CS212 – Lab 7 – Implementing a 2-way list

A. Modify your pointer-based queue to use a 2-way pointer-based list. All your original functions should remain in place, except they will now be using 2-way nodes, as discussed in class. Define your anchor with the name "anchor", so we can quickly see it in your source. The functions below are the same as your original functions, but now must be modified to use a 2-way list. The queue acts the same way as your pointer-based queue in the previous lab. Your Node is a struct containing a "next" pointer and a “previous” pointer, as discussed in class, and a data item (a character).

   a. Push – adds one element to the end of the queue (accepts a character as input), returns an int:
      i. -1 if the queue is already full,
      ii. 0 if the item was successfully added
      iii. If the queue is empty (use "isFull" to find out) then create the first element of the queue
   b. Pull – removes the front element (if it exists) and returns the value:
      i. NULL if there were no elements (queue was empty)
      ii. The Front element itself, if there was one
   c. Pulback – removes the rear element (if it exists) and returns the value:
      i. NULL if there were no elements (queue was empty)
      ii. The Front element itself, if there was one
   d. Front – does NOT remove anything. Returns the value:
      i. NULL if there are no elements (queue is empty)
      ii. The front char element if there is one
   e. Back – does NOT remove anything. Returns the value:
      i. NULL if there are no elements (queue is empty)
      ii. The rear char element if there is one
   f. isFull – does NOT remove anything. Returns an int:
      i. 0 if there are no elements (the queue is empty)
      ii. 1 if the queue is full
      iii. -1 for all other cases
   g. Getnext – retrieves the next element in the list (if any)– uses a static pointer to keep track of where it was last pointing. Starts (1st time only) at the element pointed to by the anchor. The direction of movement is defined by a parameter: 0: toward the rear; 1: toward the anchor. When one end of the list is reached the function returns NULL. It is up to the calling application to reverse the direction or reset the pointer to the anchor or end using Reset.
   h. GetCurrent – returns the value of the static pointer used by Getnext. May be used to tell Delete what node to delete.
   i. Reset: takes 1 parameter: 0: reset the Getnext pointer to the start; 1: reset the pointer to the end. Note that, if the list is empty, start=end.
   j. Delete – deletes the element pointed at by the input parameter (a node * pointer). Returns:
      i. 1, if the element was properly deleted
      ii. 0, if there was no node to delete (the input was NULL)

B. Using the above functions, write a program that:

   a. Creates the 1st node and makes the anchor pointer point to it.
   b. Uses your functions above to do the following:
      i. Accept input characters, 1 at a time until the symbol ^ is input
      ii. Push each character onto the queue
      iii. Output the queue (front to end) after all inputs have been received
      iv. Output the queue in reverse order (end to front)

CS212 – Lab 7 – Implementing a 2-way list

Notes:

1. Your main program should have NO access to the queue except through the above functions. The queue & value pointer to the front and end ARE global, so all your functions can get at them, but the main code must never look at or use them.
2. Test the returned value from "malloc". If memory is full (the return is NULL) and more data keeps coming, your program should ignore the data and reply,

   "The queue is full. Enter the '^' character to stop."
3. Repeat the above message if more input (other than "^" keeps coming).