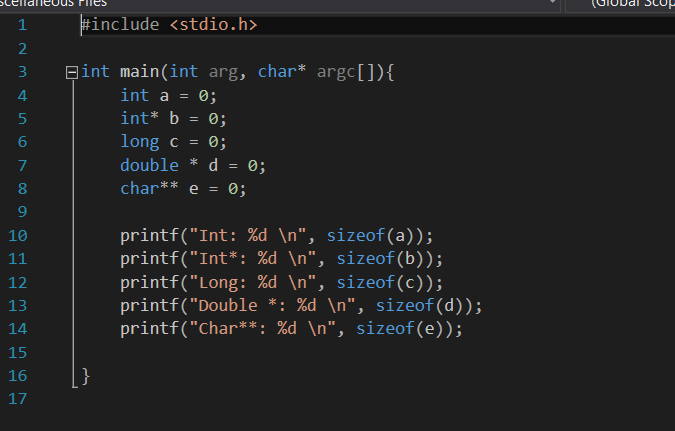
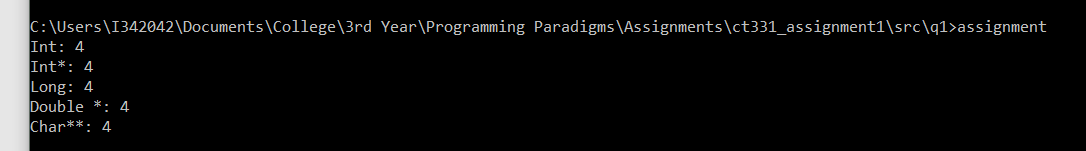
**Cormac Buckley 15534413**

