FAST NATIONAL UNIVERSTIY OF COMPUTER AND EMERGING SCIENCES, PESHAWAR

DEPARTMENT OF COMPUTER SCIENCE

CL217 – OBJECT ORIENTED PROGRAMMING LAB

Operator Overloading in C++

LAB MANUAL # 12

Instructor: Fariba Laiq

SEMESTER SPRING 2023

Operator Overloading

In C++, we can change the way operators work for user-defined types like objects and structures. This is known as operator overloading. Since operator overloading allows us to change how operators work, we can redefine how the + operator works and use it to add the data members of our objects.

Syntax for C++ Operator Overloading

To overload an operator, we use a special operator function. We define the function inside the class or structure whose objects/variables we want the overloaded operator to work with.

```
class className {
    ... .. ...
    public
       returnType operator symbol (arguments) {
           ... .. ...
       }
    ... .. ...
};
```

Here,

returnType is the return type of the function.

operator is a keyword.

symbol is the operator we want to overload. Like: +, <, -, ++, etc.

arguments is the arguments passed to the function.

Operator Overloading in Unary Operators

Unary operators operate on only one operand. The increment operator ++ and decrement operator -- are examples of unary operators.

Using a Class Method:

In the following code, the prefix ++ operator is overloaded using the operator++() function, which increments the meter member variable by 1. The postfix ++ operator is overloaded using the operator++(int) function, which also increments the meter member variable by 1.

In the main() function, a Distance object d is created with an initial value of 5, and its value is printed using the display() function. The postfix ++ operator is then applied to d, which increments the meter member variable by 1 using the operator++(int) function and prints the message "Postfix increment" to the console. The updated value of d is printed using the display() function.

Next, the prefix ++ operator is applied to d, which increments the meter member variable by 1 using the operator++() function and prints the message "Prefix increment" to the console. The updated value of d is printed using the display() function. As the methods are defined inside the class so there is no need to pass the value to the operator function, as it will operate on the data members of the class directly.

```
#include <iostream>
using namespace std;
class Distance {
                int meter;
        public:
                Distance(int meter) {
                        this->meter=meter;
                }
                void display() {
                        cout<<"distance: "<<meter<<endl;
                }
                void operator ++()
                {
                        cout<<"Prefix increment"<<endl;
                        ++meter;
                }
                void operator ++(int)
                {
                        cout<<"Postfix increment"<<endl;
                        meter++;
                }
};
int main() {
        Distance d(5);
        d.display();
        d++;
        d.display();
        ++d;
        d.display();
}
```

Output:

```
distance: 5
Postfix increment
distance: 6
```

```
Prefix increment
distance: 7
```

Using a Friend Function:

The below code does exactly the same task as the previous one, the only difference is here the operator function is defined outside the class and declared as friend in the class so that it can directly access the private members of the class. In that case we have to pass one argument to the friend function.

```
#include <iostream>
using namespace std;
class Distance {
                int meter;
        public:
                Distance(int meter) {
                        this->meter=meter;
                }
                void display() {
                        cout<<"distance: "<<meter<<endl;
                }
                friend void operator ++(Distance &d);
                friend void operator ++(Distance &d, int);
};
void operator ++(Distance &d) {
        cout<<"Prefix increment"<<endl;
        ++d.meter;
}

void operator ++(Distance &d, int) {
        cout<<"Postfix increment"<<endl;
        d.meter++;
}
int main() {
        Distance d(5);
        d.display();
        d++;
        d.display();
        ++d;
        d.display();
}
```

Output:

```
distance: 5
Postfix increment
distance: 6
Prefix increment
distance: 7
```

Operator Overloading in Binary Operators

Using a Class Method:

The operator+ function overloads the + operator for Distance objects. The function takes a reference to a Distance object d as input parameter and returns a new Distance object d3, which is the result of adding the meter member variable of the input Distance object and the meter member variable of the calling Distance object. The function also prints the message "Addition" to the console.

In the main() function, two Distance objects d1 and d2 are created with initial values of 5 and 3, respectively, and their values are printed using the display() function. The + operator is then applied to d1 and d2, which calls the operator+ function defined in the Distance class and returns a new Distance object d3, whose value is the sum of the meter member variables of d1 and d2. The value of d3 is printed using the display() function.

```
#include <iostream>
using namespace std;
class Distance {
                int meter;
        public:
                Distance(int meter) {
                        this->meter=meter;
                }
                void display() {
                        cout<<"distance: "<<meter<<endl;
                }
                Distance operator + (Distance &d) {
                        cout<<"Addition"<<endl;
                        int meter=this->meter+d.meter;
                        Distance d3(meter);
                        return d3;
                }
};
int main() {
        Distance d1(5);
        Distance d2(3);
        d1.display();
        d2.display();
        Distance d3=d1+d2;
        d3.display();

}
```

Output:

```
distance: 5
distance: 3
Addition
distance: 8
```

Using a Friend Function:

This code demonstrates the same logic as the previous ones, the only difference is that the operator function is defined outside the class and declared as a friend in the Distance class so that it can access the private data members of the class directly. As this function is defined outside the class so in that case of binary operator overloading, we have to pass both arguments to the function.

```
#include <iostream>
using namespace std;
class Distance {
            int meter;
        public:
            Distance(int meter) {
                this->meter=meter;
            }
            void display() {
                cout<<"distance: "<<meter<<endl;
            }
            friend Distance operator + (Distance&, Distance&);
};
Distance operator + (Distance &d1, Distance &d2) {
        cout<<"Addition"<<endl;
        int meter=d1.meter+d2.meter;
        Distance d(meter);
        return d;
}

int main() {
        Distance d1(5);
        Distance d2(3);
        d1.display();
        d2.display();
        Distance d3=d1+d2;
        d3.display();

}
```

Output:

```
distance: 5
distance: 3
Addition
distance: 8
```