National University of Computer & Emerging Sciences (NUCES) Islamabad,
Department of Computer Science

DATA STRUCTURES – FALL 2021

LAB 09

Learning Outcomes

In this laboratory, you will implement the Applications of Stack.

Stack ADT

A stack is an abstract data type that serves as a collection of elements. Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack. This feature makes it LIFO data structure. In the previous lab we have familiarized ourselves with the operations and basic working of stack. In this lab we will implement some practical applications of stack.

NOTE: You are allowed to use the stack ADT implementation from the previous lab.

TASK 1

Write a function that determines whether the order of the parenthesis pairs in an input expression is correct and is therefore balanced?

For example:
{ [3 + (5 * 4) / (2 * 3)] + [(2 * 2) + (9 * 1)] } is a balanced expression whereas
{ 3 + (5 * 4) / (2 * 3)] + [(2 * 2 + (9 * 1)] } is not a balanced expression.
Your function must return True in case of a balanced expression and False otherwise.

Flex your Neurons!

1. Should you bother yourself with the numbers and operators in an expression? Or the parenthesis only?
2. '(' ')','{' '}' & '[' ']' are parenthesis pairs.

TASK 2

One of the applications of Stack is in the conversion of arithmetic expressions in high-level programming languages into machine readable form. As our computer can only understand and work on a binary language, it assumes that an arithmetic operation can take place in two operands only e.g., A+B, C*D, D/A etc. But in our usual form an arithmetic expression(infix notation) may consist of more than one operator and two operands e.g. (A+B)*C(D/(J+D)). These complex arithmetic operations can be converted into postfix notation using stacks which then can be executed in two operands and an operator form.

Input Infix Expression= **((3+4) *(5/6 - 7))**

Scanned Input	Stack	Postfix Expression	Description
(	(		push parenthesis
(	((		push parenthesis
3		3	add operand to expression
+	((+	3	stack top has parenthesis, push operator
4	((+	34	add operand to expression
)	(	34+	ending parenthesis occurred,pop till its starting is found in stack
*	(*	34+	stack top has parenthesis, push operator
(	(*(	34+	push parenthesis
5	(*(	34+5	add operand to expression
/	(*(/	34+5	stack top has parenthesis, push operator
6	(*(/	34+56	add operand to expression
-	(*(-	34+56/	stack top has operator having higher precedence than scanned input, pop stack,push scanned operator
7	(*(-	34+56/7	add operand to expression

)	**(***	**34+56/7-**	**ending parenthesis occurred,pop till its starting is found in stack**
)		**34+56/7-***	**ending parenthesis occurred,pop till its starting is found in stack**
END	**EMPTY**	**FINAL POSTFIX EXP**	

Output Postfix Expression= **34+56/7-***

Keeping in mind the discussion above, write a program that converts an infix expression into a postfix expression.

TASK 3

Implement a function getBinary() that takes input a decimal number and returns its binary.

Make use of a stack to store the binary digits as they get computed. Note the order of the binary digits being pushed to the stack, the last binary digit computed will end up at the top of the stack, hence giving us the correct binary sequence when it is popped from top to bottom. Add the popped digit to a string until the stack becomes empty, return the string as output.