Indian Institute of Technology Kanpur
ESO207 : Data Structures and Algorithms
Programming Assignment 1

---

1. **Evaluating a Prefix Expression**
   Unlike humans, computers are fond of Prefix expressions. Evaluating a prefix expression can be much faster that processing the equivalent infix expression. Since prefix form does not need brackets to realize the associativity of operators.

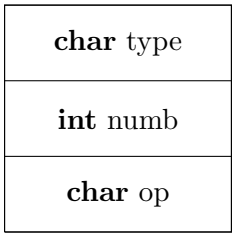
   You are required to write two related programs. In the first program you will implement a Stack and its functionality will be tested. In the second program the same stack will be used. This program will evaluate the input pre-fix arithmetic expression.

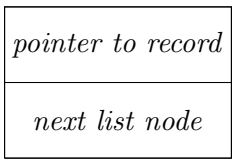
Program 1: (Implement a stack)

In this program you will implement a stack using a linked list. Each node (called *list-node*) in the linked list will have two fields: *item* and *next*. First will carry the item and the second will carry the pointer to the next *list-node* in the list.

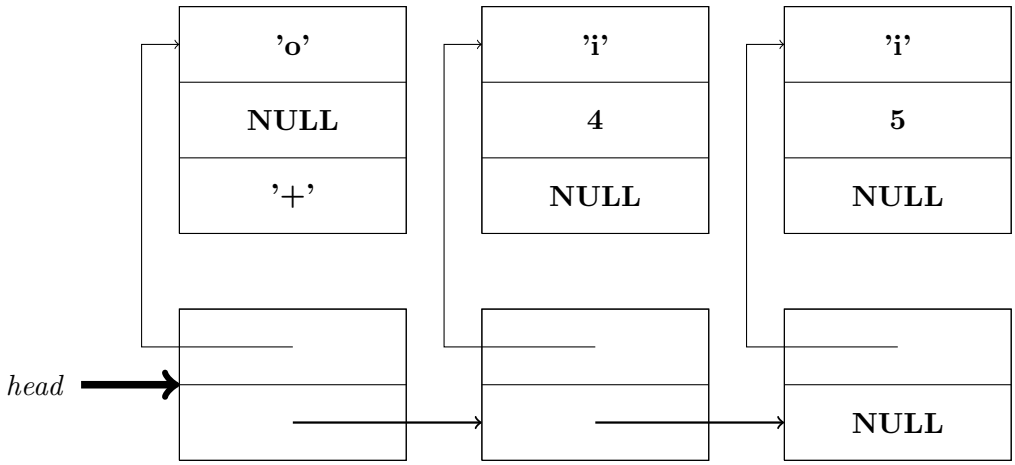
The item will be either a floating point number or one of the four binary operator symbols $\{+, -, *, /\}$. Each item will be stored in a record, called *numop*, described below. The item-field in a *list-node* will contain only a pointer to a "numop" record.

The *numop* record will be as follows. It will have three fields: *type*, *num*, and *op*. The *type* field will store the letter "o" if the record is to store an operator symbol. It will store the letter "n" if the record is to store a floating point number. In a *numop* record if the *type* field has stored "o", then its number field will carry NIL and the operator field will carry one of the operator symbols. Similarly if the *type* has "n", then the *num* field will carry a number and the *op* field will be NIL.





Following diagram shows a stack containing $+$ **4 5** with $+$ at the top.



Now, Implement a stack (using the above definition of linked list) with following functionalities. Let S be a pointer to a stack.

- isEmpty(S): checks if the stack pointed by S is empty. Return type: **int** (1 if empty, 0 o/w)
- Push(node* key): pushes key to the top of stack. Return type: **void**
- Top(): reads the top of stack and returns it (does NOT modify the stack). Return type: **numop***
- Pop(): deletes the top element of stack and returns it. Return type: **numop***
- CreateStack(): creates a pointer of type **numop***, sets it to equal to NIL and returns it. Return type: **numop***.

Now we will describe the main program which will use the above created structure. The main program will first create a stack and then it will read a list of coded-instructions which are described below. The main program must execute these instructions in sequence. The list of code-instructions are

1 n : means push number n to the stack and print a special character '$'.

2 symbol : means push the symbol if it is one of the operator symbols and print a special character '$'.

3 print the content of record at the top of stack, print "error" if stack is empty

4 print 'true' if the stack is empty, 'false' otherwise

5 remove the top node of the stack and print its content.

First line of input will be a single integer **Q**, the number of instructions you have to process. Also each instruction's output should be printed in a newline, except push.

Sample Tests:

.	input	output
	5	
	3	error
	1 4	$
	2 +	$
	3	+
	4	false

Program 2: Evaluation of a Pre-fix Expression

This program will use the Stack routines and write a new main program.

This program will read a prefix expression as input. The first entry will be the left most item of the prefix expression. The input will use the codes given above.

The main program will first create a stack and enter the input expression into it with the left most entry of the expression being at the top. The input will use the first two codes given above. Next it will call a subroutine called *Evaluate(S)*. Finally it will pop the top item from the stack and output its content and stop.

The subroutine *Evaluate(S)* will be a recursive program which will evaluate the expression sitting at the top of the stack and put the evaluated value back into the stack.

The expressions in the input will be a space-separated string of characters, which can either be a single digit $\{0, 1, \ldots, 9\}$ or a symbol $\{+, -, *, /\}$

Example test cases:

> Case 1: + * 4 5 6
> the output for this case should be **26**.
>
> Case 2: / 6 - 5 3
> the output for this case should be **3**.

How to submit?
We will be sending you the contest link and the related procedure within this week. You will have 7 days to finish the assignment. The allowed language will ONLY be **C**.