1- **(Recursively Search a List)** Write a function searchList that recursively searches a linked list for a specified value. The function should return a pointer to the value if it is found; otherwise, NULL should be returned. Use your function in a test program that creates a list of integers. The program should prompt the user for a value to locate in the list.

```
Enter your choice:
1 to insertItem an element into the list.
2 to recursively search list for an element
3 to end.
? 1
Enter an integer: 7
The list is:
7 --> NULL
? 1
Enter an integer: 99
The list is:
7 --> 99 --> NULL
? 1
Enter an integer: 56
The list is:
7 --> 56 --> 99 --> NULL
? 1
Enter an integer: 73
The list is:
7 --> 56 --> 73 --> 99 --> NULL
? 2
Enter integer to recursively search for: 7
7 is in the list.
? 2
Enter integer to recursively search for: 55
55 is not in the list.
? 2
Enter integer to recursively search for: 99
99 is in the list.
? 3
End of run.
```

2. (**Prefix to Postfix Conversion**)

Prefix : An expression is called the prefix expression if the operator appears in the expression before the operands. Simply of the form (operator operand1 operand2).
Example : *+AB-CD (Infix : (A+B) * (C-D))

Postfix: An expression is called the postfix expression if the operator appears in the expression after the operands. Simply of the form (operand1 operand2 operator).
Example : AB+CD-* (Infix : (A+B * (C-D))

Given a Prefix expression, convert it into a Postfix expression.

Help:

- Read the Prefix expression in reverse order (from right to left)
- If the symbol is an operand, then push it onto the Stack
- If the symbol is an operator, then pop two operands from the Stack
  Create a string by concatenating the two operands and the operator after them.
  string = operand1 + operand2 + operator
  And push the resultant string back to Stack
- Repeat the above steps until end of Prefix expression.

```
Input :  Prefix :  *+AB-CD
Output : Postfix : AB+CD-*
Explanation : Prefix to Infix :  (A+B) * (C-D)
              Infix to Postfix :  AB+CD-*

Input :  Prefix :  *-A/BC-/AKL
Output : Postfix : ABC/-AK/L-*
Explanation : Prefix to Infix :  A-(B/C)*(A/K)-L
              Infix to Postfix : ABC/-AK/L-*
```

3. (**Delete middle element of a stack**) Given a stack with push(), pop(), empty() operations, delete middle of it without using any additional data structure.

```
Input  : Stack[] = [1, 2, 3, 4, 5]
Output : Stack[] = [1, 2, 4, 5]

Input  : Stack[] = [1, 2, 3, 4, 5, 6]
Output : Stack[] = [1, 2, 4, 5, 6]
```