Assignment 1 Feedback
In Assignment 1, we were asked to implement (overload) the == operator and the > operator in the Patient class.

This signifies that it is the responsibility of the Patient class to:
- Determine when two Patient objects are equal, and
- Establish when Patient object A > Patient object B.

It is not the responsibility of the List class:
- This is to say that List class should not refer to any of the Patient class data members.
- In other words, we should not use/see careCard in List.cpp.
Patient Class

// Overloaded Operators
// Description: Comparison operator. Compares "this" Patient object with "rhs" Patient object.
// Returns true if both Patient objects have the same care card number.
bool operator == (const Patient & rhs);

// Description: Greater than operator. Compares "this" Patient object with "rhs" Patient object.
// Returns true if the care card number of "this" Patient object is > the care card
// number of "rhs" Patient object.
bool operator > (const Patient & rhs);

// Description: Prints the content of "this" patient.
// Should have a const
void printPatient();

};

// end of Patient.h
Patient Class

```cpp
// Overloaded Operators
// Description: Comparison operator. Compares "this" Patient object with "rhs" Patient object.
// Returns true if both Patient objects have the same care card number.
bool Patient::operator == (const Patient & rhs) {
    // Compare both Patient objects
    if (this->careCard == rhs.getCareCard())
    {
        return true;
    }
    else
    {
        return false;
    }
} // end of operator ==

// Description: Greater than operator. Compares "this" Patient object with "rhs" Patient object.
// Returns true if the care card number of "this" Patient object is > the care card number of "rhs" Patient object.
bool Patient::operator > (const Patient & rhs) {
    // Compare both Patient objects
    if (this->careCard > rhs.getCareCard())
    {
        return true;
    }
    else
    {
        return false;
    }
} // end of operator >

// Description: Prints the content of "this" patient.
// Should have a const
void Patient::printPatient( ) {
    cout << this->careCard << ", Patient: " << this->name << ", " << this->address << ", " << this->phone << ", " << this->email << endl;
    return;
}
```
List Class

```cpp
// Description: Remove an element.
// Postcondition: toBeRemoved is removed and elementCount has been decremented.
bool List::remove( const Patient& toBeRemoved ) {
    bool ableToRemove = false;
    for (int index = 0; index < elementCount; index++) {
        if (elements[index] == toBeRemoved) {
            for (int shift = index+1; shift < elementCount; shift++) {
                // shift elements <-
                elements[index] = elements[shift];
                index = shift;
            }
            ableToRemove = true;
            elementCount--;
        }
    }
    return ableToRemove;
}
```
List Class - Insert(...)

// If newElement not already in List and there is still some space in the List
if (elementCount < capacity) {
    if (search(newElement) == NULL) { // newElement not already in List
        for (int index = 0; index < elementCount && !done; index++) { // Shifting to make some space for newElement
            if (elements[index] > newElement) {
                cout << elements[index] << " > " << newElement << " at index " << index << endl;
                int shift = elementCount; shift > index; shift--)
                // shift elements right
                cout << elements[shift - 1] << " copied to " << shift << endl;
                elements[shift] = elements[shift - 1];
            }
            done = true;
            where = index;
        }
    }
    // Insert newElement
    cout << "Insert newElement at : " << where << endl;
    elements[where] = newElement;
    ableToInsert = true;
    elementCount++;
}
List Class – Problems!!!

```cpp
// Description: Search for target element.
// Returns a pointer to the element if found, otherwise, returns NULL.
Patient* List::search(const Patient& target) {
    for (int i = 0; i < elementCount; i++) {
        if (target.getCareCard() == elements[i].getCareCard()) {
            return &elements[i];
        }
    }
    return NULL;
}
```

Solution - Remove the call to the getter. We get this new statement:

```cpp
if (target == element[i])
```

However, the above line will not compile!!! 😞

Indeed, we get this error message:

```
List.cpp:117:36: error: passing ‘const Patient’ as ‘this’ argument of ‘bool Patient::operator==(const Patient&)’ discards qualifiers
^ make: *** [List.o] Error 1
```
Why would this statement not compile?

```cpp
if (target == element[i])
```

**Answer:**
Notice, from the method header below, that the `rhs` (right hand side) operand has a type of `const Patient&`, i.e., a reference to a `Patient` object that cannot be modified (i.e., `const`). The left operand, on the other hand is of type `Patient` object (and in the code of the overloaded operator, the left operand is `this`, i.e., a pointer to that `Patient` object). Not the same as a reference.

So, how do we fix this? See next slide!
List Class - Solution

We flip the operands to match the method header:

```cpp
// Description: Search for target element.
// Returns a pointer to the element if found, otherwise, returns NULL.
Patient* List::search(const Patient& target) { // Linear or binary search
    Patient* result = NULL;
    bool found = false;

    for (int index = 0; index < elementCount && !found; index++) {
        if (elements[index] == target) {
            result = &elements[index];
            found = true;
        }
    }

    return result;
} // end of search
```
On the Good Programming Style (GPS) web page of our course web site, it says: “In general, a function/method must only have one (unique) return statement…”

If we compare the above two implementations of the search method, we observe that the one on the left has 2 return statements and the one on the right, only one, located at the end of the method. This is to say that return is not used to exit the loop (as in the first implementation). Instead, a flag found is used to exit the loop. Also, notice the compound condition of the for loop.
WICPS

- Responsible for user interaction
- Suggestion: as it reads the user input, it can also validate this input -> more efficient