Assignment
Linked List

Part A: (Doubly Link List)

Develop a C++ Template class doubly linked list to maintain **unordered** data, with a dummy node head with following functions.

1. **Insertat_at_Head:** Add an element at the head of link list.
2. **Delete_from_Head:** Delete an element form the head of list.
3. **Delete_All:** This function will search and remove all entries of an element from list.
4. **Replace_All:** Search and replace all entries of element(s) from the list with provided input values.
5. **Swap:** Swap two nodes of the list. You cannot just swap data of the two nodes; you have to swap the nodes themselves.

Iterator class:

Implement a bidirectional iterator class for this link list, for handling of following operations.

1. **Increment Operator ++:** both post and pre.
2. **Decrement Operator --:** both post and pre.
3. **Dereference Operator:** *
4. **Equality operator:** ==, !=

Implement following functions using iterator object in the linked list class.

1. **Begin:** Points to first element in List.
2. **End:** Points to last element in List usually null.
3. **Insert:** Adds an element at any position in list. This function will take an iterator object, and a value in parameters and will add the elements after the node to which iterator pointer will be pointing.
4. **Insert_Multiple:** Adds multiple elements at any position in list. This function will take an iterator object, and values in parameters and will add the elements after the node to which iterator pointer will be pointing.
5. **Delete:** Delete one element from list at any position. This function will take an iterator object, and a value in parameters and will remove the element, to which iterator pointer will be pointing. (This means actually removing the node from the list)
6. **Delete_Multiple:** Delete multiple consecutive elements from list at any position. This function will take an iterator object, and values in parameters and will remove the elements starting from the node to which iterator pointer will be pointing. (This means actually removing the nodes from the list)
7. **Delete_in_Range:** Delete all nodes that lie in given range. This function will take two iterator objects in parameters and will remove all the elements between first and second iterator pointer.
8. **Search_First:** This function will search and return position of the first occurrence of values in the list. It will return position as an iterator object, or null otherwise.
9. **Search_All:** This function will search and return position of the all occurrences of values from the list. It will return position as iterator objects, or null otherwise.

Part B: (Text Editor)

You have to implement a text editor using the doubly linked list (created in **Part A**). Each character will be stored as a node in the list.

Code Provided: You are provided with four files myconsole.h, myconsole.cpp, editor.h and editor.cpp. Main function is written in editor file so run the program as it is. You can type any character in a given cmd window and use the arrow keys to navigate the screen. There is a maximum window length is set, in which a user can enter characters.

The function mainEditor could be your starting point. Helper functions like checkWhichKeyPressed are given in myconsole source files. Look at the corresponding .h files to see their usage. There is also the point class to store the current coordinates of the screen. You can use different methods of this class wherever required.

What to Implement?

1. Implement backspace functionality, which should work as normal backspace key works. Deletion of a character from the editor and from the linked list also.
2. Insertion of a character in the editor. So, if a user places the cursor in between text, characters should get inserted. Insertion has to be done correspondingly in the linked list also.
3. Up, down, left, right arrow keys to navigate text are already implemented, but you have to add any additional logic that you require there. (For example, making the iterator move in both directions if required)
4. **Search** (function key **F1**) for a string in the entered text. The user can enter a word/string to find at the bottom of the text editor. After inputting the text, the system should search for an instance of that string in the text and the cursor should jump to that location.
5. **Search All** (function key **F2**) for a string in the entered text. The user can enter a word/string to find at the bottom of the text editor. After inputting the text, the system should search for all instances of that string in the text and highlight those all-matched strings as yellow.
6. **Search and replace all** (function key **F3**) for a string in the entered text. The user will enter a word/string to find at the bottom of the text editor, and will also enter second string to replace. After inputting the text, the system should search for all instances of that string in the text and then replace all of them that with second one.
7. **Save** (function key **F4**) should save the text in a file called ‘myeditor.txt’. This function should use the iterator class for printing.
8. Escape key should exit the editor. (Already implemented)

HAPPY PROGRAMMING