```

///////// Lecture 63 - Operator Overloading IV (Friend keyword)

// While overloading operators as global functions, we may need to access the private members of the class. Imagine our Integer class does not contain a function called SetValue(). then how do we initialise the object 'a' with the value in x? In this case, we may want to directly access the private members of the Integer class: if we try to do that,

std::istream &operator >> (std::istream &input, Integer &a)
{
    int x;
    input >> x;
    *a.m_pInt = x; // Pointer is not visible!!
    // Compiler gives an error.
}
// This is for good reason, but in some cases we do want private access to a class. That is why C++ provides the keyword 'friend'. Using the keyword 'friend', using the keyword friend, we can make a function a FRIEND of a class, that function will then have access to ALL THE MEMBERS OF THE CLASS - whether they are private, protected or public.

// So we declare the istream operator (remember this is still a global function) as a friend of the class - DEFINE THIS WITHIN THE INTEGER class

// if you omit the friend keyword then the function will become part of the class.
class Integer{
    ...
public:
    ...
    friend std::istream & operator >> (std::istream &input, Integer &a);
};
```

```
    friend class Printer; //
// after doing this, it builds fine.


// You can also make a class which is a FRIEND of another class.
// So if I have some class let's say printer
class Printer{

};

// obviously, usage of friend is discouraged, because it breaks the OOP paradigm. it allows you to access the internal data of the class directly, which can be a source of bugs. that is why friend classes and friend functions should be used only as a last resort to solve your problem.

```