PE 1 - Practice Exercises

Singly- and Doubly-linked Lists: Practice Suggestions for the Programming Exam

This is a list of possible functions that might be included in the programming exams. All these functions apply to both **SLL** and **DLL** and should be implemented as **member** functions of the respective classes.

Here are the the **files** you need to **practice** implementing any of the **functions**: **SLL_DLL_files.zip (https://coastdistrict.instructure.com/courses/100494/files/14708541?wrap=1)** (https://coastdistrict.instructure.com/courses/100494/files/14708541/download?download_frd=1)

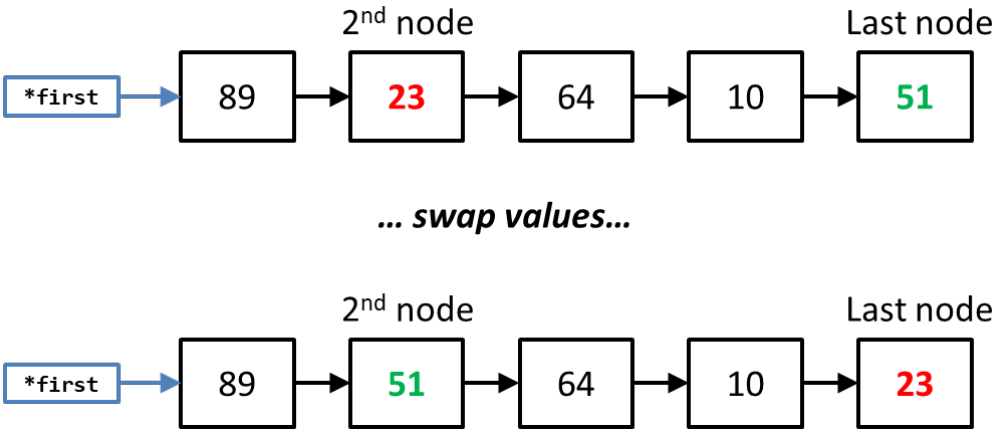
Details to keep in mind

- You will need to determine whether the function is **const** and, in the case you are passing parameters, whether the **parameter**(s) should be passed by **reference** and as **const**.
- Consider all options: list is empty/non-empty, has only one node, has two nodes, has three nodes, has several nodes.
- When changing the position of nodes within the list, it is important that you check that all pointers in a node are set properly. For example, if the first node is deleted, then the second node becomes the first node in the list, in which case the previous pointer should be set to **nullptr**.
- Always assume that there are **NO** duplicates, unless otherwise stated.
- **While taking the exam:**
  - **Assumptions:** You might have one or more assumptions listed--list is empty, list has at least two nodes, etc. Make sure you pay attention to it. Checking if the list is empty when the assumption is that the list has at least one node can cost you a point.
  - **Restrictions:** You might have one or more restrictions--can use only one loop, cannot use any pointers. I suggest you write the code the way it comes to you and then you clean it up according to the restrictions. If, for example, you create a pointer when the restriction is not to create pointers, you can lose one point.
  - Apply **ALL standards** you learned.

Note about manipulating values and nodes

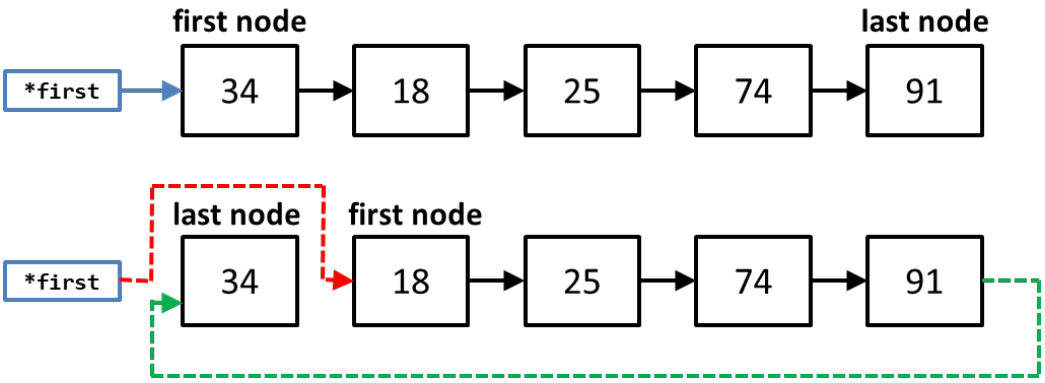
- There is a **difference** in implementation when manipulating values and manipulating nodes.
- **SWAPPING VALUES:** A function that manipulates **values** will simply deal the data stored in the node. For example, if you are asked to swap the value stored in the **second node** with

value stored in the last node, this is what you need to do:



All pointers are still pointing to the same nodes. Only the values stored in the nodes were swapped.

