|  |  |
| --- | --- |
|  | **DEPARTMENT OF COMPUTER ENGINEERING** |

**Experiment No. 02**

|  |  |
| --- | --- |
| Semester | S.E. Semester III – Computer Engineering |
| Subject | Data Structures Lab (CSL301) |
| Subject Professor In-charge | Prof. Swapnil S. Sonawane |
| Assisting Teachers | Prof. Swapnil S. Sonawane |

|  |  |
| --- | --- |
| Student Name | Trisha Shah |
| Roll Number | 20102A0004 |

**Title:**

Convert an Infix expression to Postfix expression using stack ADT.

**Objective:**

Students will be able to choose appropriate data structure and apply it to solve problems in various domains.

**Explanation:**

A Stack is used to convert and evaluate the following 3 types of notations:

1. Infix Notation: In this operator is used in between operands.

Example: A+B

1. Prefix Notation: In this, operator is used before operands.

Example: +AB

1. Postfix Notation: In this, operator is used after operands

Example: AB+

Infix expressions are readable and solvable by humans. We can easily distinguish the order of operators, and also can use the parenthesis to solve that part first during solving mathematical expressions. The computer cannot differentiate the operators and parenthesis easily, that’s why postfix conversion is needed.

To convert infix expression to postfix expression, we will use the stack data structure. By scanning the infix expression from left to right, when we will get any operand, simply add them to the postfix form, and for the operator and parenthesis, add them in the stack maintaining the precedence of them.

**Program Code:**

#include<stdio.h>

#include<string.h>

#define N 20

typedef struct stack

{

char a[N];

int top;

}stack;

void push(stack \*s,char x)

{

s->top++;

s->a[s->top]=x;

}

char pop(stack \*s)

{

char x;

x=s->a[s->top];

s->top--;

return x;

}

int isempty(stack \*s)

{

if(s->top==-1)

return 1;

else

return 0;

}

char stacktop(stack \*s)

{

return s->a[s->top];

}

int isoperand(char x)

{

if((x>='A'&&x<='Z')||(x>='a'&&x<='z'))

return 1;

else

return 0;

}

int isoperator(char x)

{

if(x=='+'||x=='-'||x=='\*'||x=='/')

return 1;

else

return 0;

}

int priority(char x)

{

if(x=='\*'||x=='/')

return 3;

else if(x=='+'||x=='-')

return 2;

else

return 1;

}

void convert(char infix[],char postfix[])

{

stack s;

s.top=-1;

int i,k=0;

char x,ele;

for(i=0;i<strlen(infix);i++)

{

x=infix[i];

if(isoperand(x))

postfix[k++]=x;

else if(x=='(')

push(&s,x);

else if(isoperator(x))

{

while(priority(x)<=priority(stacktop(&s)))

{

ele=pop(&s);

postfix[k++]=ele;

}

push(&s,x);

}

else

{

while(stacktop(&s)!='(')

{

ele=pop(&s);

postfix[k++]=ele;

}

ele=pop(&s);

}

}

while(isempty(&s)==0)

{

ele=pop(&s);

postfix[k++]=ele;

}

postfix[k]='\0';

}

int main()

{

char infix[20],postfix[20];

printf("Enter infix expression=");

gets(infix);

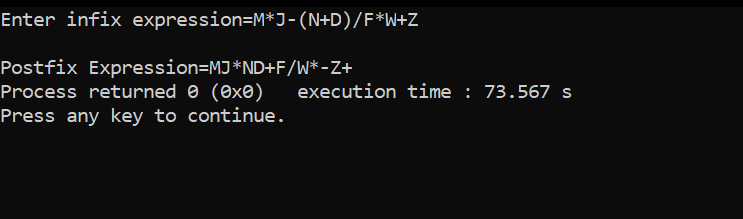
convert(infix,postfix);

printf("\nPostfix Expression=%s",postfix);

return 0;

}

**Output:**

****

**Conclusion:** Through thisexperiment, we learned about different concepts like infix and postfix operations and how to apply them to a program to convert expressions from one form to another.