1. Implementation of Stack Operations Implemented by Array

a. Push elements in the stack.
b. Pop elements from the stack.
c. Print the stack top element.

Program –

```
#include<stdio.h>
int stack[10],choice,n,top,x,i; // Declaration of
variables

void push();
void pop();
void display();

int main()
{
     top = -1;     // Initially there is no element in
stack
     printf("\n Enter the size of STACK : ");
     scanf("%d",&n);
     printf("\nSTACK IMPLEMENTATION USING ARRAYS\n");
     do
     {
         printf("\n1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n");
          printf("\nEnter the choice : ");
          scanf("%d",&choice);
          switch(choice)
          {
               case 1:
                         {
                              push();
                              break;
                         }
               case 2:
                         {
                              pop();
                              break;
                         }
               case 3:
```

```
                              {
                                   display();
                                   break;
                              }
                    case 4:
                              {
                                   break;
                              }
                     default:
                              {
                                   printf ("\nInvalid
Choice\n");
                              }
               }
          }while(choice!=4);
          return 0;
}

void push()
{
          if(top >= n - 1)
          {
               printf("\nSTACK OVERFLOW\n");
          }
          else
          {
               printf("Enter a value to be pushed : ");
               scanf("%d",&x);
               top++;              // TOP is incremented after
an element is pushed
               stack[top] = x;   // The pushed element is made
as TOP
          }
}

void pop()
{
          if(top <= -1)
          {
               printf("\nSTACK UNDERFLOW\n");
```

```
        }
        else
        {
                printf("\nThe popped element is
%d",stack[top]);
                top--;     //  Decrement TOP after a pop
        }
}

void display()
{
        if(top >= 0)
        {
                //  Print the stack
                printf("\nELEMENTS IN THE STACK\n\n");
                for(i = top ; i >= 0 ; i--)
                        printf("%d\t",stack[i]);
        }
        else
        {
                printf("\nEMPTY STACK\n");
        }
}
```

2. Implementation of Stack Operations Implemented by Linked List

a. Push elements in the stack.
b. Pop elements from the stack.

Program –

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int info;
    struct node* next;
};
```

```
void push(struct node**, int);
int pop(struct node**);

int main()
{
    struct node* top = NULL;
    int ch, n;

    while(1) {
        printf("1.Push\n 2.Pop\n 3.Exit\n");
        printf("Enter your choice: ");
        scanf("%d",&ch);

        switch(ch)
        {
            case 1:
                printf("Enter data to push: ");
                scanf ("%d",&n);
                push(&top, n);
                break;

            case 2:
                n = pop(&top);
                if (n!=-9999)
                    printf("The popped element is:
%d",n);
                break;

            case 3:
                exit(1);

            default:
                printf("Invalid choice\n");
        }
    }
}

void push(struct node** top, int item)
{
    struct node* new_node;
```

```
    new_node = (struct node*)malloc(sizeof(struct node));
    new_node -> info = item;
    new_node -> next = *top;
    *top = new_node;
    return;
}

int pop(struct node** top)
{
    int item;
    struct node* temp;
    if(*top==NULL) {
        printf("Stack is empty\n");
        return(-9999);
    }

    item = (*top) -> info;
    temp = *top;
    *top = (*top) -> next;
    temp -> next = NULL;

