CSCI 241 Assignment8 Part 1

1. Execute the command

   ```
   setup 8
   ```

   to create an Assign8 directory in the csci241 directory and copy files from Professor McMahon's account to your account.

2. Go into the ***Assign7*** directory that was created for Assignment 7. ***This is not a typo***. Files from Assignment 7 are needed to complete Assignment 8.

   Copy the two inpost files from Assignment 7 into the directory that is used for Assignment 8:

   ```
   cp inpost.* ../Assign8
   ```

   Copy the two mystack files from Assignment 7 into the directory that is used for Assignment 8:

   ```
   cp mystack.* ../Assign8
   ```

   The mystack class from Assignment 7 must be altered so that the stack is maintained as a linked list rather than a dynamic array.

3. Go into the ***Assign8*** directory.

4. In the ***mystack.h*** file, add a definition for a structure that will be used to represent a node in the linked list. This definition should be placed BEFORE the definition for the mystack class. The Hints section of the assignment write up has a structure definition that you can use (you can change the names if you like).

   The mystack class only requires 2 data members now. A pointer to a node object, which represents the head pointer for the linked list (this is the top of the stack). The 2nd data member is the size_t that holds the size of the stack (this is the number of nodes in the linked list).

   The capacity and reserve methods are not needed for this assignment.

   Change the return type on the top method so that it's a reference to a constant integer rather than a reference to a constant character.

   Change the argument type on the push method so that it's an integer rather than a character.

5. This step is optional. The copy constructor and overloaded assignment operator both have code that creates a copy of the stack. For Assignment 7, the copying was simple because values were copied from one array to another array.

   Copying linked lists requires a little more work/thought because you're working with individual nodes.

   One way to copy a linked list is to write a recursive method. This method requires two arguments: a reference to a node pointer (node *&) that holds the address of where a new node should be placed, and a node pointer (node *) that holds the address of the node to be copied. The method returns nothing.

   If the node pointer argument is pointing at a node:
   - Allocate a new node with a data field that is equal to the data field from the node pointed to by the node pointer argument. The address of the newly allocated node should be assigned to the reference to the node pointer argument.

   - Recursively call the method and pass the address of the next node in the list for the reference to the node pointer argument and address of the next node in the list for the node pointer argument.

   If you decide to use this method, make sure to add the prototype to the class definition for the mystack class.

6. In the ***mystack.cpp*** file, modify the constructors and methods based on the descriptions from the assignment 8 write-up.

   If you're using the method that is described in step 5, add the code for the new method. A calling statement for this method will be used to replace the use of new to allocate an array and the loops in the copy constructor and overloaded assignment operator. The calling statement should pass the head pointer for the mystack object as the 1st argument and the head pointer from the constructor/method argument as the 2nd argument. Don't forget that you still need to copy the size of the stack data member as well.

   Make sure that the clear method is used in the overloaded assignment operator and destructor to replace the delete operation.

7. Compile and execute the code at the command line. Make sure you're using the stack_test.cpp file to test the code.

   ```
   make stack_test
   ```

   This will compile and link the stack_test.cpp source code file. It produces an executable named stack_test.

   ```
   ./stack_test
   ```

```
Default constructor

Testing empty():           SUCCESS
Testing size():            SUCCESS

push(), top(), and pop()

Testing push() and top(): SUCCESS
Testing empty():           SUCCESS
Testing size():            SUCCESS
Testing top():             SUCCESS
Testing push() and top(): SUCCESS
Testing empty():           SUCCESS
Testing size():            SUCCESS
Testing top():             SUCCESS
Testing top() and pop():  SUCCESS

Copy constructor

Testing empty():           SUCCESS
Testing size():            SUCCESS
Testing top():             SUCCESS

Copy assignment operator

Testing empty():           SUCCESS
Testing size():            SUCCESS
Testing top():             SUCCESS
Testing empty():           SUCCESS
Testing size():            SUCCESS
Testing top():             SUCCESS
Testing top() and pop():  SUCCESS
Testing empty():           SUCCESS
Testing size():            SUCCESS
Testing top():             SUCCESS

clear()

Testing empty():           SUCCESS
Testing size():            SUCCESS
```

8. Once the output is correct, take a break!