Experiment No.3
Aim - Evaluate Postfix Expression using Stack ADT.
Name: Shravani Sandeep Raut
Roll No: 48
Date of Performance:
Date of Submission:

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
```

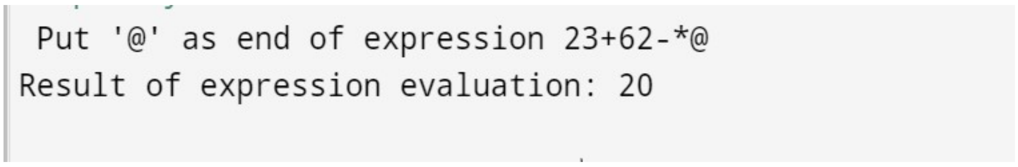
Code:

```
#include <stdio.h>
#include <ctype.h>
#define MAX 100
int a[MAX];
int top = -1;

void push(int n)
{
        if (top == MAX - 1)
        {
                printf("Stack is full");
        }
```

```
	else
	{
		top++;
		a[top] = n;
	}
}

int pop()
{
	int num;
	if (top == -1)
	{
		printf("Stack is empty");
	}
	else
	{
		num = a[top];
		top--;
		return num;
	}
}


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

```
                push(val);
            }
        }
        printf("Result of expression evaluation: %d", pop());
}


void main()
{
        int i;
        char postfix[100];
        printf(" Put '@' as end of expression ");
        for (i = 0; i <= 99; i++)
        {
                scanf("%c", &postfix[i]);
                if (postfix[i] == '@')
                {
                        break;
                }
        }
        EvalPostfix(postfix);
}
```

Output:



```
 Put '@' as end of expression 23+62-*@
Result of expression evaluation: 20
```

Conclusion:

1) Elaborate the evaluation of the following postfix expression in your program. AB+C-

To evaluate the postfix expression AB+C- step-by-step using a program, we'll assume that `A`, `B`, and `C` represent some values (e.g., integers). Here's how we can approach it:

`Evaluation Rules:`

1. Postfix expression is evaluated using a stack.
2. Operands (e.g., `A`, `B`, `C`) are pushed onto the stack.
3. Operators (e.g., `+`, `-`) pop operands from the stack, apply the operation, and push the result back onto the stack.
4. The result is the final value left in the stack after all operations are done.

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

Yes, the input **AB+C-** will be accepted by the program.

- The program will accept the input **AB+C-**.
- It uses a stack-based approach to evaluate the postfix expression.
- The dictionary `values` provides the values for the operands (`A`, `B`, `C`), and the program processes each operator (`+`, `-`) by popping the operands from the stack and pushing the result back after performing the operation.