Data Structure Lab Assignment (CS 2172)

Assignment 2:Stack

Time: 1 week

1. Implementation of an Integer-Stack

   In this assignment you are required to implement a stack where integer data can be pushed and popped. You should define (typedef) an appropriate type called ***stack*** such that multiple variables of type ***stack*** can be defined.

   Your implementation should support the following functions (interface) for the stack.

   1. ***intinitIntegerStack(stack s, intstackSize)*** - This allocates space for the stack to hold maximum "***stackSize***" number of integers and initializes that space.

   2. ***intpushIntegerStack(stack s, int d)*** - It pushes the data ***d*** in the stack ***s***. It returns 1 if the operation is successful. If the operation fails (say, when stack ***s*** is full and ***d*** cannot be pushed), the function returns 0.

   3. ***intpopIntegerStack(stack s, int *dp)*** - It pops from the stack ***s*** and stores the popped element at address ***dp***. It returns 1 if the operation is successful. If the operation fails (when stack s is empty and ***popIntegerStack()*** is attempted), the function returns 0.

   4. ***intfreeIntegerStack(stack s)*** - It frees the space allocated for stack ***s***. It returns 1 if the operation is successful. If the operation fails (say, s does not refer to a valid stack), the function returns 0.

   5. ***intisIntegerStackFull(stack s)*** - It returns 1 if the stack associated with ***s*** is full. The function returns 0 otherwise. If ***s*** does not refer to a valid stack then too the function returns 1.

   6. ***intisIntegerStackEmpty(stack s)*** - It returns 1 if the stack associated with ***s*** is empty. The function returns 0 otherwise. If ***s*** does not refer to a valid stack then too the function returns 1.

   Write a suitable main() function to demonstrate that your functions are working as desired.

2. Using the above Stack implementation, simulate the following

   Assume two stacks are created as ***stack1*** and ***stack2*** of size **N** and **M**, respectively. Also, assume that a series of integers are read from the user and pushed into ***stack1*** first; if ***stack1*** is full, then push into ***stack2***. This process continues till ***stack2*** is not full. Once ***stack2*** is full, it should pop all the elements from ***stack2*** and then ***stack1***. Every time an element is popped should be printed in the console.

3. Using the above Stack implementation, check the sanity of a mathematical expression with different kinds of parenthesis. The application results are***correct*** or ***incorrect*** based on the matching parenthesis only.
   For example "`{ ( A + B ) * C } + D`" and "`( [ A + B ] - C )`" are a correct expression, whereas "`[ ( A + B )`"and "`( A + B ) } - ( C + D`"are incorrect.