- **SWAPPING NODES:** A function that manipulates nodes will require **rearranging pointers**. You are **NOT** creating a new node, and you are **NOT** moving the node anywhere else in memory. For example, if you are asked to move the first node to the end of the list, this is what you need to do:



Pointer **first** is now pointing to what used to be the second node in the list (the one storing 18), which is now the first node in the list. What used to be the last node in the list (storing 91) is now pointing to what used to be the first node in the list (storing 34), which is now last node.

List of Possible Functions

=> PARAMETER: An int or more storing values.

1. Insert a new node, storing a given value, to the front of the calling object. (Example given in the mock exam for both singly- and doubly- lists.)

2. Insert a new node, storing a given value, at the end of the calling object.
3. Insert a new node, storing a given value, between the first and second node in the calling object.
4. Insert a new node, storing a given value, before the last node of the calling object.
5. Given two int parameters, value1 and value2, insert a new node, storing value1, before the node that stores value2.
6. Given two int parameters, value1 and value2, insert a new node, storing value1, after the node that stores value2.
7. Delete the first node of the calling object.
8. Delete the second node of the calling object.
9. Delete the last node of the calling object.
10. Delete the node before-last of the calling object.
11. Replace the value of the first node of the calling object with the value passed by the parameter.
12. Replace the value of the last node of the calling object with the value passed by the parameter.
13. Replace the value of the second node of the calling object with the value passed by the parameter.
14. Replace the value of the second-to-last node of the calling object with the value passed by the parameter.
15. Given two int parameters, oldValue and newValue, replace the first occurrence of the oldValue with the newValue.
16. Given two int parameters, oldValue and newValue, replace all occurrences of the oldValue with the newValue.
17. Traverse the calling object and search for the value passed by the parameter; return true if found and false if not found. Make sure you stop the loop if the value is found; no need to keep on iterating.

=> NO PARAMETERS

18. Swap the first node with the last node.
19. Swap the value of the first node with the value of the last node.
20. Swap the first node with the second node.
21. Swap the value of the first node with the value of the second node.
22. Swap the first node with the node before last.
23. Swap the value of the first node with the value of the node before last.
24. Swap the second node with the last node.
25. Swap the value of the second node with the value of the last node.
26. Swap the second node with the node before last.

27. Swap the value of the second node with the value of the node before last.
28. Swap the last node with the node before last.
29. Swap the value of the last node with the value of the node before last.
30. Move the first two nodes (or more) to the end of the calling object. This will require resetting pointer first, and last if it is a doubly-linked list, and all other necessary pointers to connect the list.
31. Move the last two nodes (or more) to the front of the calling object. This will require resetting pointer first, and last if it is a doubly-linked list, and all other necessary pointers to connect the list.
32. Search for specific values (all odds, all evens, all multiples of some integer, etc.) and return the number of occurrences.
33. Search for specific values (all odds, all evens, all multiples of some integer, etc.) and add a node at the end (or beginning) of the list. The node will store the number of occurrences.
34. Search for specific values (all odds, all evens, all multiples of some integer, etc.) and return true if at least one occurrence of that specific value is found.
35. Rotate a list to the left (or right) by rotating the nodes. You will need to reset pointers.
36. Rotate a list to the left (or right) by rotating the values stored in the nodes. No pointers will be reset.

