AIM- to study the concept of infix to postfix and prefix

THEORY-

Notation Overview

Infix Notation: The operator is placed between operands (e.g., `A + B`)

Postfix Notation: The operator is placed after the operands (e.g., `A B +`).

Prefix Notation: The operator is placed before the operands (e.g., `+ A B`).

Characteristics

Infix: Readable and commonly used in arithmetic expressions but requires parentheses to enforce precedence.

Postfix: Eliminates the need for parentheses, as the order of operations is clear from the position of operators

Prefix: Similar to postfix in terms of eliminating parentheses, but the operators come before their operands.

Operator Precedence and Associativity-Before performing conversions, it is essential to understand operator precedence and associativity:

Precedence: Determines the order of operations (e.g., multiplication has higher precedence than addition).

Determines the order of operations for operators of the same precedence (e.g., left-to-right for addition and multiplication).

INPUT-

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
// Stack implementation for characters
#define MAX 100

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

void initStack(Stack *s) {
    s->top = -1;
}

int isEmpty(Stack *s) {
    return s->top == -1;
}

int isFull(Stack *s) {
    return s->top == MAX - 1;
}
```

```
void push(Stack *s, char item) {
    if (isFull(s)) {
        printf("Stack overflow\n");
        return;
    }
    s->items[++(s->top)] = item;
}

char pop(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack underflow\n");
        return '\0';  // Return null character
    }
    return s->items[(s->top)--];
}

char peek(Stack *s) {
    if (isEmpty(s)) {
        return '\0';  // Return null character
    }
    return s->items[s->top];
}

// Function to check precedence of operators
int precedence(char op) {
    switch (op) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        default:
            return 0;
    }
}

// Infix to Postfix Conversion
void infixToPostfix(char* infix, char* postfix) {
    Stack s;
    initStack(&s);
    int k = 0;  // Index for postfix expression
```

```
    for (int i = 0; i < strlen(infix); i++) {
        char ch = infix[i];

        if (isalnum(ch)) {
            postfix[k++] = ch;  // Add operand to postfix expression
        } else if (ch == '(') {
            push(&s, ch);
        } else if (ch == ')') {
            while (!isEmpty(&s) && peek(&s) != '(') {
                postfix[k++] = pop(&s);
            }
            pop(&s);  // Remove '(' from stack
        } else {  // Operator
            while (!isEmpty(&s) && precedence(ch) <= precedence(peek(&s))) {
                postfix[k++] = pop(&s);
            }
            push(&s, ch);
        }
    }

    while (!isEmpty(&s)) {
        postfix[k++] = pop(&s);
    }
    postfix[k] = '\0';  // Null-terminate the postfix expression
}

// Infix to Prefix Conversion
void infixToPrefix(char* infix, char* prefix) {
    // Reverse the infix expression
    int len = strlen(infix);
    char revInfix[MAX];
    char revPrefix[MAX];

    for (int i = 0; i < len; i++) {
        if (infix[i] == '(') {
            revInfix[len - i - 1] = ')';
        } else if (infix[i] == ')') {
            revInfix[len - i - 1] = '(';
        } else {
            revInfix[len - i - 1] = infix[i];
        }
    }
    revInfix[len] = '\0';
```

```
    // Convert reversed infix to postfix
    char revPostfix[MAX];
    infixToPostfix(revInfix, revPostfix);

    // Reverse the postfix expression to get the prefix expression
    int revLen = strlen(revPostfix);
    for (int i = 0; i < revLen; i++) {
        prefix[i] = revPostfix[revLen - i - 1];
    }
    prefix[revLen] = '\0';
}

int main() {
    char infix[MAX] = "A+B*C";
    char postfix[MAX];
    char prefix[MAX];

    // Convert infix to postfix
    infixToPostfix(infix, postfix);
    printf("Infix to Postfix: %s\n", postfix);

    // Convert infix to prefix
    infixToPrefix(infix, prefix);
    printf("Infix to Prefix: %s\n", prefix);

    return 0;
}
```

OUTPUT-

```
Infix to Postfix: ABC*+
Infix to Prefix: +A*BC


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

CONCLUSION-

In this way we understood the places operators between operands in easy and understandable manner