INFSCI 2610 Data Structures
Homework 1
Due Date: By 5:00PM, Thursday, Jan 22

1. Do the following problems from the book **[20]**

   Exercises 2.40 and 2.43

2. **Linked List exercise [40]**: In this exercise you will start with the code given in the book and implement some new functions.

   Start with the following program code from the book:

   list.h : Defines the basic list processing interfaces (Program 3.12). Here you will use int data type for "itemType".

   list.c : Implements the basic list processing interfaces.

   Now, create a client program file client.c (with the main() function). In client.c, implement the following functions:

   (a) Swap an arbitrary pair of nodes (call it swap (?, ?)). That is, your program should ask the user to indicate the elements he wants swapped (i^{th} and j^{th}). Provided they are valid, your program should:
   Print the original values
   Swap the nodes (do not simply swap the values)
   Print the new values.

   (b) Write a function to split the linked list into two lists (call split(?)), such that the first list contains the elements that were at *odd* positions in the original list and the second list contains the elements that were in the *even* positions of the original list. You are not to create any new list!

3. **Infix Calculator [40]**: In this exercise you will implement the Stack ADT in the book and extend the code provided in the book to write an infix calculator that allows the following operations: *, /, +, / and %. Essentially, you have to put together the code in programs 4.1, 4.2, and 4.3.

   a. Program STACK.h: gives you the basic interfaces. For this exercise you can define the Item data type in STACK.h file itself - no need to make a separate header file Item.h as indicated in the text.
   b. Program 4.2 implements + and *. You will add code for the others.
   c. Program 4.3 gives you the Infix-to-postfix conversion. You will add the code for the others.

   The objective of this exercise is to have you compose together the pieces that are already there and write the missing pieces. This should not be difficult, given that other operators are already implemented there.

   (**Extra Credit: 10**)

   Implement the unary operator "–" as well. You can use any technique you want.