Project 1 (in C++): Linked-list implementation of Stack, Queue, and ordered list (in ascending order). The operations include insertion, deletion and print.

The input file contains a list of pair {<op, data>} where op is either + or -, where + means insert, - means delete, and data is a positive integer. For example, + 20, means insert 20, -20 means delete 20.

Your program will construct a stack, then, a queue and then, an ordered list, 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, then do the same on an ordered list.
** For Stack and Queue, the data after the '-' is not used.

****
Language: C++ // C++Tutorial and C++ project Tutorial are posted in Google Classroom
****
Project points: 10 pts
Due Date: Soft copy (*.zip) and hard copies (*.pdf):

-0 2/10/2021 Wednesday before midnight
-1 for 1 day late: 2/11/2021 Thursday before midnight
-2 for 2 days late: 2/12/2021 Friday before midnight
-10/10: 2/12/2021 Friday 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 **the same email attachments** 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 positive integer.
For example,
+ 12
+ 6
+ 8
+ 25
- 8
- 10
:
:
****
II. Outputs: There will be three output files.
****
1) outFile1 (use argv[2]): for stack outputs.
2) outFile2 (use argv[3]): for queue outputs.
3) outFile3 (use argv[4]): for ordered List output.

****
III. Data structure:
****
- listNode class
  friend of LLStack, LLQueue, LLlist
  - data (int)
  - next (listNode)

  Method:
  - constructor (data) //create a node with given data
  - printNode (node, outFile)
    in the format as below:
    (node's data, data of node's next) →

- LLStack class
  - listNode * top

  Methods:
  - constructor (...) // creates a new stack with a dummy node, set the data in the dummy node to -9999 and set next to null; and let top points to the dummy node
  - buildStack (inFile) // see algorithm below. Implement all other stack operations on your own.

    Use the following stack operations in buildStack. You may add other operations as needed.
  - push (newNode) // puts newNode at the top of Stack, after the dummy node.
  - listNode pop(…) // if stack is empty returns null, otherwise deletes and returns the top node of Stack.
  - bool isEmpty (…) // if Stack is empty returns true, otherwise returns false.
  - printStack (outFile1) // DO NOT delete nodes in Stack! Outputs to outFile1, the entire stack, including dummy node, you may call printNode(…) using the following format:

    Top -->(-9999, next's data)--> (data, next's data)..... --> (data, NULL)--> NULL
    For example:
    Top -->(-9999, 8)-->(8, 11) -->(11, 21) -->(21, 42)..... --> (87, 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; tail points to dummy initially.

  Methods:
  - constructor (...) // creates a new Queue with a dummy node, set the data in the dummy node to -9999 and sets next to null; let both head and tail point to the dummy node.
  - buildQueue (inFile) // see algorithm below. Implement all other Queue operations on your own.

    ** Use the following queue operations in buildQ. You may add other operations as needed.
  - insertQ (newNode) // insert the newNode after the tail of Q.
  - listNode deleteQ (…) // if Q is not empty, deletes the first node, after the dummy from Q and returns the // deleted node, otherwise, returns null.
  - bool isEmpty (…) // checks to see if Q is empty, i.e., tail points to the dummy node. It returns true is Q is empty and false otherwise.
  - printQ (outFile2) // DO NOT delete nodes in Queue! Outputs to outFile2, the entire Q, including the dummy node, you may calling printNode (…) using the following format:

    Front -->(-9999, next's $data_1$)--> (data, next's data)..... --> (data, NULL)--> NULL
    For example:
    Front -->(-9999, 8)-->(8, 11) -->(11, 21) -->(21, 42)..... --> (87, NULL)--> NULL

- LLlist class
  - listNode* listHead // always points to the dummy node

  Methods:
  - constructor (...) // create new list with a dummy node, set the data in the dummy node to -9999. and let listHead points to the dummy node.
  - buildList (inFile) // see algorithm below. Implement all other list operations on your own.

  ** Use the following queue operations in buildList. You may add other operations as needed.

  - listInsert (newNode) // Use the algorithm from the lecture note.

  - listNode deleteOneNode (data) // First, searches a node in the list that contains data; if such node exits, deletes the node from the list and returns the deleted node, otherwise, returns null.

  - printList (outFile3) // Output to outFile3, the entire list, including the dummy node, by calling pirntNode(…) until at the end of the list, using the following format:

    listHead -->(-9999, next's $data_1$)--> (data, next's data)..... --> (data, NULL)--> NULL
    For example:
    listHead -->(-9999, 8)-->(8, 11) -->(11, 21) -->(21, 42)..... --> (87, NULL)--> NULL

```
*******************************
IV. main (…)
*******************************
Step 0:  inFile ← open from argv[1]
         outFile1, outFile2, outFile3 ← open from argv[2], argv[3], argv[4],

Step 1: buildStack (inFile, outFile1) // Algorithm steps is given below

Step 2: close inFile

Step 3: re-open inFile

Step 4: buildQueue (inFile, outFile2) // Algorithm steps is given below

Step 5: close inFile

Step 6: re-open inFile

Step 7: buildList (inFile, outFile3) // Algorithm steps is given below

Step 8: close all files

*******************************
V. buildStack (inFile, outFile1)
*******************************
Step 0: S ← Use LLStack constructor to establish a stack

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
                  Push (newNode) // put newNode onto the top of the stack

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

Step 3:  printStack(outFile1)

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

*******************************
VI. buildQueue(inFile, outFile2)
*******************************
Step 0: Q ←Use LLQueue constructor to establish a queue, with head and tail.

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
                  insertQ (newNode) // insert the newNode after the tail of Q;
         else  if op == "-"
                           Junk ← 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.
```

****
VII. buildList(inFile, outFile3)
****

Step 0: listHead ← Use LList constructor to establish a LList

Step 1: op, data ← read the pair from inFile
outFile3 ← output op, data // write description

Step 2: if op == "+"
newNode ← get a listNode with data // Use listNode constructor
listInsert (newNode)

else if op == "-"
Junk ← deleteOneNode (data)
If junk not null
Free junk
else outputs to outFile3 a message: the data is not in the list

Step 3: printList (outFile3)

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