

**NAME: UDIT BHURKE ROLLNO: 03**

**DIV: 03 CLASS: SE**

**Experiment no 3: Evaluation of postfix Expression using stack ADT**

**Aim:** Implementation of Evaluation of Postfix Expression using stack ADT

**Objective:**

1. Understand the use of stack
2. Understand importing an ADT in an application program
3. Understand the instantiation of stack ADT in an application Program
4. Understand how the member function of an ADT are accessed in an application program

**Theory:**

To evaluate a postfix expression we can use a stack. Iterate the expression from left to right and keep on storing the operands into a stack. Once an operator is received, pop the two topmost elements and evaluate them and push the result in the stack again. Scan 2, it's a number, So push it into stack.

Postfix notation (also known as Reverse Polish Notation) is a way to represent an expression, where operators follow their corresponding operands. Evaluating an expression represented as postfix notation can easily be done using the stack data structure.

Postfix notation is one of the few ways to represent algebraic expressions. It is used in computers because it is faster than other types of notations (such as infix notation) as parentheses are not required to represent them.

As the name suggests, in the postfix expression operators follow their operands. Therefore, the process of postfix evaluation is quite different than solving the infix notation (normal notation used in daily use).

**Algorithm:**

1. START
2. item = READ\_symbol()
3. while item != ’#’

if item = operand then push (item)

else

op=item

y=pop()

x=pop()

t=xopy()

PUSH(t)

end if

item = READ\_symbol()

end while

4. value =pop()

5. STOP

**Code :**

#include<stdio.h>

int stack[20];

int top = -1;

void push(int x)

{

stack[++top] = x;

}

int pop()

{

return stack[top--];

}

int main()

{

char exp[20];

char \*e;

int n1,n2,n3,num;

printf("Enter the expression :: ");

scanf("%s",exp);

e = exp;

while(\*e != '\0')

{

if(isdigit(\*e))

{

num = \*e - 48;

push(num);

}

else

{

n1 = pop();

n2 = pop();

switch(\*e)

{

case '+':

{

n3 = n1 + n2;

break;

}

case '-':

{

n3 = n2 - n1;

break;

}

case '\*':

{

n3 = n1 \* n2;

break;

}

case '/':

{

n3 = n2 / n1;

break;

}

}

push(n3);

}

e++;

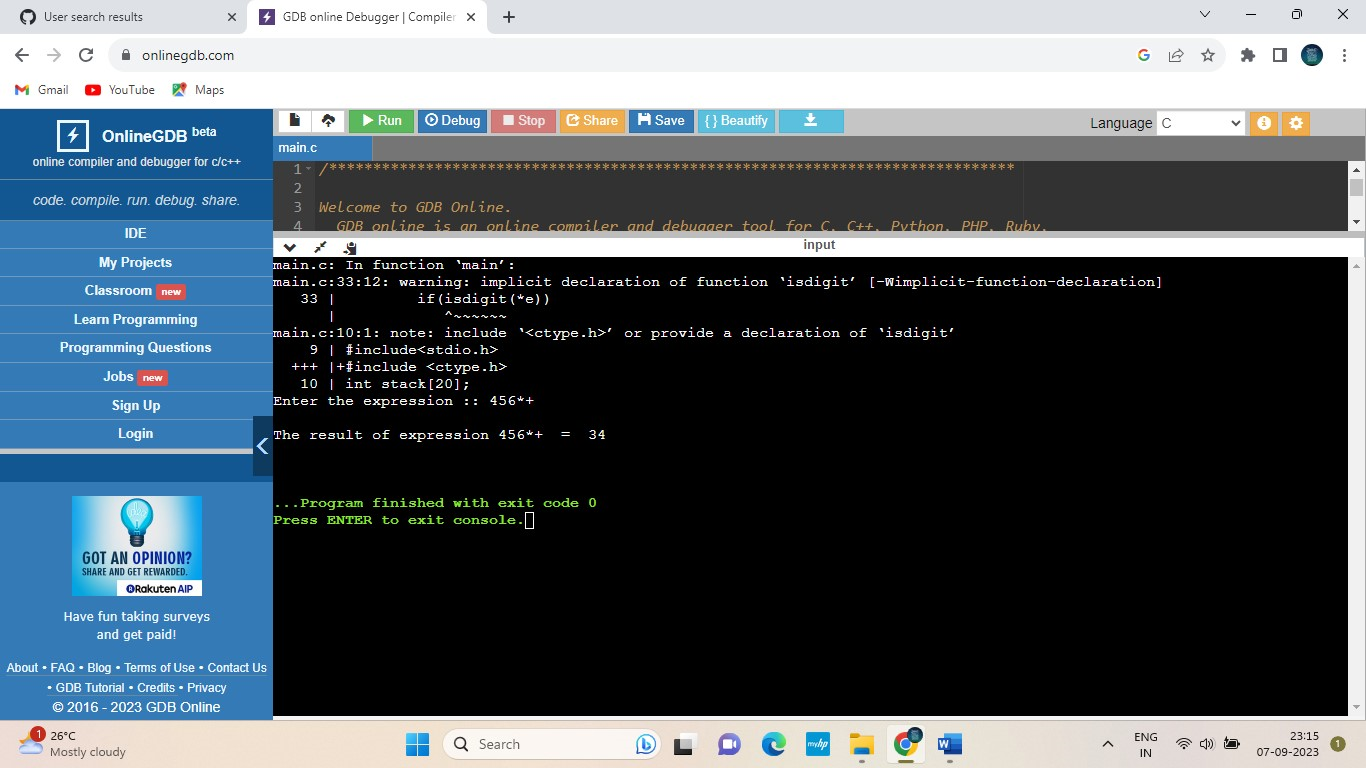
}

printf("\nThe result of expression %s = %d\n\n",exp,pop());

return 0;

}

**Output:**



**Conclusion :**

* In postfix notation, the operator appears after the operands.
* It does not require the use of parentheses and therefore it is faster when compared to other notations.
* Stack data structure is used for evaluation of postfix expression efficiently.