Lesson 9: Function and operator overloading

9.1. Introduction

In our previous lesson, you learnt about functions: function declaration, definitions and function calls. You also learnt about inline functions and recursive functions. Today, you will learn about overloading of functions and operators.

9.2. Lesson objectives

By the end of this lesson, you will be able:

- Describe function overloading
- Discuss the limitations and benefits of function overloading
- Discuss operator overloading

9.3. Lesson outline.

This lesson is organized as follows:

9.4. Function overloading

Function overloading is a feature of C++ used to create multiple functions with the same name, so long as they have different formal parameters. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list.

You cannot overload function declarations that differ only by return type. **Example:** A program to use overloading to calculate area.

```
#include<iostream> using
 namespace std;
float area(float l, float w ) //the formal parameters are l and w
 {      return
(l*w);
 }

 float area(float r ) //the formal parameter is r
{
            return
(3.142*r*r); }

 int main()
{ float
 length,width,radius;
cout<<"Enter the length"<<endl; cin>>length;
cout<<"Enter the width"<<endl;
 cin>>width;
cout<<"Enter radius"<<endl;
 cin>>radius;
cout<<"The area of a circle "<<area(radius)<<endl;
 cout<<"The area of a rectangle "<<area(length,width)<<endl;
return 0;
 }
```

9.5. Operator overloading

This is a programming feature that allows the programmer to specify more than one definition for an operator in the same scope. When an overloaded operator is called, the compiler will have to determine the most appropriate definition to use by comparing the argument types used to call the operator within the parameter types specified in the definitions. Overloaded operators are essentially functions with special names and keyword operator followed by the symbol for the operator being defined. The syntax for operator overloading is:

Return	operator	symbol	(argument

```
{
//definition
}
```

Return type: Like any other function, an overloaded operator must have a return type e.g. int, double etc.
Operator: This is a keyword that precedes the operator to be overloaded.
Symbol: This is the symbol for the overloaded operator e.g. *,/,+,- etc.
Argument list: This specifies the type and name of the parameters.

Example: Consider a binary operator "-" for subtraction. We can overload this operator. See the program below:

```
#include <iostream>
using namespace std;
class overload {
private:
        int num;
public:
    overload() :num(10){ }  void operator -() //definition of
    operator overloading inside the class
            {
                    num = num - 3;
            }
            void Display()  //method to display the result/number
            {
                    cout << "Number: " << num<<endl;
            }
    };
    int main()
    {
            overload a;  //instantiation
        -a;       // call operator function void operator -()
        a.Display(); //invoke the display method
            return 0;
    }
```

Classes and objects will be discussed in the next lesson. However there are some observations to be made:

In the above program, an operator function **void operator-()** is defined (inside class **overload**), which is invoked when - operator operates on the object of type overload. This function will decrease the value of num by 3.

Note:

- Operator overloading cannot be used to change the way operator works on built-in types.
- The operators for assignment (=) and address (&) do not need to be overloaded.
- Operator overloading cannot change the precedence of operators and associativity of operators.
- Not all inbuilt operators in C++ language can be overloaded.

9.6. Function overriding

This is a feature in object oriented programming languages that allows a derived class to provide a specific implementation of a function that is already provided by one of its base class. The implementation in the derived class replaces the implementation in the base class by providing a function/method that has same name, same parameters or signature, and same return type as the method in the base class. If base class and derived class have member functions with same name and arguments. When you create an object of derived

class and write code to access that member function then, the member function in derived class is only invoked, i.e., the member function of derived class overrides the member function of base class. Consider the example below:

```
#include
<iostrea
m> using
namespa
ce std;
class BaseClass
{
protected
:

    in
t age;
public
:
        void getage();
        void displayage();
};
void BaseClass:: getage()
{
        cout << "Enter your age:" << endl;
        cin >> age;
}
void BaseClass::displayage()
{
        cout << "Your age from the base class isc " << age << endl;
}
class DerivedClass:public BaseClass
{
public:
        void getage();
        void displayage();
};
void DerivedClass::getage() //This method will overrride base class method
getage
{
    cout << "Enter your age from derived class:"
<< endl;    cin >> age;
}
void DerivedClass::displayage() //This method //This method will override base
class displayage
{
        cout << "Your age from the derived class is " << age << endl;
}
```

```
int main()
{
        DerivedClass Dc;  //instantiation
    Dc.getage();
    Dc.displayage();
        return 0;
}
```

9.7. Revision questions

[a]. Explain the use of function overriding
[b]. Explain the syntax for operator overloading
[c]. Explain function overloading

9.8. Summary

In this lesson you have learnt about overloading of operators and functions in C++. We have learnt that you can specify more than one definition for an operator or a function in the same scope. This feature we learnt is referred to as overloading. You have also learnt about function overriding in C++. Using function overriding, a member function of derived class can replace the member function of base class.

9.9. Suggestions for further reading

[1]. Sams teach yourself c++ in 24 hours by Jesse Liberty and Rogers Cadenhead.
[2]. Object oriented programming in c++ by Joyce Farrel
[3]. http://www.programiz.com/cpp-programming/operator-overloading