Intermediate Code Generation – Infix to Prefix and Postfix
Compiler Design – Lab 10

Name: Ojas Patil
Reg No: 21BAI1106

AIM

To write a C program to convert infix to prefix and infix to postfix.

Algorithm - Prefix

1. Set up necessary libraries and constants.
2. Define stack operations and global variables.
3. Implement precedence determination for operators.
4. Develop prefix conversion function.
5. Scan infix from left to right, appending operands to prefix.
6. Push/pull operators based on precedence rules.
7. Append remaining operators from stack to prefix.
8. Reverse prefix string to obtain final result.
9. Optionally, generate 3-address code from prefix expression.
10. In main function, prompt user for infix, call prefix conversion, and optionally generate 3-address code.

Algorithm - Postfix

1. Define libraries and constants.
2. Implement stack operations and global variables.
3. Determine operator precedence.
4. In main, prompt user for infix expression.
5. Scan infix from left to right.
6. Append operands to postfix string.
7. Handle parenthesis and operators according to precedence.
8. Append remaining operators from stack to postfix.
9. Print postfix expression.
10. Optional: evaluate postfix expression.

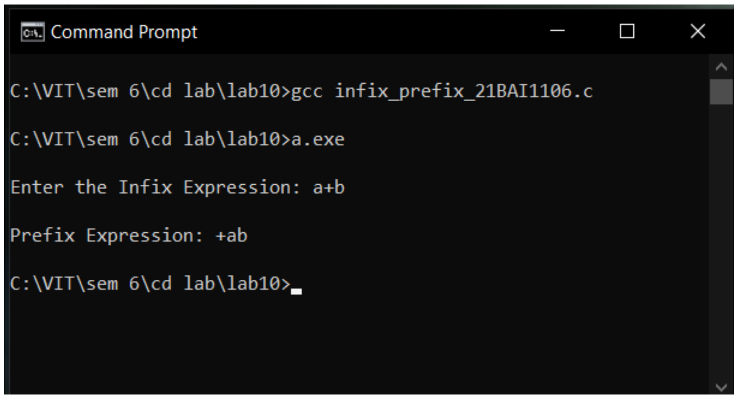
Explanation

These programs aim to convert infix expressions to their equivalent prefix and postfix notations. The prefix conversion algorithm initializes stacks for operators and operands, scans the infix expression from left to right, and employs stack-based operations to rearrange operators and operands according to precedence rules, ultimately producing the corresponding prefix expression. Similarly, the postfix conversion algorithm utilizes a stack to reorder operators and operands based on precedence, generating the postfix expression. Both algorithms ensure correct expression notation conversion through systematic scanning and stack manipulation, facilitating further analysis or evaluation of mathematical expressions.

Sample Input

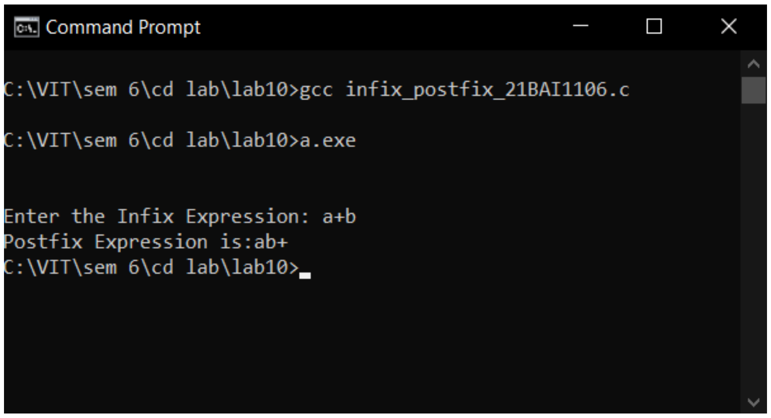
```
Enter the Infix Expression: a+b
```

Output Screenshot – Prefix



```
C:\VIT\sem 6\cd lab\lab10>gcc infix_prefix_21BAI1106.c

C:\VIT\sem 6\cd lab\lab10>a.exe

Enter the Infix Expression: a+b

Prefix Expression: +ab

C:\VIT\sem 6\cd lab\lab10>
```

Output Screenshot – Postfix



```
C:\VIT\sem 6\cd lab\lab10>gcc infix_postfix_21BAI1106.c

C:\VIT\sem 6\cd lab\lab10>a.exe


Enter the Infix Expression: a+b
Postfix Expression is:ab+
C:\VIT\sem 6\cd lab\lab10>
```