**CT331 - Assignment #1 – C**

**Github Repo:** <https://github.com/CormacBuckley/ct331_assignment1>

**Q1.**

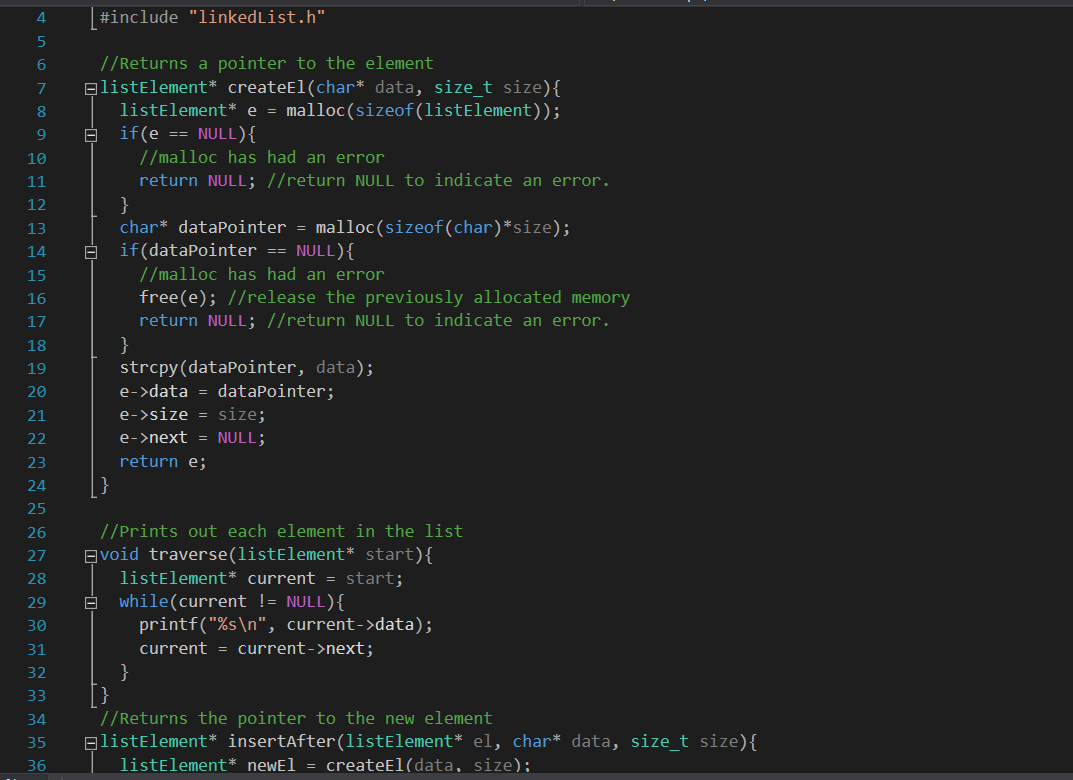


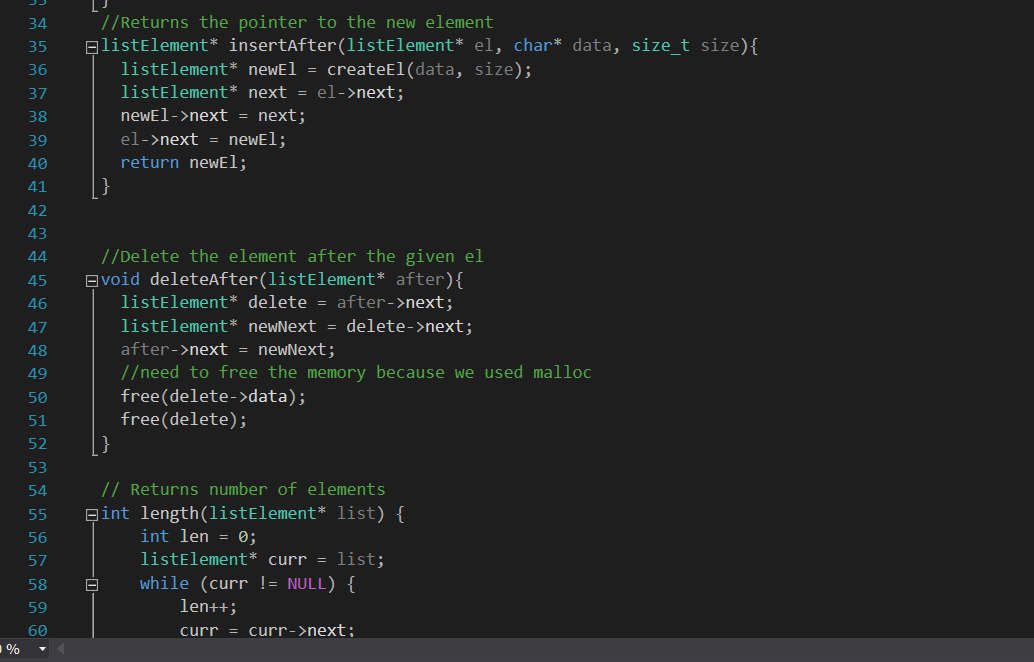


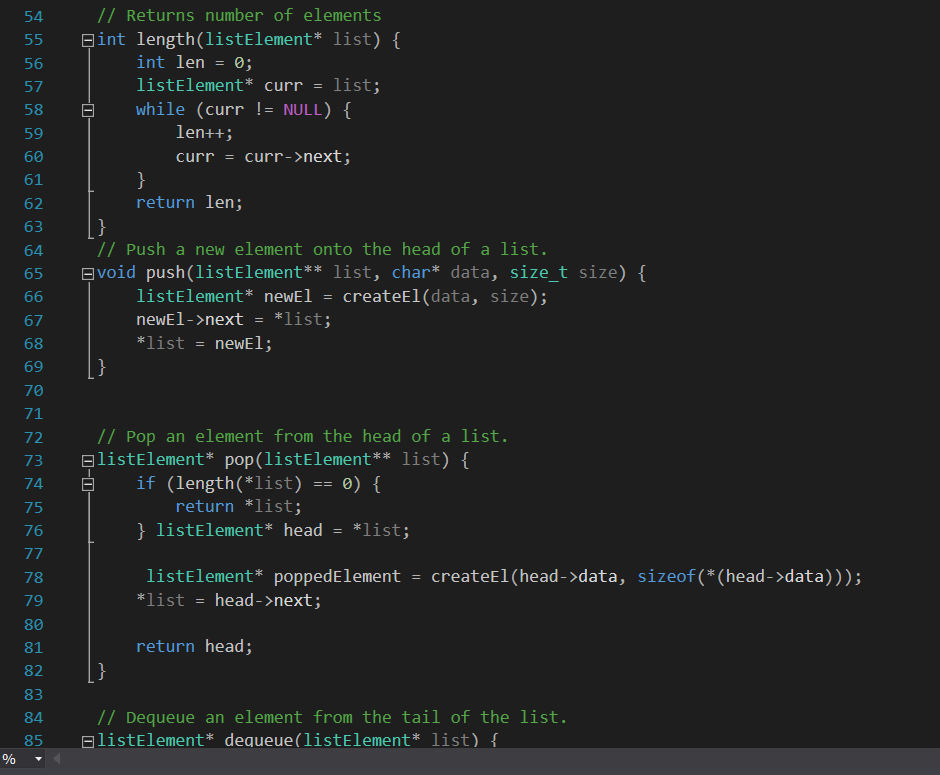
This is my code and results from Q1.

