Name: Atandrit Chatterjee

Reg. No.: RA2011031010042

**Experiment-10 Intermediate Code Generation – Postfix, Prefix**

Aim:

To implement intermediate code generation in C/C++.

Procedure:

1. Declare set of operators.

2. Initialize an empty stack.

3. To convert INFIX to POSTFIX follow the following steps

4. Scan the infix expression from left to right.

5. If the scanned character is an operand, output it.

6. Else, If the precedence of the scanned operator is greater than the precedence of the

operator in the stack(or the stack is empty or the stack contains a '(' ), push it.

7. Else, Pop all the operators from the stack which are greater than or equal to in precedence

than that of the scanned operator. After doing that Push the scanned operator to the stack.

8. If the scanned character is an '(', push it to the stack.

9. If the scanned character is an ')', pop the stack and output it until a '(' is encountered, and

discard both the parenthesis.

10. Pop and output from the stack until it is not empty.

11. To convert INFIX to PREFIX follow the following steps

12. First, reverse the infix expression given in the problem.

13. Scan the expression from left to right.

14. Whenever the operands arrive, print them.

15. If the operator arrives and the stack is found to be empty, then simply push the operator

into the stack.

16. Repeat steps 6 to 9 until the stack is empty

## Code:

## Infix to Postfix:

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#define SIZE 100
char stack[SIZE];
int top = -1;
void push(char item) {
if(top >= SIZE-1)
        printf("\nStack Overflow.");
else {
        top = top+1;
        stack[top] = item;
}
}
char pop() {
char item ;
if(top <0) {
        printf("stack under flow: invalid infix expression");
        getchar();
        exit(1);
```

```
}
else {
        item = stack[top];
        top = top-1;
        return(item);
}
}
int is_operator(char symbol) {
if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol =='-')
        return 1;
else {
return 0;
}
}
int precedence(char symbol) {
if(symbol == '^')
        return(3);
else if(symbol == '*' || symbol == '/')
        return(2);
else if(symbol == '+' || symbol == '-')
        return(1);
else
        return(0);
}

void InfixToPostfix(char infix_exp[], char postfix_exp[]) {
int i, j;
char item;
char x;
push('(');
strcat(infix_exp,")");
```

```
i=0;  j=0;
item=infix_exp[i];
while(item != '\0') {
        if(item == '(')
                push(item);
        else if( isdigit(item) || isalpha(item)) {
                postfix_exp[j] = item;
                j++;
        }
        else if(is_operator(item) == 1) {
                x=pop();
                while(is_operator(x) == 1 && precedence(x)>= precedence(item)) {
                        postfix_exp[j] = x;
                        j++;
                        x = pop();
                }
                push(x);
                push(item);
        }
        else if(item == ')')  {
                x = pop();
                while(x != '(') {
                        postfix_exp[j] = x;
                        j++;
                        x = pop();
                }
        }
        Else {
                printf("\nInvalid infix Expression.\n");
                getchar();
```

```
                exit(1);
        }
        i++;
        item = infix_exp[i];
}
if(top>0) {
        printf("\nInvalid infix Expression.\n");
        getchar();
        exit(1);
}
if(top>0) {
        printf("\nInvalid infix Expression.\n");
        getchar();
        exit(1);
}
postfix_exp[j] = '\0';
}
int main() {
char infix[SIZE], postfix[SIZE];
printf("ASSUMPTION: The infix expression contains single letter variables and single digit constants only.\n");
printf("\nEnter Infix expression : ");
gets(infix);
InfixToPostfix(infix,postfix);
printf("Postfix Expression: ");
puts(postfix);
   printf("\n");
return 0;
}
```

Postfix to Prefix:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
#define MAX 20
char str[MAX], stack[MAX];
int top = -1;
void push(char c) { stack[++top] = c; }
char pop(){  return stack[top--];}
int checkIfOperand(char ch){ return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');}

int isOperator(char x)  {
  switch (x) {
  case '+':
  case '-':
  case '/':
  case '*':
    return 1;
  }
  return 0;
}

void postfixToprfix()  {
  int n, i, j = 0;
  char c[20];
  char a, b, op;
  printf("Enter the postfix expression:\n");
  scanf("%s", str);
  n = strlen(str);
  for (i = 0; i < MAX; i++)
```

```
    stack[i] = '\0';
  printf("Prefix expression is:\t");
  for (i = n - 1; i >= 0; i--) {
    if (isOperator(str[i])) push(str[i]);
    else { c[j++] = str[i];
      while ((top != -1) && (stack[top] == '#'))  {
        a = pop();
        c[j++] = pop();
      }
      push('#');
    }
  }
  c[j] = '\0';
  i = 0;
  j = strlen(c) - 1;
  char d[20];
  while (c[i] != '\0') {
    d[j--] = c[i++];
  }
  printf("%s\n", d);
}
int main() {
  postfixToprfix();
  return 0;
}
```

Output:

```
INPUT THE EXPRESSION: A+B^C/R
PREFIX: +^/CR
POSTFIX: AB^CR/+
```

Result:

The implementation of intermediate code generation was successful.