CS1020E: Data Structures and Algorithms I

Tutorial 5 – List and Stack

(Week starting 2 March 2015)

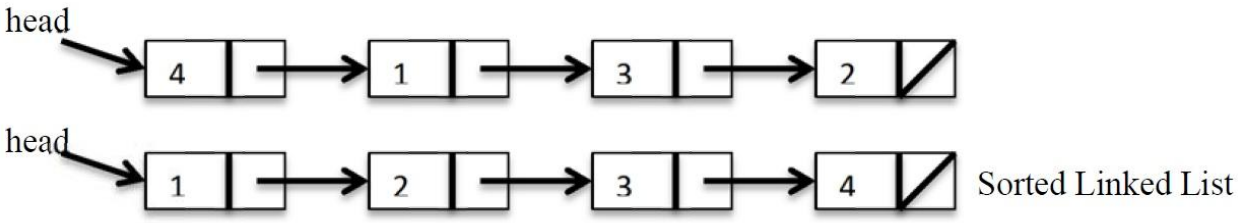
For question Q1, Q2 and Q3, you can use a list implementation given in `ListNode.h` and `List.cpp` files. For Q4, you can use a stack implementation given in `myStack.cpp` file. Implement your answers into `CS1020E_Tut5_Qk.cpp` files, where `k = 1..4`. These sample files serve as a guide for you; you are not required to follow them.

Q1 [Linked List Sorting and Merging]

a. *Linked List Sorting:*

Implement a function called ***sort*** for the List ADT. The *sort()* function rearranges the nodes in the list so that the nodes are in ascending order of their values. One example is illustrated below.

Note: You are not allowed to create any new nodes. You may refer to this link from Wikipedia http://en.wikipedia.org/wiki/Selection_sort for Selection sort algorithm.

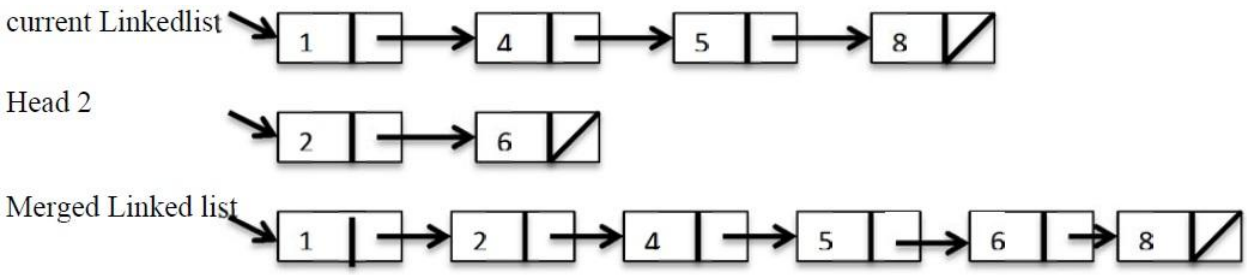


Example of sorting linked list

Sample input: 10, 90, 20, 80, 40, 70, 50, 40
Sample output: 10, 20, 40, 40, 50, 70, 80, 90

b. *Linked List Merging:*

Given two sorted linked lists (nodes in ascending order of their values), write a method that merges these 2 linked lists, with the resulting merged list having nodes that are also in ascending order of their values. You are allowed to traverse each list **only once**. The method should return the Head element of the merged list.



Example of merging 2 linked lists into a sorted linked list

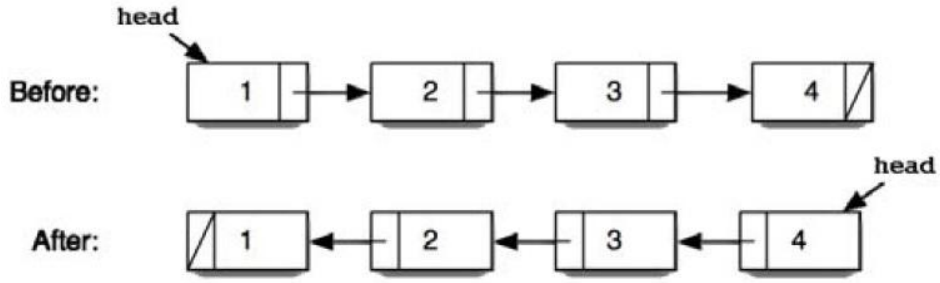
Sample input:
10 40 50 80
20 60
Sample output:
10 20 40 50 60 80

Implement your answer into `CS1020E_Tut5_Q1.cpp` file.

CS1020E: Data Structures and Algorithms I

Q2 [Linked List Reversal]

Given a linked list, write a method to reverse the order of the ListNodes.
Note: you are not allowed to create any new ListNode. One example of the problem is below:



Example of list reversal

Sample input: 10 30 50 20 40
Sample output: 40 20 50 30 10

Implement your answer into `CS1020E_Tut5_Q2.cpp` file.

Q3 [Linked List Manipulating]

a. Given a linked list L and two indexes u and v. Write a method to swap u^{th} and v^{th} nodes.

b. Given a linked list L of distinct integer numbers. Make it a sawtooth list. If there are multiple ways to make sawtooth list, use ***any*** of them. List L is called *sawtooth* list, if its sequence of numbers has sawtooth shape, that is 1^{st} number is less than or equal to 2^{nd} number, 2^{nd} number is greater than or equal to 3^{rd} number, 3^{rd} number is less than or equal to 4^{th} number and so on. An example of sawtooth list is following:
Original L: 10, 40, 50, 20, 60, 30, 90
A sawtooth list from L: 20 40 10 90 30 60 50

Hint: make use of previous methods in this tutorial.
Implement your answer into `CS1020E_Tut5_Q3.cpp` file.

Q4 [Infix to Postfix Transformation using Stack]
In previous tutorials, you created a simple calculator that is able to perform addition, subtraction, multiplication, and division. However, no input format is given for the calculator. In this question, we will simplify the calculator while focusing on the evaluation of input expression itself.

Our mathematical expression is based on *infix* notation, which is the most common expression notation [http://en.wikipedia.org/wiki/Infix_notation]. While easier for human to understand, *infix* notation is harder for computer to parse. An easier notation would be *postfix* notation, also called as *Reverse Polish notation* [http://en.wikipedia.org/wiki/Reverse_Polish_notation].

An example of ***infix*** notation is: `2 + 3 * 5 / 2`. However, it can be ambiguous like in the example, should we perform multiplication before, or after addition, and should they occur before or after the

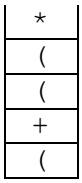
division? In short, the order of evaluation is ambiguous. To remove ambiguity, we have convention on how to evaluate them. However, this is not about ambiguity. We assume that there is no ambiguity, we will put bracket around every operation such that the expression above will be: `( 2 + ( ( 3 * 5 ) / 2 ) )`. You should know how to match the bracket from the lecture notes. Each operator/operands will be separated by a whitespace. However, the operands may consists of multiple digits.

Postfix notation is simpler: the operator is always at the end. As such, the above expression will be written as: `2 3 5 * 2 / +`. Note that there are no brackets as the expression is non-ambiguous.

a) Write a code in C++, using the given Stack class (or your own), to convert *infix* notation to *postfix* notation.

b) Given the following state:

```
Remaining input:          9 ) / 12 ) )
Current postfix output:   6 2 / 4
Stack:
```



What is the original input?

c) What is the result of evaluating the expression given in *question 1b*?

Implement your answer into `CS1020E_Tut5_Q4.cpp` file.