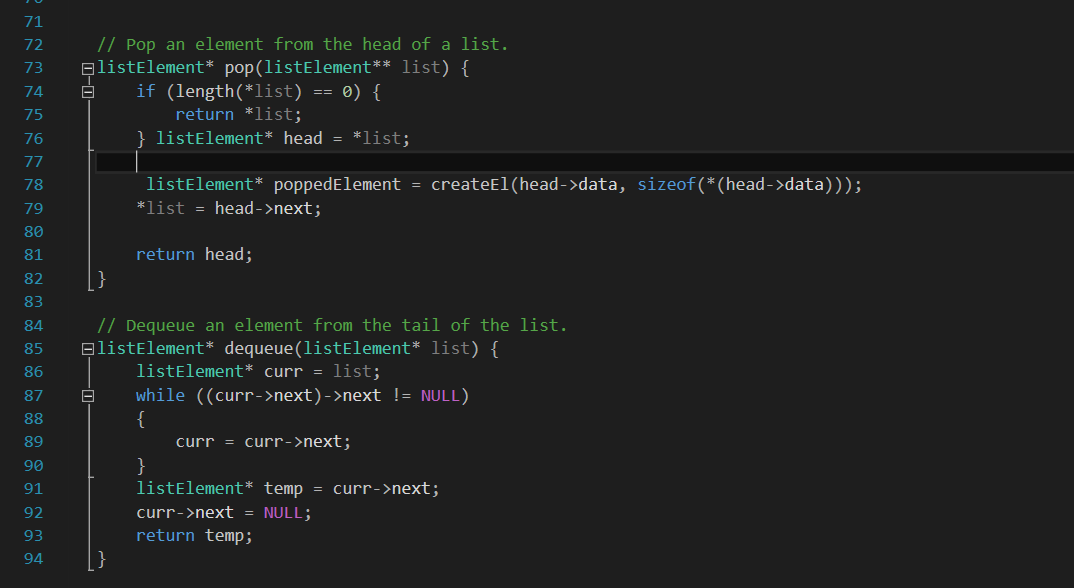
I was initially surprised at my results as each variable type was showing as only using 4 bytes. However, after some research I discovered that data types such as long, double etc. only need to be at least as large as an int. It seems that on certain Windows versions this rule was followed exactly whereas on other systems such as Linux/Mac the sizes are often greater.

