DATA STRUCTURES

Division	**CS(AIML) -A**
Batch	**2**
GR-no	**12311493**
Roll no	**54**
Name	**Atharva Kangralkar**

Assignment 9:

WAP to convert given Infix expression into its Equivalent Prefix and Postfix form using Stack.

Code:-

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

#define MAX 100

char stack[MAX];
int top = -1;

void push(char c) {
   stack[++top] = c;
}

char pop() {
   if (top == -1) return -1;
   return stack[top--];
}

char peek() {
   if (top == -1) return -1;
   return stack[top];
}

int precedence(char op) {
   if(op == '^') return 3;
   if(op == '*' || op == '/') return 2;
```

```
    if(op == '+' || op == '-') return 1;
    return 0;
}

int isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
}

void infixToPostfix(char* infix, char* postfix) {
    int i = 0, k = 0;
    char ch;
    while((ch = infix[i++]) != '\0') {
        if(isalnum(ch)) {
            postfix[k++] = ch;
        } else if(ch == '(') {
            push(ch);
        } else if(ch == ')') {
            while(peek() != '(') {
                postfix[k++] = pop();
            }
            pop()
        } else if(isOperator(ch)) {
            while(top != -1 && precedence(peek()) >= precedence(ch)) {
                postfix[k++] = pop();
            }
            push(ch);
        }
    }
```

```
    while(top != -1) {
        postfix[k++] = pop();
    }
    postfix[k] = '\0';
}

void reverse(char* str) {
    int i, j;
    char temp;
    for(i = 0, j = strlen(str) - 1; i < j; i++, j--) {
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;
    }
}

void infixToPrefix(char* infix, char* prefix) {
    char revInfix[MAX], revPostfix[MAX];
    int i;

    strcpy(revInfix, infix);
    reverse(revInfix);

    for(i = 0; revInfix[i] != '\0'; i++) {
        if(revInfix[i] == '(')
            revInfix[i] = ')';
        else if(revInfix[i] == ')')
            revInfix[i] = '(';
```

```
    }

    top = -1;
    infixToPostfix(revInfix, revPostfix);
    reverse(revPostfix);
    strcpy(prefix, revPostfix);
}

int main() {
    char infix[MAX], postfix[MAX], prefix[MAX];

    printf("Enter infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);
    infixToPrefix(infix, prefix);

    printf("Postfix: %s\n", postfix);
    printf("Prefix: %s\n", prefix);

    return 0;
}
```

Code Screenshot:-

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

#define MAX 100

char stack[MAX];
int top = -1;

void push(char c) {
    stack[++top] = c;
}

char pop() {
    if (top == -1) return -1;
    return stack[top--];
}

char peek() {
    if (top == -1) return -1;
    return stack[top];
}

int precedence(char op) {
    if(op == '^') return 3;
```

```
    if(op == '^') return 3;
    if(op == '*' || op == '/') return 2;
    if(op == '+' || op == '-') return 1;
    return 0;
}

int isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
}

void infixToPostfix(char* infix, char* postfix) {
    int i = 0, k = 0;
    char ch;
    while((ch = infix[i++]) != '\0') {
        if(isalnum(ch)) {
            postfix[k++] = ch;
        } else if(ch == '(') {
            push(ch);
        } else if(ch == ')') {
            while(peek() != '(') {
                postfix[k++] = pop();
            }
            pop()
        } else if(isOperator(ch)) {
            while(top != -1 && precedence(peek()) >= precedence(ch)) {
                postfix[k++] = pop();
            }
            push(ch);
        }
    }
    while(top != -1) {
        postfix[k++] = pop();
    }
```

```
59      postfix[k] = '\0';
60  }
61
62  void reverse(char* str) {
63      int i, j;
64      char temp;
65      for(i = 0, j = strlen(str) - 1; i < j; i++, j--) {
66          temp = str[i];
67          str[i] = str[j];
68          str[j] = temp;
69      }
70  }
71
72  void infixToPrefix(char* infix, char* prefix) {
73      char revInfix[MAX], revPostfix[MAX];
74      int i;
75
76      strcpy(revInfix, infix);
77      reverse(revInfix);
78
79      for(i = 0; revInfix[i] != '\0'; i++) {
80          if(revInfix[i] == '(')
81              revInfix[i] = ')';
82          else if(revInfix[i] == ')')
83              revInfix[i] = '(';
84      }
85
86      top = -1;
87      infixToPostfix(revInfix, revPostfix);
88      reverse(revPostfix);
89      strcpy(prefix, revPostfix);
90  }
91
92  int main() {
```

```c
int main() {
    char infix[MAX], postfix[MAX], prefix[MAX];

    printf("Enter infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);
    infixToPrefix(infix, prefix);

    printf("Postfix: %s\n", postfix);
    printf("Prefix: %s\n", prefix);

    return 0;
}
```

Output:-

```
Output
Preorder: 1 2 4 5 3 6 7
Inorder: 4 2 5 1 6 3 7
Postorder: 4 5 2 6 7 3 1

=== Code Execution Successful ===
```