Practical No.6

1) Implement a Stack and perform the stack operations: Infix to Postfix, Infix to Prefix, Evaluation of Postfix Expression, Print using Menu Driver Program such as :-
   1.Infix to Postfix
   2.Infix to Prefix
   3.Evaluation of Postfix Expression
   4.Exit.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#define MAX 100
char stack[MAX];
int top = -1;
char expression[MAX];
char result[MAX];
void push(char item) {
   if (top == MAX - 1) {
      printf("Stack overflow\n");
      return;
   }
   stack[++top] = item;
}
char pop() {
   if (top == -1) {
      printf("Stack underflow\n");
      return '\0';
   }
   return stack[top--];
}
```

```
char peek() {
    return stack[top];
}
int isEmpty() {
    return top == -1;
}
int precedence(char ch) {
    switch (ch) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
    }
    return -1;
}
void infixToPostfix() {
    int k = 0;
    for (int i = 0; expression[i]; i++) {
        char c = expression[i];
        if (isalnum(c)) {
            result[k++] = c;
        } else if (c == '(') {
            push(c);
        } else if (c == ')') {
            while (!isEmpty() && peek() != '(') {
```

```
            result[k++] = pop();
        }
        pop();
    } else {
        while (!isEmpty() && precedence(c) <= precedence(peek())) {
            result[k++] = pop();
        }
        push(c);
    }
  }
  while (!isEmpty()) {
    result[k++] = pop();
  }
  result[k] = '\0';
}
void reverse(char *exp) {
  int length = strlen(exp);
  for (int i = 0; i < length / 2; i++) {
    char temp = exp[i];
    exp[i] = exp[length - i - 1];
    exp[length - i - 1] = temp;
  }
}
void replaceParentheses() {
  for (int i = 0; expression[i]; i++) {
    if (expression[i] == '(') expression[i] = ')';
    else if (expression[i] == ')') expression[i] = '(';
  }
}
```

```
void infixToPrefix() {
   reverse(expression);
   replaceParentheses();
   infixToPostfix();
   reverse(result);
}
int evaluatePostfix() {
   int valueStack[MAX];
   int valueTop = -1;
   for (int i = 0; result[i]; i++) {
      char c = result[i];
      if (isdigit(c)) {
         valueStack[++valueTop] = c - '0';
      } else {
         int val1 = valueStack[valueTop--];
         int val2 = valueStack[valueTop--];
         switch (c) {
            case '+':
               valueStack[++valueTop] = val2 + val1;
               break;
            case '-':
               valueStack[++valueTop] = val2 - val1;
               break;
            case '*':
               valueStack[++valueTop] = val2 * val1;
               break;
            case '/':
               valueStack[++valueTop] = val2 / val1;
```

```
                break;
            }
        }
    }
    return valueStack[valueTop];
}
int main() {
    int choice;
    while (1) {
        printf("\nChoose an operation:\n");
        printf("1. Infix to Postfix\n");
        printf("2. Infix to Prefix\n");
        printf("3. Evaluate Postfix Expression\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter infix expression: ");
                scanf("%s", expression);
                infixToPostfix();
                printf("Postfix expression: %s\n", result);
                break;
            case 2:
                printf("Enter infix expression: ");
                scanf("%s", expression);
                infixToPrefix();
                printf("Prefix expression: %s\n", result);
                break;
```

```
        case 3:
            printf("Enter postfix expression: ");
            scanf("%s", result);
            printf("Evaluation result: %d\n", evaluatePostfix());
            break;
        case 4:
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid choice! Try again.\n");
      }
   }
   return 0;
}
```