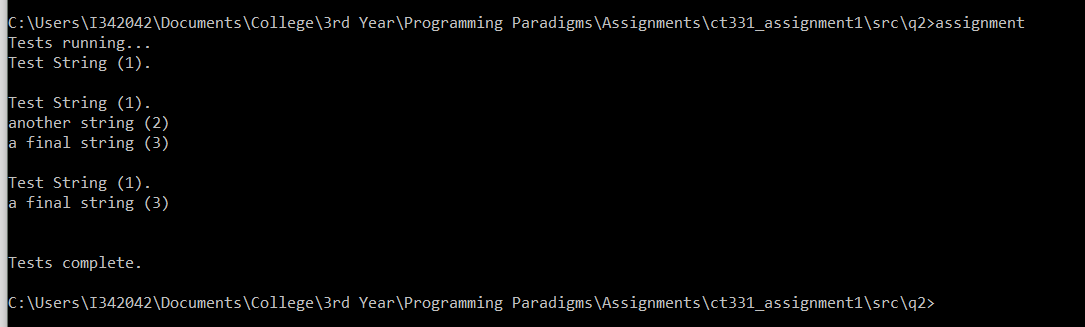
**Q2.**





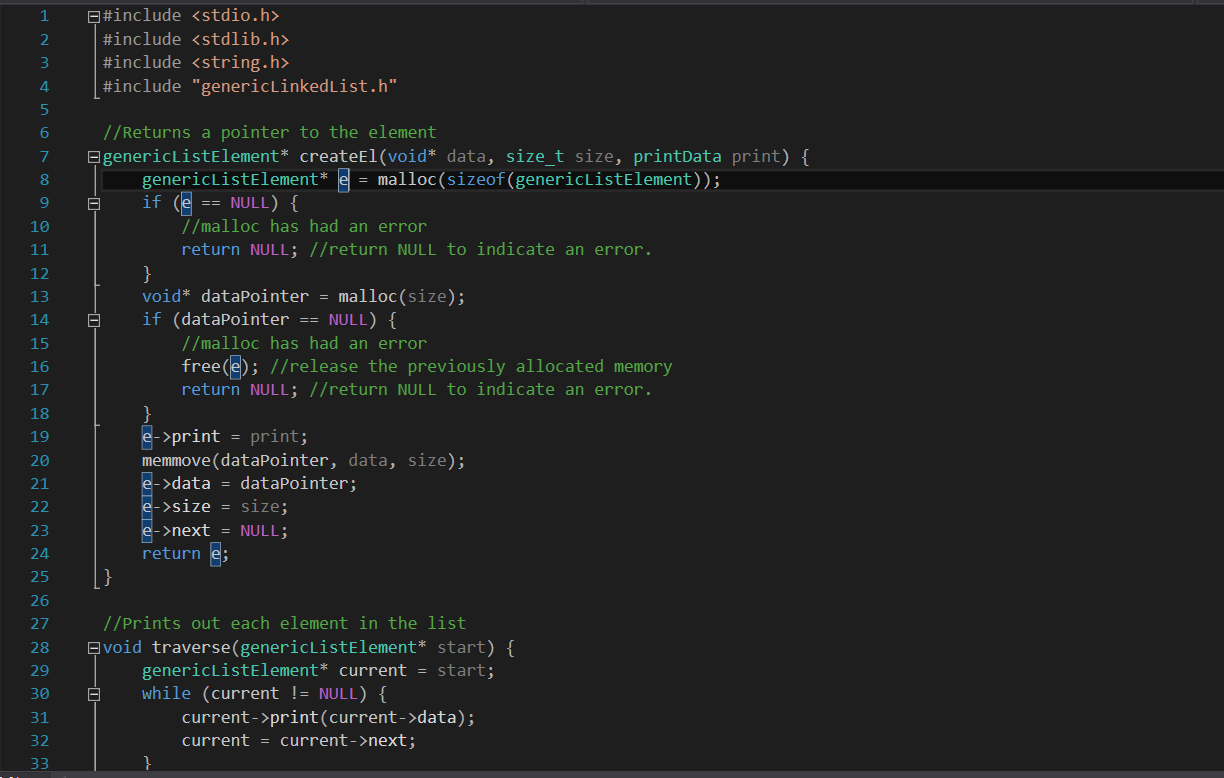


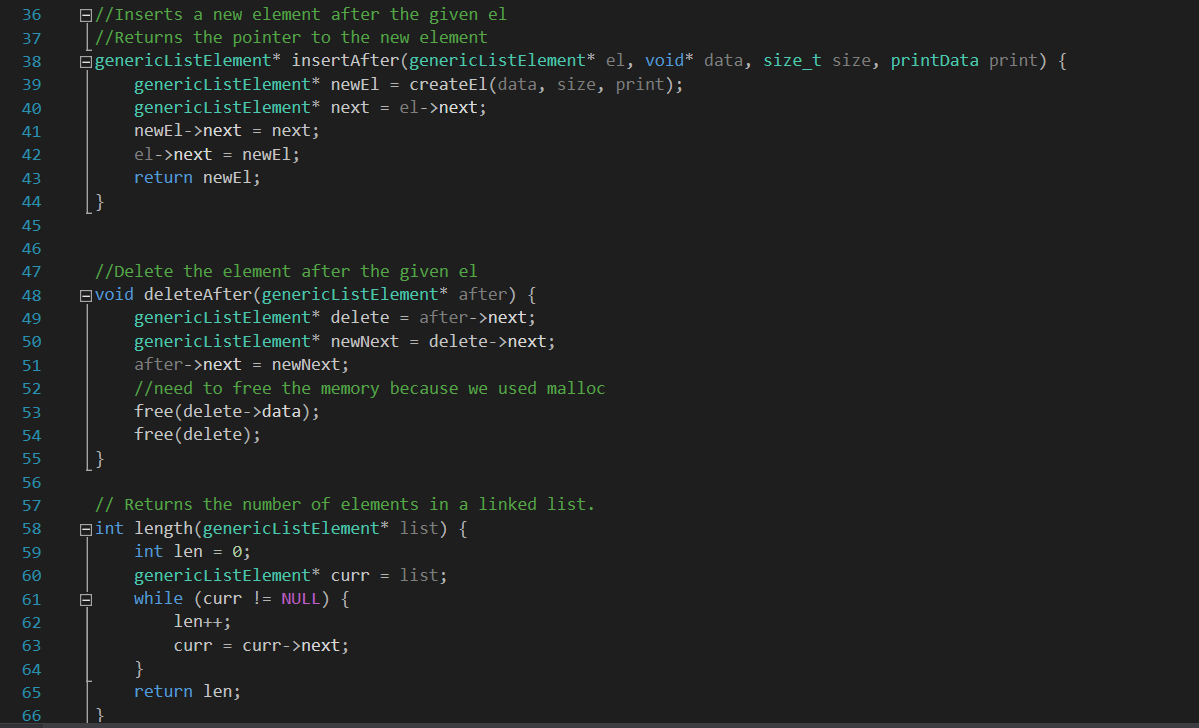


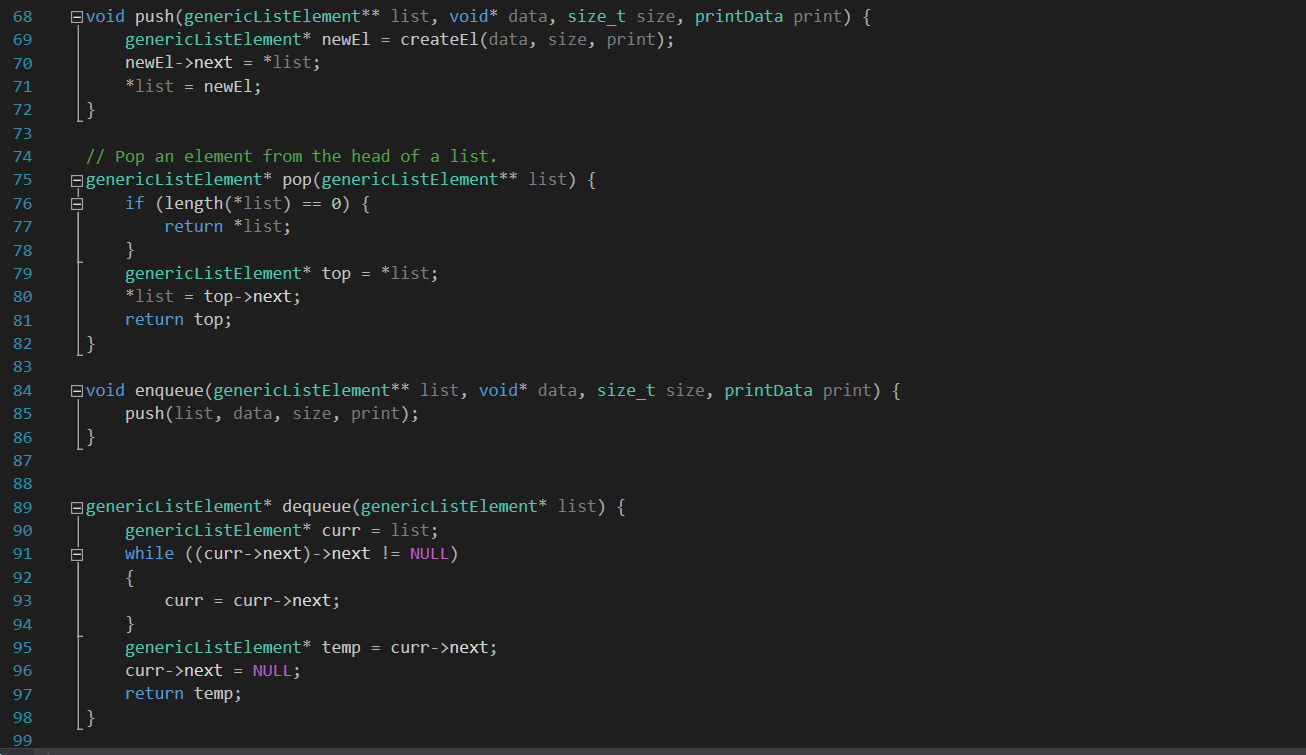


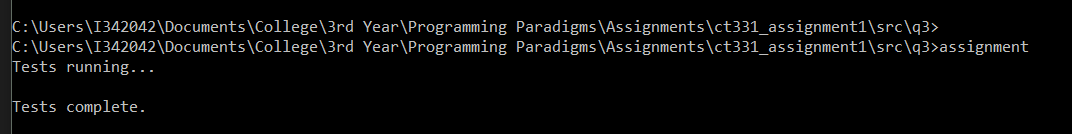
This is my code and results from Q2.

**Q3.**



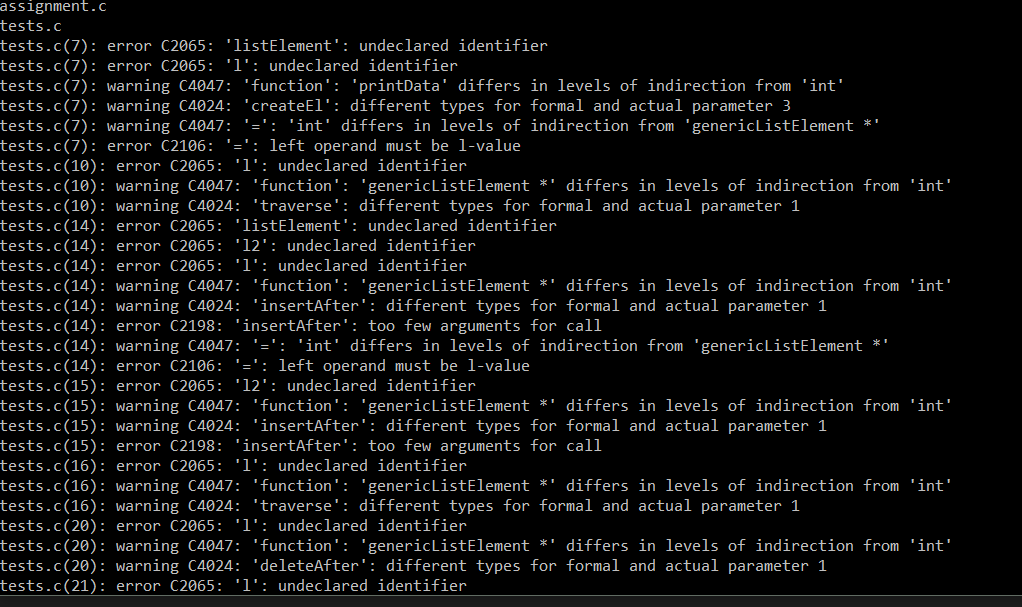






This is my code and results from Q3.

I became very confused when trying to write a test for this question. Below are the errors I got when using the same test code from Q2:



I understand that these will not work in a straight copy paste due to the code and variable changes that I made, however I just became lost in the amount of errors and the terminology used. I was unable to get my head around what I had to change and where it had to be changed.

**Q4.**

Memory usage to traverse a linked list in reverse will be relatively high due to the inability of singly linked lists to go backwards directly. To go backwards you will need to iterate over the list, each time stopping 1 position further back than the last iteration. This gives a bug O complexity of O(n)

An improvement to this method would be to use a double linked list. This data structure uses pointers in both directions and so can reverse traversal direction without any need for iteration. This method gives a complexity of O(1). This provides a substantial improvement as you start to increase the number of elements in your list.