Experiment No.3
Evaluate Postfix Expression using Stack ADT.
Name: Sharvari Anand Bhondekar
Roll No: 06
Date of Performance:
Date of Submission:
Marks:
Sign:

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 functions of an ADT are accessed in an application program

Theory:

An arithmetic expression consists of operands and operators. For a given expression in a postfix form, stack can be used to evaluate the expression. The rule is whenever an operand comes into the string, push it onto the stack and when an operator is found then the last two elements from the stack are popped and computed and the result is pushed back onto the stack. One by one the whole string of postfix expressions is parsed and the final result is obtained at an end of computation that remains in the stack.

Algorithm

Step 1: Add a ")" at the end of the postfix expression
Step 2: Scan every character of the postfix expression and repeat Steps 3 and 4 until ")"is encountered
Step 3: IF an operand is encountered, push it on the stack
IF an operator 0 is encountered, then
    a. Pop the top two elements from the stack as A and B as A and B
    b. Evaluate BOA, where A is the topmost element and B is the element below A.
    c. Push the result of evaluation on the stack [END OF IF]
Step 4: SET RESULT equal to the topmost element of the stack
Step 5: EXIT

Code:

```
#include <stdio.h>

#include <ctype.h>

#define Max 100
```

```
int Top = -1;
int a[Max];

void push(int n)
{
if(Top == Max - 1)
{
printf("Stack is Full\n");
}
else
{
Top++;
a[Top] = n;
}
}

int pop()
{
int num;
if(Top == -1)
{
printf("Stack is Empty\n");
return -1;
}
else
{
```

```
num =a[Top];
Top--;
return num;
}
}

void EvalPostfix(char Postfix[])
{
char ch;
int A,B,value,i;
for(i=0;Postfix[i] != ')';i++)
{
ch = Postfix[i];
if(isdigit(ch))
{
push(ch-'0');
}
else if (ch == '*' || ch == '/' || ch == '+' || ch ==
'-' )
{
B = pop();
A = pop();
switch (ch)
{
case '*':
{
```

```
value = A * B;
break;
}
case '/':
{
value = A / B;
break;
}
case '+':
{
    value = A + B;
break;
}
case '-':
{
value = A - B;
break;
}
}
push(value);
}
}
printf("%d",pop(value));
}
int main()
{
```

```
int i;
char Postfix[100];
printf("Enter Your Expresion:\n");
printf("[Note : Put ')' in the End of Expresion]\n");
for(i=0; i<=99 ; i++)
{
scanf("%c",&Postfix[i]);
if(Postfix[i] == ')')
{
break;
}
}

EvalPostfix(Postfix);
return 0;
}
```

Output:

```
Enter Your Expresion:
[Note : Put ')' in the End of Expresion]
57+62-*)
48
```

Conclusion:

1. Elaborate the evaluation of the following postfix expression in your program.

● AB+CWill

this input be accepted by your program. If so, what is the output?

→ Supposing AB+C- with A = 9, B = 5, and C = 6:

1. The program will map A to 9, B to 5, and C to 6.
2. Push 9 (value of A) onto the stack.
3. Push 5 (value of B) onto the stack.
4. Perform 9 + 5 and push the result onto the stack.
5. Push 2 onto the stack.
6. Perform and push the result onto the stack.

The final result is 8, which will be printed by the program.

Input and Output:

● Input: AB+C-)

● Output: 8