit conversions between types (such as int to float), and it can also call explicit conversion functions (or implicit ones).*

  static_cast <*new_type***> (***expression***)** →Returns a value of type new_type.

- *The* ***floor****() function in* ***C++*** *returns the largest possible integer value which is less than or equal to the given argument.*
- ***fabs()*** *function is a library function of cmath header, it is used to find the absolute* ***value*** *of the given number, it accepts a number and returns absolute* ***value****.*

ASSIGNMENT

Submission Deadline:23rd March 2020

1- Re-write the Example 1(in this lecture) to overload the operators **++** and **--** using friend functions instead of the member functions.

2- Write a class **Person** with a couple of private members ((String) **Name** and (integer) **Age**) to overload the **stream insertion >>** and **extraction operators <<** using a friend function.