Project 1 (in C++): Linked-list implementation of Stack and Queue. The operations include insertion, deletion and print. (An easy project!)

The input file contains a list of pair {<op, data>} where op is either + or -, where + means insert, - means delete, and data is a character string. For example, + Tom, means insert Tom, - Tom means delete Tom. However, the data will not be used after delete operation.

Your program will construct a stack, then, a queue, all in linked list implementations. Reading from the input pairs, your program will perform operations, given by the pairs, on the Stack. Then, reading the same input pairs, your program will perform operations on a queue.

****

Language: C++ // C++Tutorial will be posted in Blackboard (BB)

****

Project points: 8 pts

Due Date: <u>Soft copy (*.zip) and hard copies (*.pdf)</u>:

+1 (9/8 pts): early submission, 9/1/2022, Thursday before midnight
-0 (8/8 pts): on time, 9/4/2022, Sunday before midnight
-1 (7/8 pts): 1 day late, 9/5/2022, Monday before midnight
-2 (6/8 pts): 2 days late, 9/6/2022, Tuesday before midnight
(-8/8 pts): non-submission, 9/6/2022, Tuesday after midnight

*** Name your soft copy and hard copy files using the naming convention as given in the project submission requirement discussed in a lecture and is posted in Google Classroom.

*** All on-line submission MUST include Soft copy (*.zip) and hard copy (*.pdf) in **<u>the same email attachments</u>** with correct email subject as stated in the email requirement; otherwise, your submission will be rejected.

****

I. Input (use argv[1]): a text file contain a list of pair {<op, data>}, one pair per text line,
where op is either - or +. - means delete, + means insert and data is a character string.

For example,
+ Tom
+ Adam
+ Sean
+ Michael
- Tom
- Tim
:
:

****

II. Outputs: There will be two output files.

****

1) outFile1 (use argv[2]): for stack outputs.
2) outFile2 (use argv[3]): for queue outputs.

****

III. Data structure:

****

- listNode class
  - (string) data
  - (listNode *) next

  Methods:
  - constructor (data) //create a listNode node for data and make sure assign node's next ← null
  - printNode (node) // print in the format as below:
    (node's data, node's next's data) →
- LLStack class
  - (listNode*) top

Methods:
- constructor (...) // create a new LLStack with a dummy node; set the data in the dummy node to "dummy" and next to null and let top points to the dummy node.
- push (newNode) // insert newNode after dummy node
- (bool) isEmpty () // if top's next is null returns true, otherwise returns false.
- listNode* pop() // if stack is not empty, deletes and returns the top of the stack, i.e., the node after dummy.
  // otherwise, print "stack is empty" to outfile 1.
- buildStack (inFile) // build a stack from the data in inFile. See algorithm steps below.
- printStack(…) // call printNode() for all nodes (include the dummy node) in the stack, i.e.,
  // Top → (dummy, John) →(John, Adam) → …→(Sue, null)→null

- LLQueue class
  - (listNode *) head // head always points to the dummy node!
  - (listNode *) tail // tail always points to the last node of the queue.

Methods:
- constructor(...) // create a new LLQueue with a dummy node, set the data in the dummy node to "dummy" //and next to null; and let both head and tail point to the dummy node.
- insertQ (Q, newNode) // insert the newNode after the tail of Q, i.e., after the node points by tail, i.e.,
  // newNode's next ←Q.tail's next
  // Q.tail's next ← newNode
  // Q.tail ← newNode
- (listNode *) deleteQ (…)
  // if Q is not empty, delete and return the node after the dummy (i.e., Q.head's next).
- (bool) isEmpty (…)// Returns true if tail points to the dummy node, returns false otherwise.
- buildQueue (…) // build a queue from nodes in the stack. See algorithm steps below.
- printQueue(…) // call printNode() for all nodes (include the dummy node) in the Queue, i.e.,
  // Head → (dummy, John) →(John, Adam) → …→(Sue, null)→null
  // Tail → (Sue, null)

****
IV. main (…)
****

```
Step 0:  inFile ← open from argv[1]
         outFile1, outFile2 ← open from argv[2], argv[3].
Step 1:  S ← creates a LLStack using constructor.
Step 2: buildStack(S, inFile, outFile1)
Step 3: close inFile
Step 4: re-open inFile
Step 5:  Q ← creates a LLQueue using constructor.
Step 6: buildQueue (inFile, outFile2) // Algorithm steps is given below
Step 7: close inFile
```

****
V. buildStack (S, inFile, outFile1)
****

```
Step 1:  op, data ← read the pair from inFile
         outFile1 ← output op, data // write description

Step 2: if op == "+"
            newNode ← get a listNode with data // Use listNode constructor
            S.Push (newNode) // put newNode onto the top of the stack (i.e., after dummy node)

        Else if op == "-"
            junk ←S.Pop ()
            if junk is not null
                free junk
            else outputs to outFile1 a message:  Stack is empty.
```

```
Step 3:  printStack(outFile1)

Step 4: repeat step 1 to step 3 until inFile is empty.
```

****
VI. buildQueue (Q, inFile, outFile2)
****

```
Step 1: op, data ← read the pair from inFile
        outFile2 ← output op, data // write description

Step 2: if op == "+"
                newNode ← get a listNode with data // Use listNode constructor
                Q.insertQ (newNode) // insert the newNode after the tail of Q;
        else  if op == "-"
                        Junk ← Q.deleteQ ()
                        If junk not null
                                Free junk
                        else outputs to outFile2 a message: the Queue is empty.
Step 3: printQ(outFile2)

Step 4: repeat step 1 to step 3 until inFile is empty.
```