Laboratory 3

Stack ADT

I. OBJECTIVES:

In this laboratory, you will

- understand the concepts of stack and how it can be made into a concrete and valuable tool in problem solving
- implement the Stack ADT based on an array representation of stack
- and develop algorithm that will use the stack operations in different applications

II. DISCUSSION:

A stack is an ordered list collection of items into which new items may be inserted and from which items may be deleted at one end, called the *top* of the stack. We can picture a stack as in Figure 3.1.

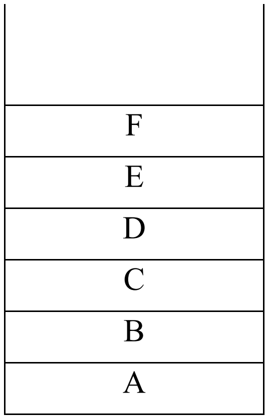

Fig 3.1 Stack containing stack items

Unlike that of array, the definition of stack provides for the insertion and deletion of items, so that a stack is a dynamic, constantly changing object. A Stack is sometimes called a last-in-first-out (LIFO) list.

STACK Operations

Push (s, i)

Given a stack s, and an item i, performing the operation push(s, i) adds item i to the top of stack S.

Pop (s)

Given a stack s, performing the operation pop(s) removes the top element and returns it as a function value. i=pop(s).

Empty (s)

The operation Empty(s) determines whether or not the stack(s) is empty. If the stack is empty, Empty(s) returns the value TRUE; otherwise it returns the value FALSE.

Stacktop (s)

The operation Stacktop (s) returns the top element of stack (s) without removing it.

i = stacktop(s)

Underflow and Overflow

The result of an illegal attempt to pop or access an item from an empty stack is called underflow. **Underflow** can be avoided by ensuring that empty(s) is false before attempting the operation pop(s) or stacktop(s).

Overflow results during array implementation of stack. If the max size is released and attempt of push(s, i) operation will cause an overflow.

III. Test and Debug

1. Create a test program that will implement the following stack operations.

   PUSH(s, i)

   POP(s)

   STACKTOP(s)

   EMPTY(s)

   Notes:

   1. *Consider that the element type (eltype) of the stack as character and maximum length is 4.*
   2. *The library of STACK functions is available in c:\datsala\lab3*

2. Test each stack operation by specifying the corresponding commands:

Commands	Operation/Action
-e	Report whether the stack is empty
-t	Output/Display the top element of the stack
-+ i	Push data item **i** onto top of the stack
--	Pop the top data item
-c	Clear the stack
-q	Exit the test program

3. Compile and do the following sample runs:

Commands	Results/Outputs
-e	True
--	Underflow!
-+ D	D
-+ L	DL
-+ S	DLS
-+ U	DLSU
-t	U
-+ M	Overflow!
--	DLS i = U
-C	Stack Empty
-Q	-

4. Modify the test program so that the max element is equal to 3. Perform 4 push (s, i) operations, what will be the result? What will be the contents of the stack?

IV. SUPPLEMENTARY EXERCISES:

Consider the following Stack ADTs,

`bool isFull(stack s)`
Precondition: None
Postcondition: Returns TRUE if a stack is full. Otherwise, returns FALSE.

`void Display(stack s)`
Precondition: None
Postcondition: Outputs the data items in a stack. If the stack is empty it will display the message "Empty STACK".

1. Create a function prototype for each ADT,
2. and provide a test plan using the commands listed in the table below:

Command	Expected Result
- f	Stack is FULL
- s	D L S U

V. Machine Problem

1. Create a program that will simulate conversions from infix to postfix notations. Examples are provided below.

Example 1:

Given Infix expression: **A + B * C**

Simulation:

	SYMB	POSTFIX	OPSTK
		STRING	
1	A	A	
2	+	A	+
3	B	AB	+
4	*	AB	+*
5	C	ABC	+*
6		ABC*	+
7		ABC*+	

Example 2:

Given Infix expression : **(A + B) * C**

Simulation Table:

	SYMB	POSTFIX STRING	OPSTK
1	(		(
2	A	A	(
3	+	A	(+
4	B	AB	(+
5	)	AB+	
6	*	AB+	*
7	C	AB+C	*
8		**AB+C***	

Note: *The instructor may change the machine problem.*

Stack ADT

DATA & RESULTS SHEET

(Tentative Laboratory Report)

Name: ____

Schedule:____________________________ Section:__________________

Date: ______________________________ Grade: __________________

Test & Debug:

<table>
<tr><th>Step No.</th><th>Answers/Results</th></tr>
<tr><td>1&2</td><td>Attach source codes</td></tr>
<tr><td>3</td><td>
<table>
<tr><th>Commands</th><th>Results/Outputs</th></tr>
<tr><td>-e</td><td>True</td></tr>
<tr><td>--</td><td>Underflow!</td></tr>
<tr><td>-+ D</td><td>D</td></tr>
<tr><td>-+ L</td><td>DL</td></tr>
<tr><td>-+ S</td><td>DLS</td></tr>
<tr><td>-+ U</td><td>DLSU</td></tr>
<tr><td>-t</td><td>U</td></tr>
<tr><td>-+ M</td><td>Overflow!</td></tr>
<tr><td>--</td><td>DLS</td></tr>
<tr><td>-C</td><td>Stack Empty</td></tr>
<tr><td>-Q</td><td>-</td></tr>
</table>
Note: Inform your instructor when you’re finished with these test runs.</td></tr>
<tr><td>4</td><td>Result:
Stack (s):</td></tr>
</table>

Supplementary Exercises:

Supplementary Problem No.	Completed?		Remarks
	YES	NO	
1.a			
1.b			
2			

Machine Problem:

Topic	Completed?		Remarks
	YES	NO	

Note: Check the column YES if completed otherwise check the column NO.(for instructors use only)

INSTRUCTOR'S SIGNATURE: ____________________________