Week - 3

STACKS

(1) Write a program to estimate the working of stack using array with the following.

(a) RUN (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){
     pf("overflow");
  }
  else
  {
    top = top + 1;
    stack[top] = value;
    pf("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");
  }
  else{
    for(i=top; i>=0; i--){
      printf("%d", stack[i]);
  } }
int main(){
   int value, choice;
   while(1){
     pf("1. push, 2. POP, 3. Display, 4. Exit");
     sf("%d", &choice);
     switch(choice){
     case 1: pf("Enter a value");
             sf("%d", &value);
             push(value);
             break;
```

```
    case 2: pop();
            break;
    case 3: display();
            break;
    case 4: exit(0);
    default: printf("wrong input");
}
```

Output:

```
stack = stack();
stack.push(1)
stack.push(2)
stack.push(3)
stack.display();
stack.pop()
1 pushed into stack
2 pushed into stack
3 pushed into stack
stack elements:
3
2
1
poped item: 3.
```

2) WAP to convert a given valid parenthesised infix arithematic expression to postfix expression

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
typedef struct stack {
        char items[MAX_SIZE];
        int top;
    } stack;
void push(stack s, char c){
    if (s->top == MAX_SIZE-2){
      printf("stack overflow");
      return;
    }
  s->items[++(s->top)] = c;
}
char pop(stack *s){
   if (s->top == -1){
       printf("stack underflow");
       return -1;
    }
  return s->items[(s->top)--];
}
int precedence(char operator){
  if (operator == '+' || operator == '-'){
     return 1;
  }
 else if (operator == '*' || operator == '/'){
     return 2;
  }
  return 0;}
```

```
void infix_topostfix (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)
    i++;
  else {
     while (stack.top != -1 && precedence
        (stack.items[stack.top]) >= precedence
          (infix[i])){
        postfix[j++] = pop (&stack);
      }
      push (&stack, infix[i++]);
     }
while (stack.top != -1){
      postfix[i++] = pop (&stack);
  }
  postfix[i] = '\0';
  }
```

```
int main(){
  char infix[MAX_SIZE];
  char postfix[MAX_SIZE];
  printf("Enter infix expression");
   scanf("%s", &infix);
   infix_to_postfix (infix, postfix);
   printf("Postfix expression : %s", postfix);
   return 0;
  }
```

o/p:

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

postfix expression : AB+ -DE/*