Operator Overloading

Operators Cannot be Overloaded

- Ternary operator (?)
- Scope resolution operator (::)
- Size of operator
- Pointer to member operator (.*)

Unary Arithmetic Operators

Ex: Negative Operator

Continued on previous myString class

```
//Unary (Negative) operator overloading
    myString operator-(){
        //allocate memory for the new string
        char *buffer = new char[strlen(str)+1];
        //copy the string to the buffer string
        strcpy(buffer,str);
        //convert the string to lowercase
        for(int i=0;i<strlen(buffer);i++){
            buffer[i] = tolower(buffer[i]);
```

```
        }
        //return the new string
        return myString(buffer);
    }
```

main

```
int main(void)
{
    myString str1{"SAFIA"};
    myString str2;
    str2 = -str1; //Convert the string to lowercase
    str1.display();
    str2.display();
    return 0;
}
```

Output:

```
safia@safia:~/CPP/Learning-CPP/SessionLabs/Session4$ ./Lab1
SAFIA
safia
```

Binary Operator

Ex: Concatenation

`str1 + str2`

```
myString operator+(const myString &source){
        //allocate memory for the new string
        char *buffer = new char[strlen(str)+strlen(source.str)+1];
        //copy the string to the buffer string
        strcpy(buffer,str);
        //concatenate the source string to the buffer string
        strcat(buffer,source.str);
        myString temp = myString(buffer);
        delete [] buffer;
        //return the new string
        return temp;
    }
```

main

```
int main(void)
{
    myString str1{"SAFIA"};
    myString str2;
    str2 = -str1; //Convert the string to lowercase
    str1.display();
    str2.display();
    myString str3;
    str1 = " Hany";
    str3 = str2 + str1; //Concatenate the strings
    str3.display();
    return 0;
}
```

Output:

```
safia@safia:~/CPP/Learning-CPP/SessionLabs/Session4$ ./Lab1
SAFIA
safia
safia Hany
```

Compersion Overloading

Ex: Equal equal == operator

```
//Compersion operator overloading == operator
    bool operator==(const myString &source){
        return (std::strcmp(str,source.str) == 0);
    }
```

main

```
int main(void)
{
    myString str1{"SAFIA"};
    myString str2;
    str2 = -str1; //Convert the string to lowercase
    str1.display();
    str2.display();
    myString str3;
    str1 = " Hany";
    str3 = str2 + str1; //Concatenate the strings
    str3.display();
    if(str1 == str2){
        std::cout << "Strings are equal" << std::endl;
```

```
    }
    else{
        std::cout << "Strings are not equal" << std::endl;
    }
    return 0;
```

Output:

```
safia@safia:~/CPP/Learning-CPP/SessionLabs/Session4$ ./Lab1
SAFIA
safia
safia Hany
Strings are not equal
```

If implementing outside the class

```
// Implementing outside the class
//Binary (+) operator overloading for concatenation
//Globpal function not related to any class
//We have to make it friend to the class to access the private member str
myString operator+(const char *lhs, const myString &rhs){
    //allocate memory for the new string
    char *buffer = new char[strlen(lhs)+strlen(rhs.str)+1];
    //copy the string to the buffer string
    std::strcpy(buffer,lhs);
    //concatenate the source string to the buffer string
    std::strcat(buffer,rhs.str);
    myString temp = myString(buffer);
    delete [] buffer;
    //return the new string
    return temp;
}
```

Full Example

Lab1.hpp file

```
#include <iostream>

using namespace std;

// Define the class Integer
class Integer
{
private:
    int *m_pInt; // Pointer to an integer in the heap
public:
    // Default constructor
    Integer();
    // Parameterized constructor
    Integer(int value);
    // Copy constructor
    Integer(const Integer &obj);
    // Move constructor
    Integer(Integer &&obj);
    // Destructor
    ~Integer();
    // Set the value of the integer
    void Set_Value(int value);
    // Get the value of the integer
    int Get_Value() const;
    // Copy assignment operator
    Integer &operator=(const Integer &obj);
    // Move assignment operator
    Integer &operator=(Integer &&obj);
    // Addition operator
```

```
    Integer operator+(const Integer &obj);
    // Subtraction operator
    Integer operator-(const Integer &obj);
    // Multiplication operator
    Integer operator*(const Integer &obj);
    // Pre-increment operator
    Integer &operator++();
    // Post-increment operator
    Integer operator++(int);
    // Pre-decrement operator
    Integer &operator--();
    // Post-decrement operator
    Integer operator--(int);
    // Greater than operator
    bool operator>(const Integer &obj);
    // Not equal to operator
    bool operator!=(const Integer &obj);
    // Insertion operator
    friend ostream &operator<<(ostream &out, const Integer &obj);
    // Extraction operator
    friend istream &operator>>(istream &in, Integer &obj);
};
```

