**SNS COLLEGE OF TECHNOLOGY**
**(An Autonomous Institution)**
**COIMBATORE- 641 035**

Department of Computer Science and Engineering

**23ITT101 - PROGRAMMING IN C AND DATA STRUCTURES**

**UNIT IV STACK AND QUEUE**

# Evaluation of Postfix Expression

***Postfix expression:*** *The expression of the form "a b operator" (ab+) i.e., when a pair of operands is followed by an operator.*

**Examples:**

***Input****: str = "2 3 1 * + 9 -"*
***Output****: -4*
***Explanation:*** *If the expression is converted into an infix expression, it will be 2 + (3 * 1) – 9 = 5 – 9 = -4.*

***Input****: str = "100 200 + 2 / 5 * 7 +"*
***Output****: 757*

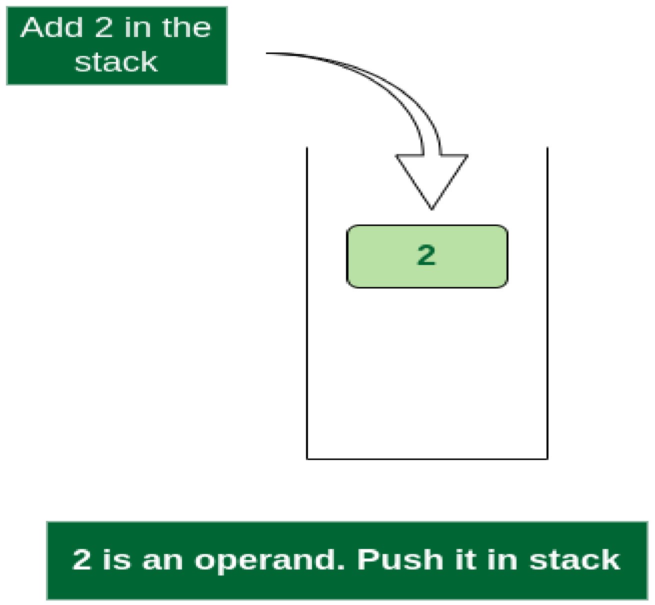
## Evaluation of Postfix Expression using Stack:

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.*

**Illustration:**
Follow the below illustration for a better understanding:

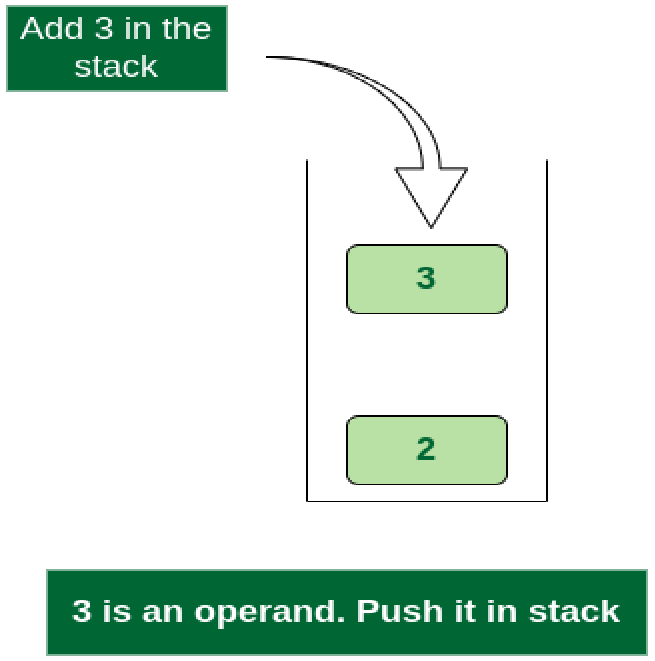
*Consider the expression: exp =* ***"2 3 1 * + 9 -"***

