**Experiment No.2b**

**Title:** Conversion from infix to postfix.

**Problem Statement:**

Write a C Program for conversion of an expression from infix to postfix.

**Algorithm:**

**Step 1.**

Push “)” onto STACK, and add “(“ to end of the A

**Step 2.**

Scan A from right to left and repeat step 3 to 6 for each element of A until the STACK is empty

**Step 3.**

If an operand is encountered add it to B

**Step 4.**

If a right parenthesis is encountered push it onto STACK

**Step 5.**

If an operator is encountered then:

a. Repeatedly pop from STACK and add to B each operator (on the top of STACK) which has same or higher precedence than the operator.

b. Add operator to STACK

**Step 6.**

If left parenthesis is encontered then

a. Repeatedly pop from the STACK and add to B (each operator on top of stack until a left parenthesis is encounterd)

b. Remove the left parenthesis

**Step 7.** Exit

**Code:**

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

#include<string.h>

#include<ctype.h>

#define SIZE 100

void infixTopostfix(char infix[],char postfix[]);

char stack[SIZE];

int top=-1;

int main()

{

char infix[SIZE],postfix[SIZE];

printf("limitation:The infix expression must have single letter variables or single digit constants only");

printf("\nEnter the valid Infix Expression:");

gets(infix);

infixTopostfix(infix,postfix);

printf("/n The equivalent postfix expression:");

puts(postfix);

return 0;

}

void push(char item)

{

if(top>=SIZE-1)

printf("\n Stack is Full!!!!");

else

{

top++;

stack[top]=item;

}

return;

}

char pop(void)

{

char item;

if(top==-1)

{

printf("Stack is Empty!!!!!");

exit(1);

}

else

{ item=stack[top];

top--;

return(item);

}

}

bool isoperator(char symbol)

{

if(symbol=='^'||symbol=='\*'||symbol=='/'||symbol=='+'||symbol=='-')

{

return true;

}

else

{

return false;

}

}

int priority(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[],char postfix[])

{

char token;

char stacktop;

int i=0,j=0;

push('(');

strcat(infix,")");

token=infix[i];

while(infix[i]!='\0')

{

if(token=='(')

{

push(token);

}

if(isdigit(token)||isalpha(token))

{

postfix[j]=token;

j++;

}

else if(isoperator(token))

{

stacktop=pop();

while(isoperator(stacktop)&&priority(stacktop)>=priority(token))

{

postfix[j]=stacktop;

j++;

stacktop=pop();

}

push(stacktop);

push(token);

}

else if(token==')')

{

stacktop=pop();

while(stacktop!='(')

{

postfix[j]=stacktop;

j++;

stacktop=pop();

} }

i++;

token=infix[i];

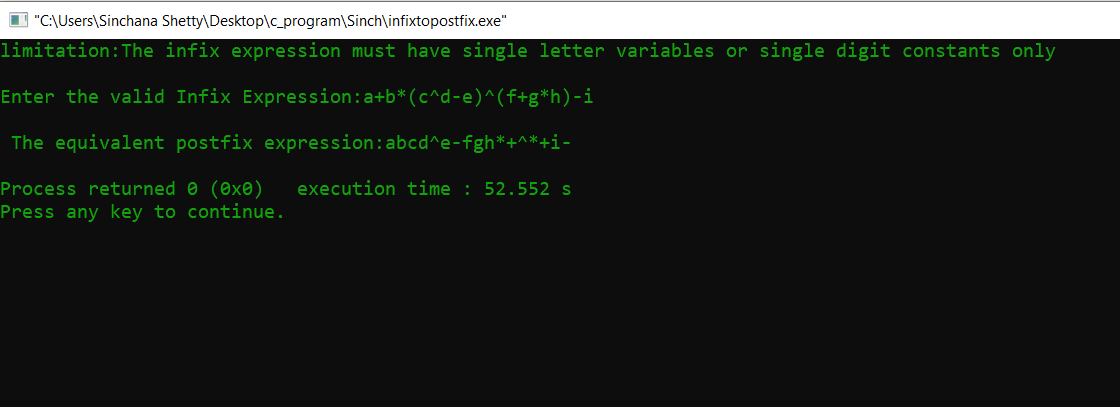
}

postfix[j]='\0';

return;

}

**Results:**



**Analysis(Limitations):**

Usage of stack to convert infix to postfix expression is easier as we have to scan the expression from left to right only once. But it has some limitations and we have to assume the following:

* Let us assume that each line is in the form of a string of digits and operator symbols and that the operands are single nonnegative digits.
* Another assumption is that the expression ends with null character (‘\0’) as we read it as a string.