Lab1.cpp file

```
#include "Lab1.hpp"
#include <iostream>
//Default Constructor
Integer::Integer()
{
    std::cout << "Default constructor: Integer() " << std::endl;
    m_pInt = new int(0);
}
```

```cpp
//Parameterized Constructor
Integer::Integer(int value)
{
    std::cout << "Parameterized constructor: Integer(int value) " << std::endl;
    m_pInt = new int(value);
}
//Copy Constructor
Integer::Integer(const Integer &obj)
{
    std::cout << "Copy constructor: Integer(const Integer &obj) " << std::endl;
    m_pInt = new int(*obj.m_pInt);
}
//Move Constructor
Integer::Integer(Integer &&obj)
{
    std::cout << "Move constructor: Integer(Integer &&obj) " << std::endl;
    m_pInt = obj.m_pInt;
    obj.m_pInt = nullptr;
}
//Destructor
Integer::~Integer()
{
    std::cout << "Destructor: ~Integer() " << std::endl;
    delete m_pInt;
}
//Copy Assignment Operator
Integer &Integer::operator=(const Integer &obj)
{
    if (this != &obj)
    {
        delete m_pInt;
        m_pInt = new int(*obj.m_pInt);
        std::cout << "Copy assignment operator: Integer &operator=(const Integer &obj) " << std::endl;
    }
```

```
    return *this;
}
//Move Assignment Operator
Integer &Integer::operator=(Integer &&obj)
{

    if (this != &obj)
    {
        delete m_pInt;
        m_pInt = obj.m_pInt;
        obj.m_pInt = nullptr;
        std::cout << "Move assignment operator: Integer &operator=(Integer &&obj) " << std::endl;
    }
    return *this;
}
//Overloading + operator
Integer Integer::operator+(const Integer &obj)
{
    std::cout << "Overloading + operator: Integer operator+(const Integer &obj) " << std::endl;
    Integer temp;
    *temp.m_pInt = *m_pInt + *obj.m_pInt;
    return temp;
}
//Overloading - operator
Integer Integer::operator-(const Integer &obj)
{
    std::cout << "Overloading - operator: Integer operator-(const Integer &obj) " << std::endl;
    Integer temp;
    *temp.m_pInt = *m_pInt - *obj.m_pInt;
    return temp;
}
//Overloading * operator
Integer Integer::operator*(const Integer &obj)
{
```

```
    std::cout << "Overloading * operator: Integer operator*(const Integer &obj) " << std::endl;
    Integer temp;
    *temp.m_pInt = *m_pInt * *obj.m_pInt;
    return temp;
}
//Overloading ++ operator for prefix
Integer &Integer::operator++()
{
    std::cout << "Overloading ++ operator for prefix: Integer &operator++() " << std::endl;
    ++(*m_pInt);
    return *this;
}
//Overloading ++ operator for postfix
// ++(int) is a dummy parameter to differentiate between prefix and postfix
Integer Integer::operator++(int)
{
    std::cout << "Overloading ++ operator for postfix: Integer operator++(int) " << std::endl;
    Integer temp(*this);
    ++(*m_pInt);
    return temp;
}
//Overloading -- operator for prefix
Integer &Integer::operator--()
{
    std::cout << "Overloading -- operator for prefix: Integer &operator--() " << std::endl;
    --(*m_pInt);
    return *this;
}
//Overloading -- operator for postfix
// --(int) is a dummy parameter to differentiate between prefix and postfix
Integer Integer::operator--(int)
{
    std::cout << "Overloading -- operator for postfix: Integer operator--(int) " << std::endl;
    Integer temp(*this);
```

```
    --(*m_pInt);
    return temp;
}
//Overloading > operator
bool Integer::operator>(const Integer &obj)
{
    std::cout << "Overloading > operator: bool operator>(const Integer &obj) " << std::endl;
    return *m_pInt > *obj.m_pInt;
}
//Overloading != operator
bool Integer::operator!=(const Integer &obj)
{
    std::cout << "Overloading != operator: bool operator!=(const Integer &obj) " << std::endl;
    return *m_pInt != *obj.m_pInt;
}
//Set Value
void Integer::Set_Value(int value)
{
    std::cout << "Set_Value() " << std::endl;
    *m_pInt = value;
}
int Integer::Get_Value() const
{
    std::cout << "Get_Value() " << std::endl;
    return *m_pInt;
}

int main()
{
    Integer i1(10);
    Integer i2(20);
    Integer i3;
    cout << "The value of i1 is:" << endl;
    cout << i1.Get_Value() << endl;
```

```
    cout << "The value of i2 is:" << endl;
    cout << i2.Get_Value() << endl;
    i3 = i1 + i2;
    cout << "The value of i1+i2 is:" << endl;
    cout << i3.Get_Value() << endl;
    i3 = i1 - i2;
    cout << "The value of i1-i2 is:" << endl;
    cout << i3.Get_Value() << endl;
    i3 = i1 * i2;
    cout << "The value of i1*i2 is:" << endl;
    cout << i3.Get_Value() << endl;
    i3 = ++i1;
    cout << "The value of ++i1 is:" << endl;
    cout << i3.Get_Value() << endl;
    i3 = i1++;
    cout << "The value of i1++ is:" << endl;
    cout << i3.Get_Value() << endl;
    i3 = --i1;
    cout << "The value of --i1 is:" << endl;
    cout << i3.Get_Value() << endl;
    i3 = i1--;
    cout << "The value of i1-- is:" << endl;
    cout << i3.Get_Value() << endl;
    cout <<"The result of i1>i2? is:" << endl;
    cout << (i1 > i2) << endl;
    cout <<"The result of i1!=i2? is:" << endl;
    cout << (i1 != i2) << endl;
    return 0;
}
```