    free(temp);
    return(item);
}
```

3. Write a program to implement the infix to postfix algorithm.

Program –

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// Stack type
struct Stack
{
     int top;
     unsigned capacity;
```

```
    int* array;
};

// Stack Operations
struct Stack* createStack( unsigned capacity )
{
    struct Stack* stack = (struct Stack*)
        malloc(sizeof(struct Stack));

    if (!stack)
        return NULL;

    stack->top = -1;
    stack->capacity = capacity;

    stack->array = (int*) malloc(stack->capacity *
                                 sizeof(int));

    return stack;
}
int isEmpty(struct Stack* stack)
{
    return stack->top == -1 ;
}
char peek(struct Stack* stack)
{
    return stack->array[stack->top];
}
char pop(struct Stack* stack)
{
    if (!isEmpty(stack))
        return stack->array[stack->top--] ;
    return '$';
}
void push(struct Stack* stack, char op)
{
    stack->array[++stack->top] = op;
}
```

```
// A utility function to check if
// the given character is operand
int isOperand(char ch)
{
     return (ch >= 'a' && ch <= 'z') ||
          (ch >= 'A' && ch <= 'Z');
}

// A utility function to return
// precedence of a given operator
// Higher returned value means
// higher precedence
int Prec(char ch)
{
     switch (ch)
     {
     case '+':
     case '-':
          return 1;

     case '*':
     case '/':
          return 2;

     case '^':
          return 3;
     }
     return -1;
}


// The main function that
// converts given infix expression
// to postfix expression.
int infixToPostfix(char* exp)
{
     int i, k;

     // Create a stack of capacity
     // equal to expression size
```

```
    struct Stack* stack = createStack(strlen(exp));
    if(!stack) // See if stack was created successfully
        return -1 ;

    for (i = 0, k = -1; exp[i]; ++i)
    {

        // If the scanned character is
        // an operand, add it to output.
        if (isOperand(exp[i]))
            exp[++k] = exp[i];

        // If the scanned character is an
        // ‘(‘, push it to the stack.
        else if (exp[i] == '(')
            push(stack, exp[i]);

        // If the scanned character is an ‘)’,
        // pop and output from the stack
        // until an ‘(‘ is encountered.
        else if (exp[i] == ')')
        {
            while (!isEmpty(stack) && peek(stack) !=
'('
                exp[++k] = pop(stack);
            if (!isEmpty(stack) && peek(stack) != '(')
                return -1; // invalid expression

            else
                pop(stack);
        }
        else // an operator is encountered
        {
            while (!isEmpty(stack) &&
                Prec(exp[i]) <= Prec(peek(stack)))
                exp[++k] = pop(stack);
            push(stack, exp[i]);
        }

    }
```

```
        // pop all the operators from the stack
        while (!isEmpty(stack))
             exp[++k] = pop(stack );

        exp[++k] = '\0';
        printf("\n\nResult :: ");
        printf( "%s", exp );
    }

    // Driver program to test above functions
    int main()
    {
        int size=100;
        char exp[size];
        printf("Insert expression :: ");
        gets(exp);
        infixToPostfix(exp);
        return 0;
    }
```

4. Write a program to implement the postfix evaluation algorithm.

Program –

```
#include<stdio.h>     //standard input output functions
#include<conio.h>        //console functions
#include<string.h>       //string functions
#define MAX 50              //max size defined
int stack[MAX];             //a global stack
char post[MAX];             //a global postfix stack
int top=-1;                  //initializing top to -1
void pushstack(int tmp);       //push function
void evaluate(char c);          //calculate function
void main()
{
   int i,l;
   //clrscr();
   printf("Insert a postfix notation :: ");
```

```
   gets(post);                    //getting a postfix
expression
   l=strlen(post);               //string length
   for(i=0;i<l;i++)
   {
      if(post[i]>='0' && post[i]<='9')
      {
          pushstack(i);             //if the element is a
number push it
      }
      if(post[i]=='+' || post[i]=='-' || post[i]=='*' ||
      post[i]=='/' || post[i]=='^')       //if element is
an operator
      {
          evaluate(post[i]);             //pass it to the
evaluate
      }
   }                      //print the result from the top
   printf("\n\nResult :: %d",stack[top]);
   getch();
}

void pushstack(int tmp)          //definiton for push
{
   top++;                              //incrementing top
   stack[top]=(int)(post[tmp]-48);    //type casting the
string to its integer value
}

void evaluate(char c)       //evaluate function
{
   int a,b,ans;        //variables used
   a=stack[top];       //a takes the value stored in the
top
   stack[top]='\0';     //make the stack top NULL as its
a string
   top--;                //decrement top's value
   b=stack[top];         //put the value at new top to b
   stack[top]='\0';      //make it NULL
   top--;                //decrement top
```

```
    switch(c)     //check operator been passed to evaluate
    {
       case '+':          //addition
           ans=b+a;
           break;
       case '-':           //subtraction
           ans=b-a;
           break;
       case '*':            //multiplication
           ans=b*a;
           break;
       case '/':           //division
           ans=b/a;
           break;
       case '^':      //power
           ans=b^a;
           break;
       default:
           ans=0;      //else 0
    }
    top++;             //increment top
    stack[top]=ans;        //store the answer at top
}
```