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

**NAME:Kumud Mohan Zimal DIV: COMPS-3 ROLL NO: 68 BATCH: C**

**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:**

Step 1 − scan the expression from left to right

Step 2 − if it is an operand push it to stack

Step 3 − if it is an operator pull operand from

stack and perform operation

Step 4 − store the output of step 3, back to stack

Step 5 − scan the expression until all operands

are consumed

Step 6 − pop the stack and perform operation

Postfix Evaluation Algorithm

1) Add ) to postfix expression.

2) Read postfix expression Left to Right until ) encountered

3) If operand is encountered, push it onto Stack

[End If]

4) If operator is encountered, Pop two elements

i) A -> Top element

ii) B-> Next to Top element

iii) Evaluate B operator A

push B operator A onto Stack

5) Set result = pop

6) END

**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++;

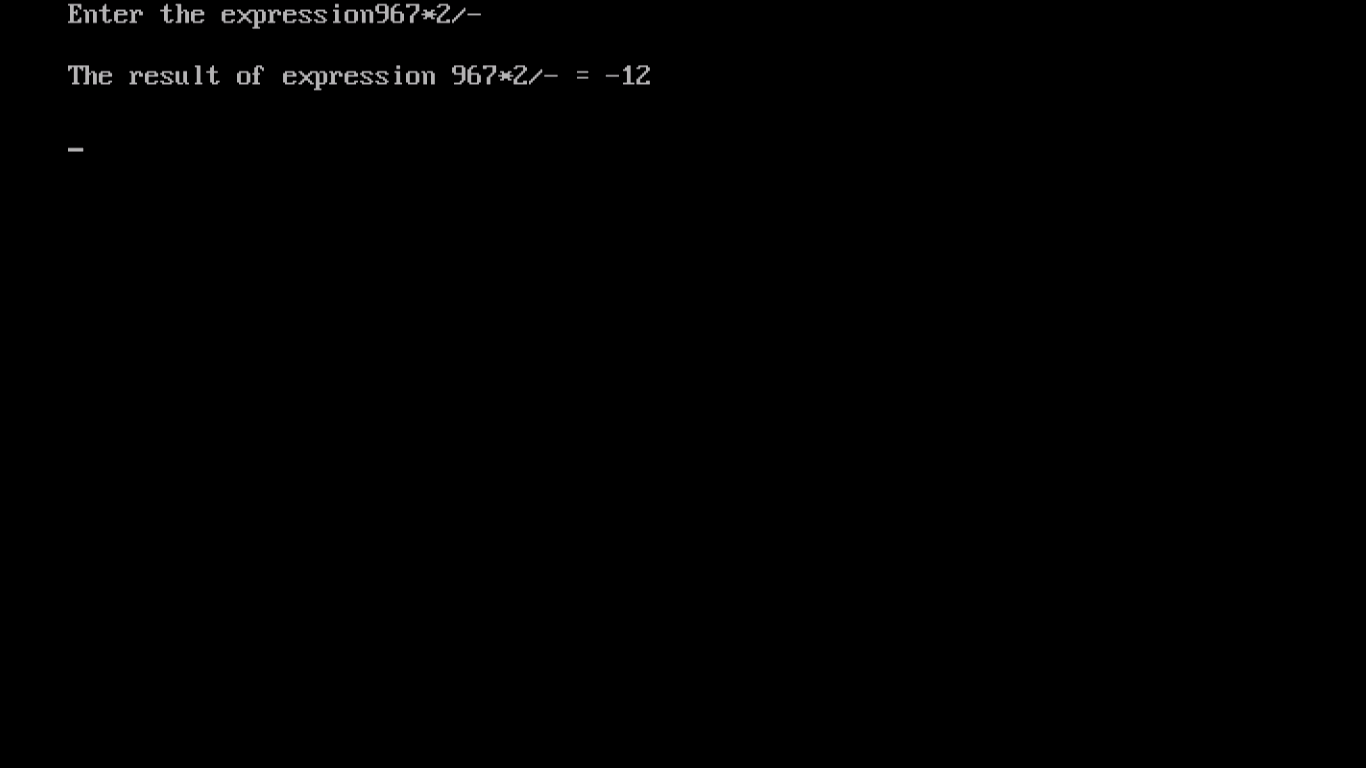
}

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

return 0;

}

**Output :**



**Conclusion :**

In conclusion, this experiment successfully demonstrated the use of a stack ADT to evaluate postfix expressions. We learned how to import and instantiate a stack ADT in a program and how to access its member functions. The implemented code correctly evaluated postfix expressions by scanning them, pushing operands onto the stack, and performing operations when operators were encountered. This experiment provided valuable insights into data structures and their practical applications in solving mathematical problems.