8. Overloading: Advanced Issues

Need for Assignment Operator

```cpp
class String {
    char* str;
public:
    String(char* sptr) {
        str = new char [strlen(sptr) + 1];
        strcpy(str, sptr);
    }
    ~String() { delete[] str; }
};
String s1("Hello"), s2("Hi");
```

![Diagram showing string assignment](image)
Default Assignment: bit-wise Copy
Assignment Vs. Initialization.

s1 = s2; // bit-wise copy

Problems:
• Memory leak
• Improper ownership - resulting in corruption

Overloading operator =

const String& String::operator=(const String& robj)
{
    delete[] str;
    str = new char [strlen(robj.str) + 1];
    strcpy(str, robj.str);
    return *this;
}

s1 = s2; // Calls String::operator=(const String & robj);
Correct Way to Overload =

Problem with above operator= function
s1 = s1; // Disaster.

```cpp
const String& String::operator=(const String& robj)
{
    if (this != &robj)
    {
        delete[] str;
        str = new char [strlen(robj.str) + 1];
        strcpy(str, robj.str);
    }
    return *this;
}
```

The Law of Big-Three

If a class needs a

- Copy Constructor or
- Destructor or
- Assignment operator

It needs all the three!
Overloading []

String mystr = “Hello”;
char c0 = mystr[0], c1 = mystr[1];

char String::operator[](int i)
{
    if (i >= 0 && i < strlen(str))
        return str[i];
    else // throw an Exception...
}

Dual Overloading of []

• What if we have a call to [] on a const object

• C++ allows overloading a method as a const and non-const at the same time!

• You can write two overloaded [] function one that is a const and the other which is not!
Increment & Decrement in `C`  

```
int x = 5;

x = x + 1;  // Add one to x
++x;       // Pre-increment
            Change value of x first
            and then give out value
x++;        // Post-increment
            Change value of x after
giving out the value.
--x;        // Pre-decrement
x--;        // Post-decrement
```

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>y = ++x;</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>y = x++;</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>y = --x;</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>y = x--;</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Overloading `++`

```cpp
class Counter {
    long count; ...
}

const Counter Counter::operator++(){  // Pre-increment
{ count++;
    return *this;
}
const Counter Counter::operator++(int){  // Post-increment
{ Counter temp = *this;
    count++;
    return temp;
}
Overloading `<<` and `>>` for input/output

Complex c1(2, 3), c2(3, 4);
cout << c1 << "", " " << c2; // Should print 2+3i, 3 + 4i
cin >> c1 >> c2;

Must be global functions

```cpp
ostream& operator<<(ostream& ostrmobj, const Complex& cobj) {
    ostrmobj << cobj.rep;
    if (cobj.im > 0)
        ostrmobj << "+";
    if (cobj.imp != 0)
        ostrmobj << cobj.imp << "i";
    return ostrmobj;
}
```

```cpp
istream& operator>>(istream& istrmobj, Complex& cobj) {
    istrmobj >> cobj.rep >> cobj.imp;
    return istrmobj;
}
```

Lab Work: Details provided on-line.