CSE 373 Homework 2 Project Description

Assigned:	Wednesday, April 10, 2002
Due:	Wednesday, April 17, 2002
	At the start of class

Introduction

For this homework project, you will develop a code module that implements a simple array based stack and answer some questions. All the source files that we are providing for this assignment are in a zip file on the web site. You should download and unzip the file. The questions were handed out in class; you can also get a copy from the web site.

Stack. The Stack program implements the infix/postfix evaluation capability described in the text. The main program is supplied in ADT/Stack/mainStack.c, and you are to write the individual stack management functions as defined in ADT/include/stack.h. The functions are based very closely on the discussion in the textbook, however, they are not exactly the same.

Grading

The 7 homework assignments of the quarter will count for a total of 50% of your class grade, and each individual homework assignment will count for about 7% of the total class grade. The grading for this project is as follows.

Questions:	10 points
Implementation:	10 points
Total	20 points

NOTE: You need to turn in TWO things: the paper copy of your answer sheet, and the electronic copy of your implementation of stack.c.

Directory Structure

All of the homework projects are implementations of one of the Abstract Data Types that we discuss in class. The directory structure of the files you receive is as follows:

ADT/	top level directory
ADT/include	header files
ADT/symbols	example symbol table files
ADT/Stack	directory for the Stack project
ADT/Lecture	examples from the lectures if any

Program: Stack

This program requires you to implement a simple set of stack management functions. The main program and the header files are supplied; you add code to the implementation file “stack.c” to supply the actual stack management functions.

To do:

1. Create the Stack project just as you did in Homework 1 for the Sum and List projects. For the Program arguments entry, use “..\symbols\infix.txt”.
2. Note that although there is a file “stack.c” provided to you it is very incomplete, and so the project will not link correctly. All the routines that you will implement are missing.
3. Write the routines that are needed by mainStack as described below, rebuild the project, and run it. Debug until done.
4. Review the code as needed in order to answer the questions in the homework.

The function headers for the routines you need to write are in ADT/include/stack.h. Note that the functions are very similar, but NOT IDENTICAL, to the ones defined in the textbook. The functions are as follows.

Stack CreateStack(int count);

Create a new StackRecord data structure, fill it in correctly, and return a pointer to it to the caller.

void DestroyStack(Stack S);

Release all memory allocated by CreateStack. Notice that this procedure does not release memory associated with any items that might be pointed to by entries on the stack. It is the responsibility of the calling program to manage the memory associated with objects other than the stack itself.

ElementType TopOfStack(Stack S);

This function returns the element that was most recently pushed onto the stack. If the stack is empty, the value returned is equal to NULL. The element is NOT popped off the stack. This is a “peek” function that lets you see what you will get if you do a pop.

ElementType Pop(Stack S);

This function the element that was most recently pushed onto the stack and removes it from the stack. If the stack is empty, the value returned is equal to NULL.

void Push(Stack S, ElementType X);

This function pushes the element X onto the stack. If the stack is already full, this method can terminate immediately using a call to the reporter.h macro FatalErrorBound(value, limit) which checks for value>limit and terminates if true. See the questions for a better way of doing this.

int IsEmpty(Stack S);

True (1) if there are no entries in the stack, false (0) if there are some entries on the stack.

int IsFull(Stack S);

True (1) if the stack is full, false (0) if it is not full.

Sample output from the finished program is:

```
infix:    18.39 = 4.99 + 5.99 + 6.99 * 1.06
postfix:  18.39 = 4.99 5.99 + 6.99 1.06 * +

infix:    18.69 = 4.99 * 1.06 + 5.99 + 6.99 * 1.06
postfix:  18.69 = 4.99 1.06 * 5.99 + 6.99 1.06 * +

infix:   288.00 = ( ( ( 2 + 3 ) * 8 ) + 5 + 3 ) * 6
postfix: 288.00 = 2 3 + 8 * 5 + 3 + 6 *

infix:    23.00 = 4.5 + 5.5 + 6.5 * 2
postfix:  23.00 = 4.5 5.5 + 6.5 2 * +

infix:    15.00 = 1 + 2 + ( 3 * 4 )
postfix:  15.00 = 1 2 + 3 4 * +

infix:     0.00 = 3 - 2 - 1
postfix:   0.00 = 3 2 - 1 -

infix:   189.00 = 1 + 2 * 3 + ( 4 * 5 + 6 ) * 7
postfix: 189.00 = 1 2 3 * + 4 5 * 6 + 7 * +

-------------

7 cases, 0 errors.
```