AC52002 – Question 1: Doubly linked list

This program is designed to store integer numbers using a doubly linked list data structure. Once populated, the program offers operations that can be carried out using this data structure, in line with those specified in the assignment brief.

Design

To begin, the user is met with some constraints on the data that can be entered into the program. These are displayed as two input rules. First, that inputs must be positive integer numbers, and secondly, that input numbers must not exceed 2147483647. These limitations do not seem unreasonable, given the task outlined in the assignment. If these had been valuable features, extra care would have been taken to include these input types. The user is then asked to enter a series of numbers to be entered into the doubly linked list. This is done using a loop which terminates once the user inputs '0'. Once complete, the un-ordered linked list is displayed and then a menu loop is entered. The menu shows an ordered list of the elements and asks the user to select an operation from the following: 1) Enter a new element to the list (in order), 2) Delete a specific value in the list, 3) Print list in ascending order, 4) Print list in descending order. The menu operates as a loop with a switch statement within it. This allows the user to see changes to the list after selecting an operation. Each switch case has a function call, corresponding to the menu options. Once the user is finished with the list, they can enter '0' to exit the program.

The class used to create the linked list is a modified version of the 'Intlist' file that was given to students in lectures. Rather than declare the 'Node' class/struct inside the list class, I have opted to separate them so that interactions are easier. The Node class is now independent from the list class and notably has two Node pointers (members) – Left and Right. This allows for traversal of the list in both directions. Modifications to the list class are as you would intuitively expect, with methods providing the functionality outlined in the project brief. 'HeadInsert' is as it was in the original and inserts at the start of the list. 'OrderInsert', as the name would suggest – enters additional elements to the list in order. This is done by using a temporary node object which iterates through the list, using comparison operators in order to determine the correct position. A 'previous' node object is used to keep track of the previous node. Once the correct position is found, links are re-arranged to complete the list. The 'Delete' method uses both temporary and previous node objects in a similar way. Once the object to be deleted is found, then links are arranged accordingly. The 'PrintAsc' function prints the list elements in ascending order, from head to tail. The 'PrintDesc' function does the opposite, printing from tail to head.

In order to sort the values in the unsorted list into an ascending order, I opted to use a bubble sort algorithm as this seemed to be the most intuitive when thinking about the linked list data structure.

Testing the structural integrity of the doubly linked list

During testing, it was found that the list, when emptied (one element at a time), would produce an unpredictable result if the 'PrintDesc' function was called (garbage value thrown). This was fixed by introducing a check to see if the list was empty before printing.

Similarly, when the list was emptied using the menu, and then populated – it would be fine until a number that was already present was entered, at which point it would become erratic. This was fixed by adding an additional comparison to check if the number was equal to any already present in the list and specifying the insertion operations after that. This can be seen with the large comparison operators in the 'OrderInsert' method.

Lastly, one bug that I could not iron out for a long time was in the case where the user empties the list (one at a time) and then enters a number, and then chooses to print the list in descending order. This was solved after many hours of tinkering with the delete method. Due to these erratic errors which occurred once the list had been emptied – It looks as though there is a lot of redundant code, or code doing similar tasks in the 'Delete' method. This is because slight variations, or comparison cases were needed to solve these issues in different combinations.

Input testing

Testing has primarily taken place at the initial input stage which populates the list.

Input at initial list population	Result
5	Handled successfully
-567	Error handling triggered
test	Error handling triggered
123163514612316516131521	Error handling triggered
When an input with numbers at start, followed by letters (such as '12455test') – The program displays error message. If '0' is entered to complete list, the number will be in the list.	List enters number part into list- As in example given: Current list: 12455 My C++ skills were not good enough to fix this.
Entering a 'space' character when inputting numbers has erratic results (i.e 343 35 232 455)	I could not fix this problem and decided to focus on the structural issues of the linked list (deletion->Insert->print).

Input at operations menu	Result
1	Handled successfully
6	Error handling triggered
13456513213516543213541531	Error handling triggered
ddkjsl	Error handling triggered
Entering spaces when adding a new element	Bizarre erratic behaviours