Stacks

1. Write a program to simulate the working of stack using an array with the following:
a) Push b) Pop c) Display.

⇒

```
#include<stdio.h>
void push(int);
void pop();
void display();
int stack[SIZE], top=-1;

void push(int value)
{
    if(top==SIZE-1)
    {
        printf("overflow");
    }
  else
   {  top=top+1;
      stack[top]=value;
      printf("Insertion Successful");
   }
}
```

```
void pop()
{
  if(top == -1)
  {
    printf("Stack is Empty");
  }
  else
  {
    printf("the deleted element is %d"), stack[top]);
      top = top - 1;
    }
}
void display()
{
    int i;
    if(top == -1)
    {
      printf("stack is empty");
    }
   for(i=top; i>=0; i--)
   {
     printf("%d ", stack[i]);
   }
}
int main()
{
   int value, choice;
   while(1)
{
   printf("\n1.PUSH, 2.POP, 3.DISPLAY, 4.EXIT");
    scanf("%d", &choice);
   switch(choice)
   {  case 1: printf("enter a value");
```

```
scanf("%d", &value);
push(value);
break;
Case 2: pop();
        break;
Case 3: display();
        break;
Case 4: exit(0);
default: printf("wrong input");
}
```

Output

```
stack = Stack(5)
stack.push(1)
stack.push(2)
stack.push(3)
stack.display()
stack.pop()
```

1 pushed into the stack
2 pushed into the stack
3 pushed into the stack.

stack elements:
3
2
1
popped item : 3

2. Write a Program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

=>

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100

typedef struct {
    char items[MAX_SIZE];
    int top;
} Stack;

void push(Stack *s, char c) {
    if (s->top == MAX_SIZE - 1) {
        printf("stack overflow");
        return;
    }
    s->items[++(s->top)] = c;
}

char pop(Stack *s) {
    if (s->top == -1) {
        printf("\nstack underflow");
        return -1;
    } return s->items[(s->top)--];
}

int precedence(char operator)
{ if (operator == '+' || operator == '-') {
    return;
```

```
} else if (operator == '*' || operator == '/') {
        return 2;
    }
return 0;
}
void infix_to_postfix(char *infix, char *postfix)
{
    Stack stack;
    stack.top = -1;
    int i=0, j=0;
    while(infix[i] != '\0') {
    if (isalnum(infix[i])) {
        postfix[j++] = infix[i++];
    } else if (infix[i] == '(') {
        push(&stack, infix[i++]);
    }
    else if (infix[i] == ')') {
        while (stack.top != -1 && stack.items[stack.top]
                != '(') {
            postfix[j++] = pop(&stack);
    }
    if (stack.top != -1 && stack.items[stack.top]
    == '(') {
        pop(&stack);
    } i++; }
    else {
        while(stack.top != -1 && precedence
        (stack.items[stack.top]) >= precedence(
        infix[i])) {
```

```
        postfin[j++] = pop(&stack);
    }push(&stack, infin[i++]);
    }
}
while(stack.top != -1){
    postfin[j++] = pop(&stack);
}
postfin[j] = '\0';
}
int main(){
    char infin[MAX_SIZE];
    char postfin[MAX_SIZE];
printf("Enter infin expression");
scanf("%s", infin);
infin_to_postfin(infin, postfin);
printf("postfin expression: %s", postfin);
return 0;
}
```

Output:

Enter infin expression: (A+B)*C-(D/E)

postfin expression: AB+C*DE/-