**Experiment 7**

**Aim**: To convert an infix expression into a postfix expression

**Theory :**

**Infix expression is an expression** of the form a op b. When an operator is in-between every pair of operands. Eg: a+b

**Postfix expression is an** expression of the form a b op. When an operator is followed for every pair of operands. Eg: ab+

One of the major uses of postfix expression is because the compiler scans the expression either from left to right or from right to left.

**Algorithm :**

1. Start
2. Create a structure stack with data members array a and top and declare an object s, and initialise top to -1.
3. Define functions push() and pop()
4. Create function prec() that takes in a character and returns its corresponding precedence value
5. Create function oper() to check whether a character is a operator or not
6. Create function operand() to check whether a character is an operand or not
7. Create function convertion(), that takes in a character array, which converts infix expression to postfix
   1. Create a character array postfix[] to store the postfix expression
   2. Till the infix array ends, push and pop characters into the stack and into the postfix array according to the precedence of operators
   3. Print out the postfix expression
8. Stop.

**Program :**

#include <iostream>

using namespace std;

struct stack

{

int a[100];

int top;

}s;

void push(char x)

{

s.a[++s.top]=x;

cout<<x<<" pushed into stack"<<"\n";

}

char pop()

{

int x=s.a[s.top--];

cout<<"G"<<"\n";

return x;

}

int prec(char x)

{

switch (x)

{

case '$': return (0);

case '(': return (0);

case '+': return (1);

case '-': return (1);

case '\*': return (2);

case '/': return (2);

case '%': return (3);

}

}

int oper(char x)

{

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

return (1);

else

return (0);

}

int operand (char x)

{

if (x>='a'&&x<='z')

return (1);

else

return (0);

}

char convertion(char\* infix)

{

char postfix[50];

s.top=-1;

push('$');

int i=0,j=0,x;

while (infix[i]!=' ')

{

if (operand(infix[i]))

postfix[j++]=infix[i++];

else if (infix[i]=='(')

push(infix[i++]);

else if (oper(infix[i]))

{

if(prec(infix[i])>prec(s.a[s.top]))

{push(infix[i++]);}

else

{

while(prec(s.a[s.top])>=prec(infix[i]))

{ postfix[j++]=pop();}

push(infix[i++]);

}

}

else if (infix[i]==')')

{

while(x=pop()!='(')

postfix[j++]=x;

}

}

do

{

x=pop();

postfix[j++]=x;

}while(x!='$');

cout<<”Given Infix Expression=”;

for (int i=0;infix[i]!=’\0’;i++\_

cout<<infix[i];

cout<<”\n”;

cout<<”Postfix Expression:= “;

for(int i=0;postfix[i]!='\0';i++)

cout<<postfix[i];

cout<<”\n”;

}

int main()

{

char infix[];

cout<<”Enter the Infix Expression = “;

for(int i=0;infix[i]!='\0';i++)

cout<<infix[i];

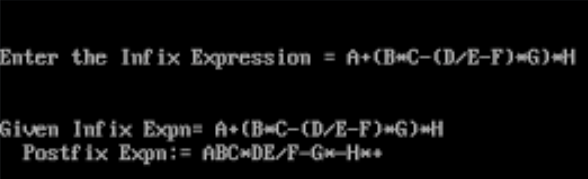
cout<<"\n";

convertion(infix);

return 0;

}

**Output**



**Learning**

Stacks are very useful when an infix expression needs to be converted to postfix expression.