Output:

```
safia@safia:~/CPP/Learning-CPP/ToBeDelivered/Session5/Lab1$ ./Lab1
Parameterized constructor: Integer(int value)
Parameterized constructor: Integer(int value)
Default constructor: Integer()
The value of i1 is:
Get_Value()
10
The value of i2 is:
Get_Value()
20
Overloading + operator: Integer operator+(const Integer &obj)
Default constructor: Integer()
Move assignment operator: Integer &operator=(Integer &&obj)
Destructor: ~Integer()
The value of i1+i2 is:
Get_Value()
30
Overloading - operator: Integer operator-(const Integer &obj)
Default constructor: Integer()
Move assignment operator: Integer &operator=(Integer &&obj)
Destructor: ~Integer()
The value of i1-i2 is:
Get_Value()
-10
Overloading * operator: Integer operator*(const Integer &obj)
Default constructor: Integer()
Move assignment operator: Integer &operator=(Integer &&obj)
Destructor: ~Integer()
The value of i1*i2 is:
Get_Value()
200
Overloading ++ operator for prefix: Integer &operator++()
Copy assignment operator: Integer &operator=(const Integer &obj)
```

```
The value of ++i1 is:
Get_Value()
11
Overloading ++ operator for postfix: Integer operator++(int)
Copy constructor: Integer(const Integer &obj)
Move assignment operator: Integer &operator=(Integer &&obj)
Destructor: ~Integer()
The value of i1++ is:
Get_Value()
11
Overloading -- operator for prefix: Integer &operator--()
Copy assignment operator: Integer &operator=(const Integer &obj)
The value of --i1 is:
Get_Value()
11
Overloading -- operator for postfix: Integer operator--(int)
Copy constructor: Integer(const Integer &obj)
Move assignment operator: Integer &operator=(Integer &&obj)
Destructor: ~Integer()
The value of i1-- is:
Get_Value()
11
The result of i1>i2? is:
Overloading > operator: bool operator>(const Integer &obj)
0
The result of i1!=i2? is:
Overloading != operator: bool operator!=(const Integer &obj)
1
Destructor: ~Integer()
Destructor: ~Integer()
Destructor: ~Integer()
```

Insertion and Extraction Operator

```
Class Integer
{
        ..
        ..
        // Insertion operator
    friend ostream &operator<<(ostream &out, const Integer &obj);
    // Extraction operator
    friend istream &operator>>(istream &in, Integer &obj);
};
}
//Insertion Operator
std::ostream &operator<<(std::ostream &out, const Integer &obj)
{
    out << "Insertion Operator" << *obj.m_pInt << std::endl;
    return out;
}
//Extraction Operator
std::istream &operator>>(std::istream &in, Integer &obj)
{
    std::cout << "Extraction Operator" << std::endl;
    in >> *obj.m_pInt;
    return in;
}
```

main

```
    cout << "Please enter a number to be added as object from class" << endl;
    cin >> i3;
    cout << "The number you have entered after being saved as object" << endl;
    cout << i3;
```

Output

```
Please enter a number to be added as object from class
Extraction Operator
5
The number you have entered after being saved as object
Insertion Operator5
```