Source Code – Infix to Prefix

```c
#include <string.h>
#include <ctype.h>
#include <stdio.h>
#include <conio.h>
#define SIZE 50

char s[SIZE];
int top = -1;
int s1[SIZE];
int top1 = -1;
void push1(int elem)
{
    s1[++top1] = elem;
}
int pop2()
{
    return (s1[top1--]);
}
void push(char elem)
{
    s[++top] = elem;
}
char pop()
{
    return (s[top--]);
}
int pr(char elem)
{
    switch (elem)
    {
    case '#':
        return 0;
    case ')':
        return 1;
    case '+':
    case '-':
        return 2;
    case '*':
    case '/':
        return 3;
    }
    return 0;
}
```

```c
void prefix(char *infx, char *prfx)
{
    int i = 0, k = 0;
    char ch, elem;
    push('#');
    strrev(infx);
    while ((ch = infx[i++]) != '\0')
    {
        if (ch == ')')
            push(ch);
        else if (isalnum(ch))
            prfx[k++] = ch;
        else if (ch == '(')
        {
            while (s[top] != ')')
                prfx[k++] = pop();
            top--;
        }
        else
        {
            while (pr(s[top]) >= pr(ch))
                prfx[k++] = pop();
            push(ch);
        }
    }
    while (s[top] != '#')
        prfx[k++] = pop();
    prfx[k] = '\0';
    strrev(prfx);
    strrev(infx);
    printf("\nPrefix Expression: %s\n", prfx);
}
void caddress(char *prfx)
{
    int i, j = 0, k = 0, l = 0;
    char op1, op2;
    for (i = strlen(prfx) - 1; i >= 0; --i)
        if (prfx[i] != '+' && prfx[i] != '-' && prfx[i] != '*' && prfx[i] != '/'
&& prfx[i] != '=')
            push(prfx[i]);
        else
        {
            k = 0;
            l = 0;
            ++j;
            op1 = pop();
            op2 = pop();

            if (op1 == '$')
                k = pop2();
            if (op2 == '$')
                l = pop2();
            if (k == 0 && l == 0)
                printf("t%d=%c%c%c \n", j, op1, prfx[i], op2);
            if (k != 0 && l == 0)
                printf("t%d=t%d%c%c \n", j, k, prfx[i], op2);
            if (l != 0 && k == 0)
                printf("t%d=%c%ct%d \n", j, op1, prfx[i], l);
            if (l != 0 && k != 0)
                printf("t%d=t%d%ct%d \n ", j, k, prfx[i], l);

            push('$');
            push1(j);
        }
}
int main()
{
    char infx[50], prfx[50];
    printf("\nEnter the Infix Expression: ");
    scanf("%s", infx);
    prefix(infx, prfx);
    // printf("\n3-address code is:\n");
    // caddress(prfx);
    return 0;
}
```

Source Code – Infix to Postfix

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <ctype.h>
#define SIZE 400

char s[SIZE];
int top = -1;
void push(char elem)
{
    s[++top] = elem;
```

```c
}
char pop()
{
    return (s[top--]);
}
int pr(char elem)
{
    switch (elem)
    {
    case '#':
        return 0;
    case '(':
        return 1;
    case '+':
    case '-':
        return 2;
    case '*':
    case '/':
        return 3;
    }
    return 0;
}
int main()
{

    char infx[400], pofx[400], ch;
    int i = 0, k = 0;
    printf("\n\nEnter the Infix Expression: ");
    scanf("%s", infx);
    push('#');
    while ((ch = infx[i++]) != '\0')
    {
        if (ch == '(')
            push(ch);
        else if (isalnum(ch))
            pofx[k++] = ch;
        else if (ch == ')')
        {
            while (s[top] != '(')
                pofx[k++] = pop();
            top--;
        }
        else
        {
            while (pr(s[top]) >= pr(ch))
```

```c
                pofx[k++] = pop();
            push(ch);
        }
    }
    while (s[top] != '#')
        pofx[k++] = pop();
    pofx[k] = '\0';
    printf("Postfix Expression is:%s", pofx);

    return 0;
}
```

Result

Thus, we have developed an C Code to perform intermediate code generation to convert infix expression into prefix and postfix expression.