LAB TASK - 4

a) Implementation of stack using array.

In Array implementation, the stack is formed by using the array. All the operations regarding stack are performed using arrays.

Algorithm for push operation:-

```
① begin
② if top = n then stack full
③    top = top+1
④ stack(top) = data
⑤    end.
```

Time complexity O(1).

Algorithm for pop operation:-

```
① begin.
② if top = 0
   then stack empty;
③ item = stack (top);
④ top = top-1;
⑤ end.
```

Time complexity : O(1)

```
switch (choice)
 {
  case 1 :
      {  push ();
         break;
      }
  case 2 :
      {
         pop ();
         break ;
      }
  Case 3 :
      {
         show () ;
         break ;
      }
  case 4:
    {
      printf ("Exiting.....");
      break;
    }
 default:
    {
     printf ("Enter a valid choice");
    }
  }
 }
}
```

```
void push ()
  {  int value;
   if (top = = n)
   printf ("\n overflow");
   else
     {
      printf ("enter the value ?");
      scanf (%d", &value);
      top = top + 1;
      stack [top] = value;
     }
   }
void pop ()
 {
   if (top = = -1)
  printf (" underflow");
  else
    top = top -1;
 }
void show ()
{
 for (i = top ; i>0 ; i--)
 { printf (" %d\n", stack [i]);
 }
 if (top = = -1) {
```

Algorithms for peek operation

```
① Begin
② if top = -1 then stack empty
③ item = stack (top)
④ return (item)
⑤ end.
```

time complexity : O(n)

Program :-

```
#include<stdio.h>
int stack [100] , i, j , choice=0 ,n, top = -1;
void push();
void pop();
void show();
void main ()
{
 printf ("Enter the number of elements instack");
 scanf ("%d", &n);
 printf ("----- stack operations using array----");

 while (choice != 4)
 {
  printf (" choose one from below options... \n");
  printf ("\n 1.push \n 2.pop \n 3.show \n 4.Exit");
  printf (" Enter your choice:");
  scanf (" %d", & choice);
```

```
    printf ("stack is empty");
  }
}
```

b) <u>Conversion of infix expression to postfix expression</u>:

<u>Algorithm</u>:-

1. Traversing the given expression from left to right should begin.

2. Just output the scanned character if it is an operand.

3. Else
   - If operand's precedence is greater than the operator's precedence in the stack (or the stack is empty or has '('), then push the operator into stack.
   - Else any operator with more or equal precedence than the traversed operator are popped. Push these scanned operator after you pop them.

4) Push the scanned character if it is a '('.

5. If the scanned operator is ')', pop the stack and output it until another '(' appears then eliminate both the parentheses.

d) Solving Tower of Hanoi problem using recursion.

```
#include <stdio.h>

void move(int n, int source , int destination,
              int auxilary)
{
   if (n==1) {
      printf("move disk 1 from source %d to
              destination %d\n", n, source, destination);
   return;
  }
move (n-1, source auxilary , destination);
printf("move disk %d from source %d to destination
         %d\n", n, source, destination);
move (n-1, auxilary , destination, source);
}

int main (){
     int n;
     printf("Enter the no of disks:");
     scanf ("%d", &n);
     move (n, 1, 3, 2);
   return 0;
  }
```

Time complexity for this problem is $O(2^n - 1)$ & Space complexity is $O(n)$

6. steps 2 through 6 should now be repeated until the entire characters, is scanned.
7. printing results.
8. pop and print until the stack is not empty.

(c) Evaluation of postfix expression :

Algorithm :-

Read in one symbol at a time from the postfix expression.

① Any time you see an operand, push it onto the stack

② Any time you see a binary operator (+, -, *, /) or unary (square root, negative sign) operator
- If the operator is binary, pop two elements off the stack.
- If the operator is unary, pop one element off the stack.
- Evaluate those operands with that operator.
- Push the result back onto the stack.

when you are done with the entire expression, the only thing left on the stack should be the final result. If there are zero or more than 1 operands left on the stack, either your program is flawed, or the expression is invalid.