```cpp

///////// Lecture 60: Operator Overloading I - Part I (Basics)

// Overload operators in the Integer class
#pragma once
#include <iostream>
class Integer {
    int *m_pInt;
public:
    //Default constructor
    Integer();
    //Parameterized constructor
    Integer(int value);
    //Copy constructor
    Integer(const Integer &obj);
    //Move constructor
    Integer(Integer &&obj);
    // Get and set methods
    int GetValue()const;
    void SetValue(int value);
    // Destructor
    ~Integer();

    // Overload the + operator for connecting 2 variables
    Integer operator +(const Integer & a) const{
        Integer temp;
        *temp.m_pInt = *m_pInt + *a.m_pInt;
        return temp;
    }
};

// Overload the operator + as a global function
Integer operator +(const Integer &a, const Integer &b){
    Integer temp;
    temp.SetValue(a.GetValue() + b.GetValue); //notice that you cannot access the pointers from outside the class.
    return temp;
}
// When you do this BOTH in the member class and the
```

```cpp
global function, the compiler complains that it is an ambiguous overload. Comment out 1 of them when you want to compile it.

//// Main.cpp
int main(){
    Integer a(1), b(3);
    Integer sum = a + b; // note that operator overloading is just a syntactic sugar over function calls, so it seems like we are adding two objects but in reality the compiler will internally invoke the overloaded operator function. You can see in assembly that a function call was made during the addition 'operation'.
    std::cout << sum.GetValue() << std::endl;
    return 0;
}

// Overloading the increment operator of the Integer class
class Integer {
    int *m_pInt;
public:
    //Default constructor
    Integer();
    //Parameterized constructor
    Integer(int value);
    //Copy constructor
    Integer(const Integer &obj);
    //Move constructor
    Integer(Integer &&obj);
    // Get and set methods
    int GetValue()const;
    void SetValue(int value);
    // Destructor
    ~Integer();

    // Overload the ++ operator
    Integer & operator ++(); // Pre-increment operator --> ++var
    Integer & operaotr ++ (int); // Post increment
```

```
      operator -->  var++
    // Overload the comparison operator
    bool operator ==(const Integer &obj) const;

};
//// In Integer.cpp,
Integer & Integer::operator++() // Pre-increment operator
{
    // Increment the pointer
    ++(*m_pInt);
    // return the value at the address of this:
    return *this; // This function returns by reference.
}
Integer Integer::operator++(int){
    // Original value is returned, then it is incremented afterwards.
    Integer temp(*this); // create copy using copy const.
    ++(*m_pInt); // increment the pointer.
    return temp; // this is a temporary - so we cannot return by reference

bool Integer::operator==(const Integer &obj){
    if (a == b) return true;
    else return false;
}
// Pre-increment operators are more efficient compared to the post increment, because post increment requires creation of a temporary object.
```