Assignments are due on the due date before class and have to include this cover page. Plagiarism in assignment answers will not be tolerated. By submitting their answers to this assignment, the authors named above declare that its content is their original work and that they did not use any sources for its preparation other than the class notes, the textbook, and ones explicitly acknowledged in the answers. Any suspected act of plagiarism will be reported to the Faculty’s Academic Integrity Officer and possibly to the Senate Discipline Committee. The penalty for academic dishonesty may range from failing the course to expulsion from the university, in accordance with Dalhousie University’s regulations regarding academic integrity.
A point I tried to make in class is that many advantages of object orientation emphasized by its proponents are in fact achievable just as well using plain old modules: encapsulation and bundling together of data and functions to manipulate these data. Proper use of object-oriented design patterns, on the other hand, can take your code to a new level of elegance that is much harder if not impossible to achieve without object orientation. The purpose of this assignment is to make you scratch the surface of design patterns in the hope that you will explore this aspect of software engineering in more depth later.

One particular source of conceptual load and indication of inelegant code is the generous sprinkling of $if$ statements throughout the code that do nothing but check for boundary conditions. Proper object-oriented design is very good at eliminating these checks, thereby allowing the programmer to arrive at a much clearer manifestation of the program's logic in the code. At the end of this assignment, your doubly-linked list implementation should be nearly devoid of $if$ and $switch$ statements.

The implementation parts of this assignment can be done in C++, Java, Ruby or Python and should be submitted in electronic form to amink@cs.dal.ca.

**Question 1 (9 marks)** Search the internet or other sources for descriptions of the following design patterns: *Null Object*, *State*, and *Visitor*. Provide a brief description of each pattern.

**Question 2 (10 marks)** Complicated data structures are hard to implement. One of the reasons is that missing parts are often represented using null pointers; any operation on the data structure then needs to test for such null pointers, in order not to dereference them. The *Null Object* pattern can often be used to eliminate these tests, thereby reducing the chance for errors and increasing the elegance of your code.

Implement a doubly-linked list that supports the following operations:

- **Create** (implemented as the list's constructor): Creates a new doubly-linked list.
- **Destroy** (implemented as the list's destructor): Destroys the list $l$, deallocating all its nodes.
- **Head**: Returns $l$'s head node or null if $l$ is empty. (Here, null means null, not a pointer to the null object.)
- **Tail**: Returns $l$'s tail node or null if $l$ is empty. (Once again, null means null, not a pointer to the null object.)
- **InsertBefore**($n, x$): Given a node $n$ of $l$ (you do not need to check whether $n$ is indeed a node of $l$) and an element $x$, create a new node $n'$ before $n$ in the list and store element $x$ in $n'$. The method should return $n'$.
- **InsertAfter**($n, x$): Same as **InsertBefore**, only the new node $n'$ succeeds $n$.
- **Delete**($n$): Delete node $n$ from $l$.
- **Pred**: Return the predecessor of node $n$ in the list or null if $n$ is the head of the list.
- **Succ**: Return the successor of node $n$ in the list or null if $n$ is the tail of the list.
- **Element**: Return the element stored at node $n$. 
For the sake of simplicity (to avoid generics), you may assume the elements to be stored in the list are integers. Proper use of the Null Object pattern allows you to eliminate all checks for null pointers and all checks whether a given node is currently the head or tail of the list.\textsuperscript{1} You can be sure that you have employed the Null Object pattern to its full extent here if your list implementation does not contain a single if or switch statement. This should be your goal here.

**Question 3 (11 marks)**

(a) The use of iterators to iterate over the elements of a collection is in itself a (ubiquitous) object-oriented design pattern, but it is not the only method that can be used to implement iteration. The Visitor pattern is another option. Augment your doubly-linked list implementation with a method \texttt{FOR EACH} that takes a visitor as an argument and applies it to each list node. The visitor should return a Boolean value that indicates whether to abort the iteration or continue to the next element, if there is one. As concrete examples, implement two visitors. The first one should multiply each list element with a value specified as an argument to the visitor’s constructor. The second one should compute the total sum of the elements in the list.

(b) Name a programming language whose default approach to iteration closely resembles the above method based on the Visitor pattern.

**Question 4 (5 marks)** Use the visitor-based approach to iteration from the previous question to accomplish the following task: compute the sum of the first 100 elements of the list (or all elements if there are fewer than 100 elements) and subtract the sum of all remaining elements in the list. In other words, if \(x_1, x_2, \ldots, x_n\) are the elements in the list, compute \(S = \sum_{i=1}^{100} x_i - \sum_{i=101}^{n} x_i\). Your visitor should use the State pattern to keep track of the half of the list it is currently in and increase or decrease \(S\) accordingly.\textsuperscript{2}

\textsuperscript{1}You need to use more than a Null Object in the strict sense. The interpretation of Null Object you should use here is: the object does what should be done if this was a null pointer.

\textsuperscript{2}This use of the State pattern is too simplistic to be convincing, but I wanted to stick with the doubly-linked list as the running example for this assignment.