CS212 – Lab 5 – Implementing a stack with pointers

A. Modify your array-based stack to use a pointer-based stack. All your original functions should remain in place, except they will now be using dynamically allocated 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 list. The stack can now grow to use up all available memory (you can have a huge number of elements in the stack). Your Node is a struct containing a "next" pointer as discussed in class, and a data item (a character). You will NOT use STACKSZ to pre-define your stack size, so you will not know how much data we will give you. (It could be thousands of bytes).
   a. Push – adds one element to the top of the stack (accepts a character as input), returns an int:
      i. -1 if the stack is already full,
      ii. 0 if the item was successfully added
      iii. If the stack is empty (use "isFull" to find out) then create the first element of the stack and return a value of 1.
   b. Pop – removes the top element (if it exists) and returns:
      i. NULL if there are no elements (stack is empty)
      ii. The element itself if there was one
   c. Top – does NOT remove anything. Returns a char with the value:
      i. NULL if there are no elements (stack is empty)
      ii. The topmost element if there is one
   d. isFull – does NOT remove anything. Returns an int:
      i. -1 if there are no elements (the stack is empty)
      ii. 1 if the stack is full
      iii. 0 in all other cases

B. Using the above functions, write a program that:
   a. Creates the anchor pointer for the stack of characters
   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 stack
      iii. Output the stack (top to bottom) after all inputs have been received

Notes:

1. Your main program should have NO access to the stack except through the above functions and creation of the anchor. The stack & pointer to the top 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 stack is full. Enter the '^' character to stop."
3. Repeat the above message if more input (other than "^" keeps coming).