



Name : Anujkumar Yadav

SE 3 C

Roll No. : 64

Experiment No 3: Evaluation of postfix Expression using stack ADT

Aim: Implementation of Evaluation of Postfix Expression using stack ADT

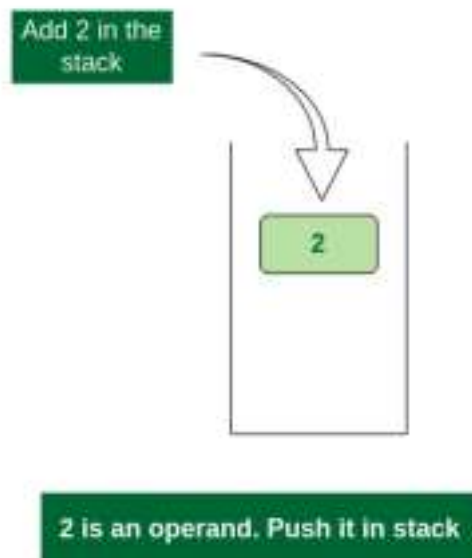
Objective:

- 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:

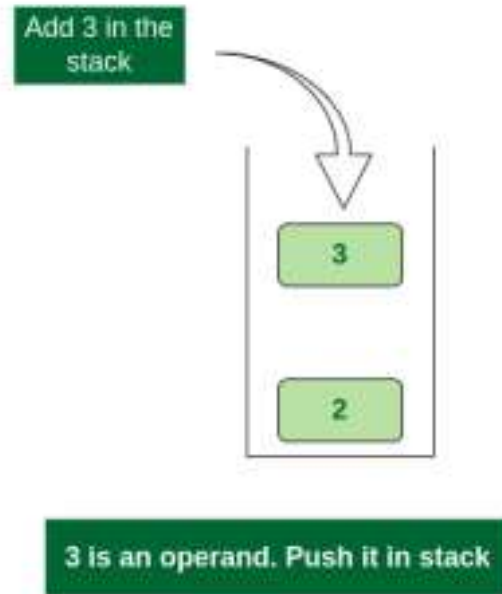
Consider the expression: $\text{exp} = "2\ 3\ 1\ * + 9\ -"$

- Scan **2**, it's a number, So push it into stack. Stack contains '2'.



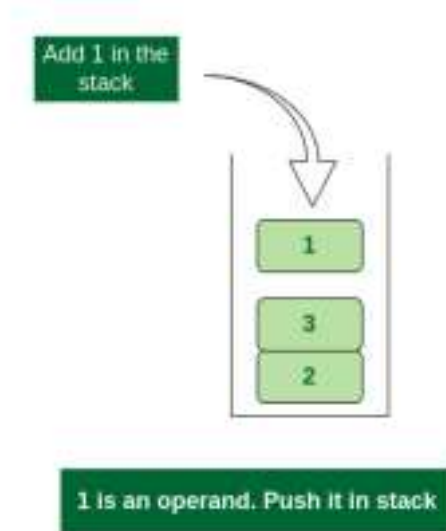
Push 2 into stack

- Scan 3, again a number, push it to stack, stack now contains '2 3' (from bottom to top)



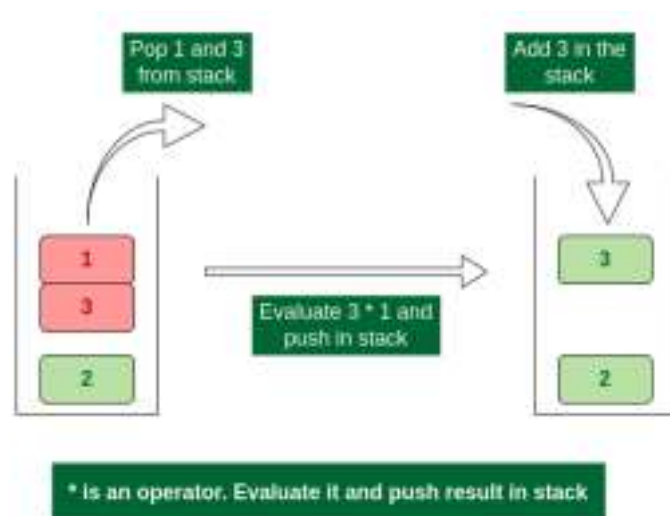
Push 3 into stack

- Scan 1, again a number, push it to stack, stack now contains '2 3 1'



