

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:**

While writing an arithmetic expression using infix notation, the operator is placed in between the operands. For example, A+B; here, plus operator is placed between the two operands A and B. Although it is easy for us to write expressions using infix notation, computers find it difficult to parse as the computer needs a lot of information to evaluate the expression. Information is needed about operator precedence and associativity rules, and brackets which override these rules. So, computers work more efficiently with expressions written using prefix and postfix notations.

In postfix notation, as the name suggests, the operator is placed after the operands. For example, if an expression is written as A+B in infix notation, the same expression can be written as AB+ in postfix notation.

Using stacks, any postfix expression can be evaluated very easily. Every character of the postfix expression is scanned from left to right. If the character encountered is an operand, it is pushed on to the stack. However, if an operator is encountered, then the top two values are popped from the stack and the operator is applied on these values. The result is then pushed on to the stack.

**Algorithm:**

1. START
2. item = READ\_symbol()
3. while \*item != ’\0’

if item = operand then push (item)

else

item = operation

y=pop()

x=pop()

t=x operation y

push(t)

end if

item++

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);

}

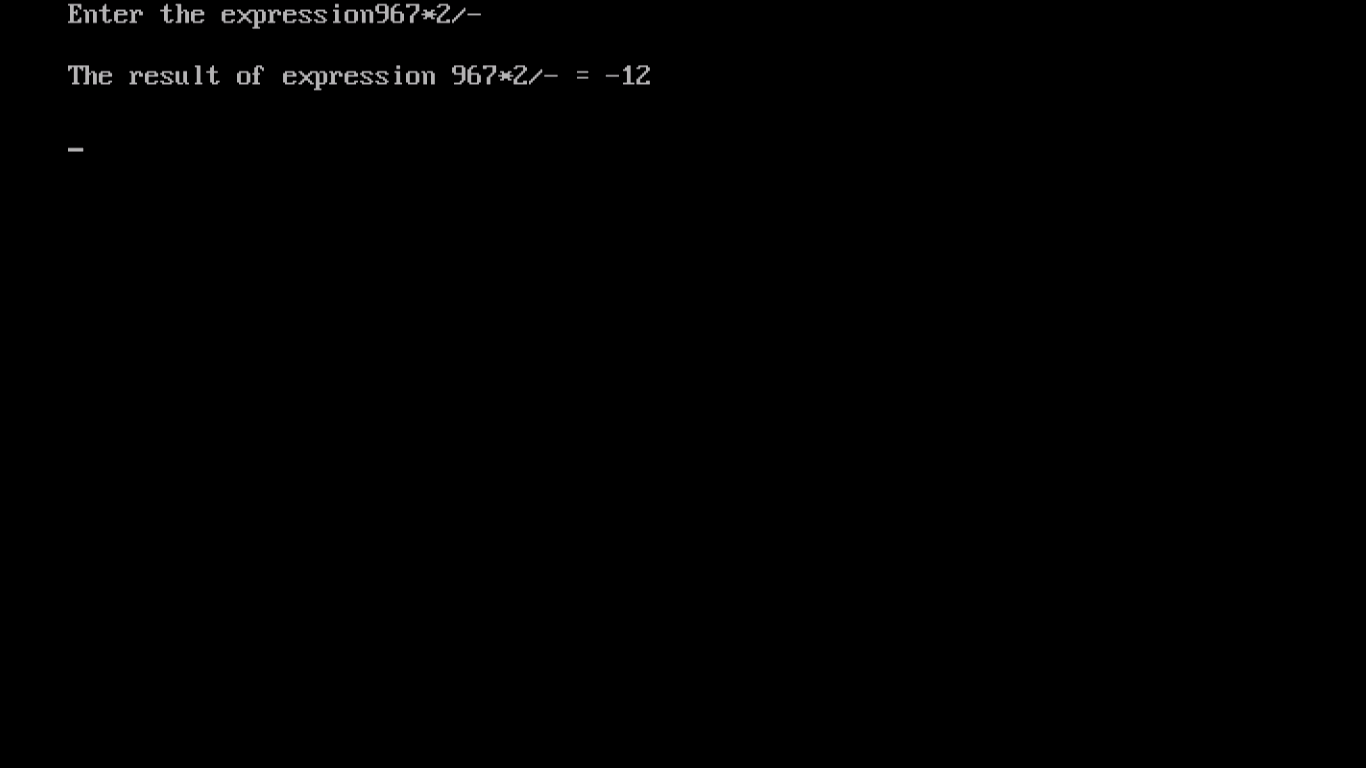
e++;

}

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

return 0;

}



**Conclusion :**

* In postfix notation, the operator appears after the operands.
* Evaluation of postfix notation is faster.
* Stack data structure is used for evaluation of postfix expression efficiently.