=> **PARAMETER:** Another **list**

37. Swap the first node of the calling object with the first node of the parameter object.
38. Swap the value of the first node of the calling object with the value of the first node of the parameter object.
39. Swap the first node of the calling object with the last node of the parameter object.
40. Swap the value of the first node of the calling object with the value of the last node of the parameter object.
41. Swap the first node of the calling object with the second node of the parameter object.
42. Swap the value of the first node of the calling object with the value of the second node of the parameter object.
43. Swap the first node of the calling object with the node before last of the parameter object.
44. Swap the value of the first node of the calling object with the value of the before last of the parameter object.
45. Swap the second node of the calling object with the first node of the parameter object.
46. Swap the value of the second node of the calling object with the value of the first node of the parameter object.
47. Swap the second node of the calling object with the last node of the parameter object.
48. Swap the value of the second node of the calling object with the value of the last node of the parameter object.

49. Swap the second node of the calling object with the second node of the parameter object.
50. Swap the value of the second node of the calling object with the value of the second node of the parameter object.
51. Swap the second node of the calling object with the node before last of the parameter object.
52. Swap the value of the second node of the calling object with the value of the before last of the parameter object.
53. Swap the last node of the calling object with the first node of the parameter object.
54. Swap the value of the last node of the calling object with the value of the first node of the parameter object.
55. Swap the last node of the calling object with the last node of the parameter object.
56. Swap the value of the last node of the calling object with the value of the last node of the parameter object.
57. Swap the last node of the calling object with the second node of the parameter object.
58. Swap the value of the last node of the calling object with the value of the second node of the parameter object.
59. Swap the last node of the calling object with the node before last of the parameter object.
60. Swap the value of the last node of the calling object with the value of the before last of the parameter object.
61. Swap the node before last of the calling object with the first node of the parameter object.
62. Swap the value of the node before last of the calling object with the value of the first node of the parameter object.
63. Swap the node before last of the calling object with the last node of the parameter object.
64. Swap the value of the node before last of the calling object with the value of the last node of the parameter object.
65. Swap the node before last of the calling object with the second node of the parameter object.
66. Swap the value of the node before last of the calling object with the value of the second node of the parameter object.
67. Swap the node before last of the calling object with the node before last of the parameter object.
68. Swap the value of the node before last of the calling object with the value of the before last of the parameter object.
69. Given an empty parameter object, copy one or more elements of the calling object into the parameter object.
70. Given a non-empty parameter object, copy one or more elements of the calling object into the parameter object at a specified position.
71. Given an empty parameter object, copy in reverse some or all elements of the calling object into the parameter object.
72. Given a non-empty parameter object, copy in reverse some or all elements of the calling object

into the parameter object at a specified position.

73. Given an empty calling object, copy one or more elements of the parameter object into the calling object.
74. Given a non-empty calling object, copy one or more elements of the parameter object into the calling object at a specified position.
75. Given an empty calling object, copy in reverse some or all elements of the parameter object into the calling object.
76. Given a non-empty calling object, copy in reverse some or all elements of the parameter object into the calling object at a specified position.
77. Given a non-empty calling object and a non-empty parameter object, append (add to the end) one or more elements of the calling object to the end of the parameter object.
78. Given a non-empty calling object and a non-empty parameter object, append (add to the end) one or more elements of the parameter object to the end of the calling object.
79. Swap calling object and parameter object. Think how to implement this one efficiently without any loops.
80. Compare the elements of the calling object and the parameter object and return true if they are the same (make sure to first check if the number of elements is the same, because there is no need to start a loop if the number of elements differs).

=> PARAMETERS: An array and the number of elements in the array

81. Given an empty calling object, copy one or more elements of the array into the calling object.
82. Given an empty calling object, copy in reverse one or more elements of the array into the calling object.
83. Given an empty calling object, copy specific elements (even, odds, multiples, etc.) of the array into the calling object.
84. Given an empty array, copy specific elements (even, odds, multiples, etc.) of the calling object into the array.
85. Given an empty array, copy in reverse one or more elements of the calling object into the array.
86. Given an empty array, copy specific elements (even, odds, multiples, etc.) of the calling object into the array.
87. Compare the elements of the calling object and the array and return true if they are the same (make sure to first check if the number of elements is the same, because there is no need to start a loop if the number of elements differs).

=> A COMBINATION of any of the above.

You can **expect** the **questions** on the **exam** to be a **combination** of **two or more functions** shown **above**.