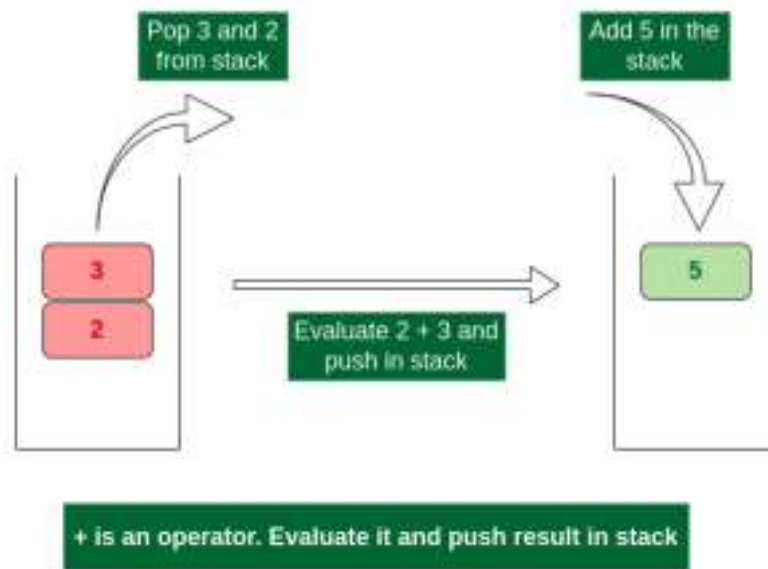
Push 1 into stack

- Scan *, it's an operator. Pop two operands from stack, apply the * operator on operands. We get $3 * 1$ which results in 3. We push the result 3 to stack. The stack now becomes '2 3'.



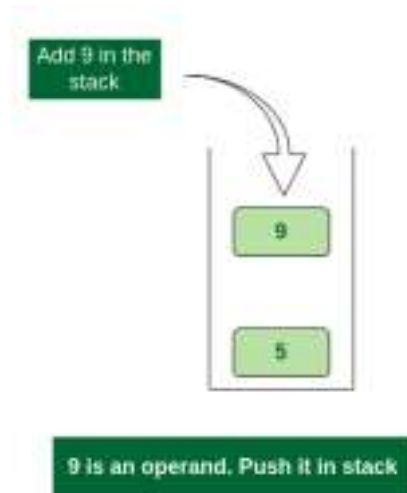
*Evaluate * operator and push result in stack*

- Scan +, it's an operator. Pop two operands from stack, apply the + operator on operands. We get $3 + 2$ which results in 5. We push the result 5 to stack. The stack now becomes '5'.



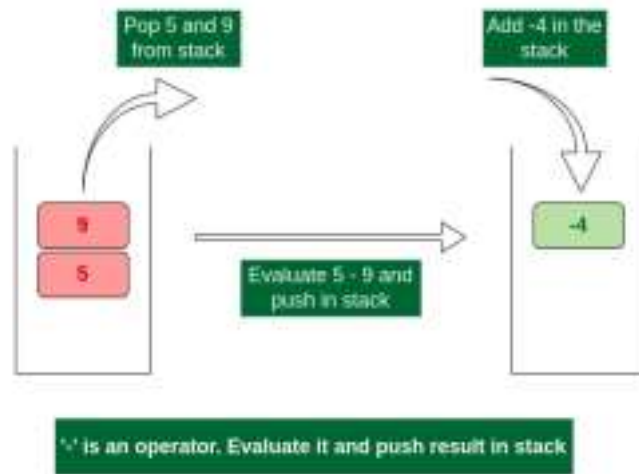
Evaluate + operator and push result in stack

- Scan 9, it's a number. So we push it to the stack. The stack now becomes '5 9'.



Push 9 into stack

- Scan -, it's an operator, pop two operands from stack, apply the - operator on operands, we get $5 - 9$ which results in -4. We push the result -4 to the stack. The stack now becomes '-4'.



Evaluate '-' operator and push result in stack

- There are no more elements to scan, we return the top element from the stack (which is the only element left in a stack).

So the result becomes **-4**.

Algorithm:

Step 1: If a character is an operand push it to Stack

Step 2: If the character is an operator

Pop two elements from the Stack.

Operate on these elements according to the operator, and push the result back to the Stack

Step 3: Step 1 and 2 will be repeated until the end has reached.

Step 4: The Result is stored at the top of the Stack, return it

Step 5: 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;
    clrscr ();
    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 '/':
                {

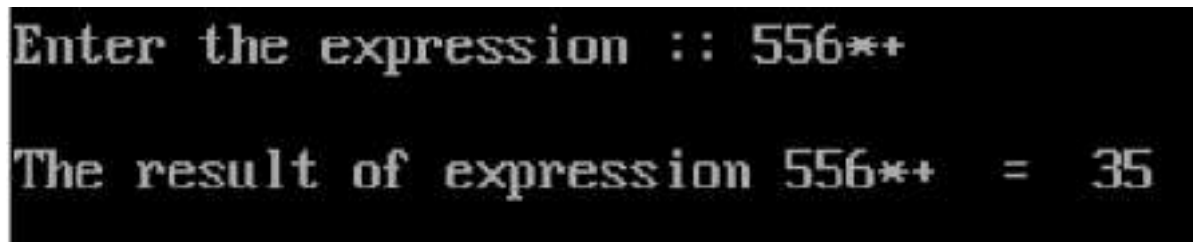
```

```

        n3 = n2 / n1;
        break;
    }
    }
    push(n3);
}
e++;
}
printf("\nThe result of expression %s = %d\n\n",exp,pop());
return 0;
getch() ;
}

```

Output:



```

Enter the expression :: 556*+
The result of expression 556*+ = 35

```

Conclusion :

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.