ASSIGNMENT 9

AIM:- Implement stack for expression conversion (infix to postfix)

OBJECTIVE:-1) Understand the Stack Data Structure and its basic operators.

2) Understand the method of defining stack ADT and implement the basic operators. 3) Learn how to create objects from an ADT and invoke member functions.

THEORY:-In high level programming languages, we use arithmetic expression in its infix form. An expression in infix form contains operators in between operands on which it operates. Parentheses also appear in infix expressions to specify the order of evaluation. During compilation, the compiler converts the infix expression to postfix for easy evaluation, since a postfix expression does not contain any parenthesis. Also, a postfix expression can be evaluated easily by using a stack. Postfix notation has the following virtues: No parenthesis. The priority of the operations is no longer relevant. Enables easy evaluation (evaluated by making a left to right scan, stacking the operands.) An infix expression can be manually converted to its post-fix form by following these steps: Step1: Fully parenthesis the expression.

Step2: Move all operators so that they replace their corresponding right parenthesis. Step3: Delete all parentheses.

CODE:-

so #include<iostream>

using namespace std;

char stack[50];

int top = -1;

void push(char ele)

{

stack[++top] = ele;

}

int pop()

{

return (stack[top--]);

}

int pr(char ele)

{

switch(ele)

{

case '#': return 0;break;

case '(': return 1;break;

case '+':

case '-': return 2; break;

case '\*':

case '/': return 3; break;

case '^': return 4; break;

}

}

int main()

{

int k = 0,i = 0,p,j;

char ch,in[50],post[50];

cout<<"\nOperation for Conversion of infix expression to postfix expression\n";

cout<<"==================================================================\n\n";

cout<<"Enter the infix expression : ";

cin>>in;

push('#');

while((ch=in[i++])!='\0')

{

if(ch == '(')

{

push(ch);

}

else if(isalnum(ch))

{

post[k++] = ch;

}

else if(ch == ')')

{

while(stack[top]!='(')

{

post[k++] = pop();

}

pop();

}

else

{

while(pr(stack[top])>=pr(ch))

{

post[k++] = pop();

}

push(ch);

}

}

while(stack[top]!='#')

{

post[k++] = pop();

}

post[k]='\0';

cout<<"Postfix expression is : "<<post<<"\n";

cout<<"\n\n";

return 0;

}

OUTPUT:-

/\*

Operation for Conversion of infix expression to postfix expression

==================================================================

Enter the infix expression : (x+2)\*(3-1)

Postfix expression is : x2+31-\*

--------------------------------

Process exited after 33.48 seconds with return value 0

Press any key to continue . . .

\*/

CONCLUSION:-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 postfix form,and for the operator and parenthesis ,add them in the stack maintaining the precedence of them.