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



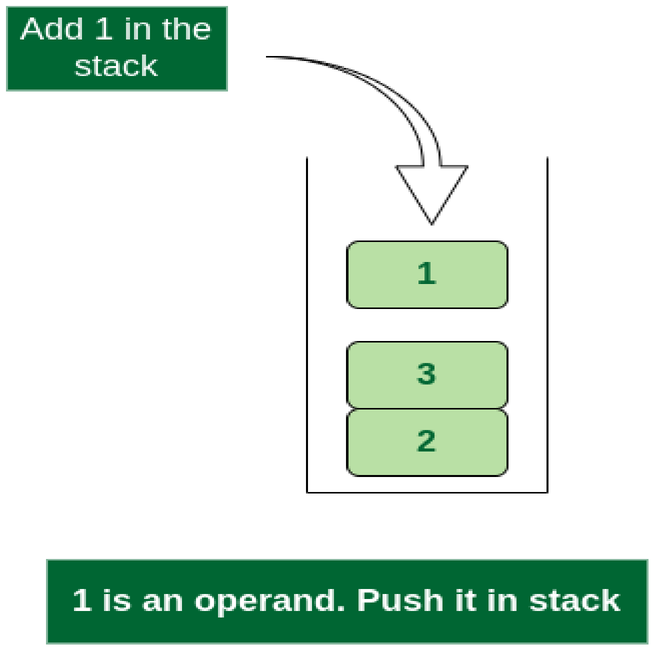
*Push 2 into stack*

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



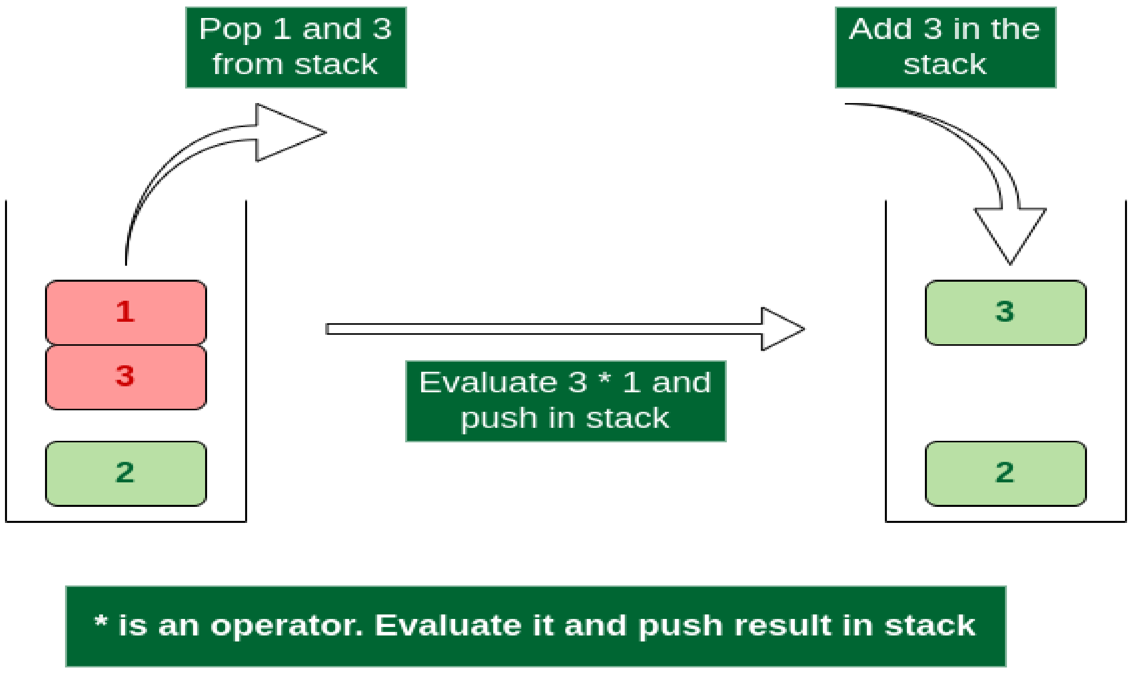
*Push 3 into stack*

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



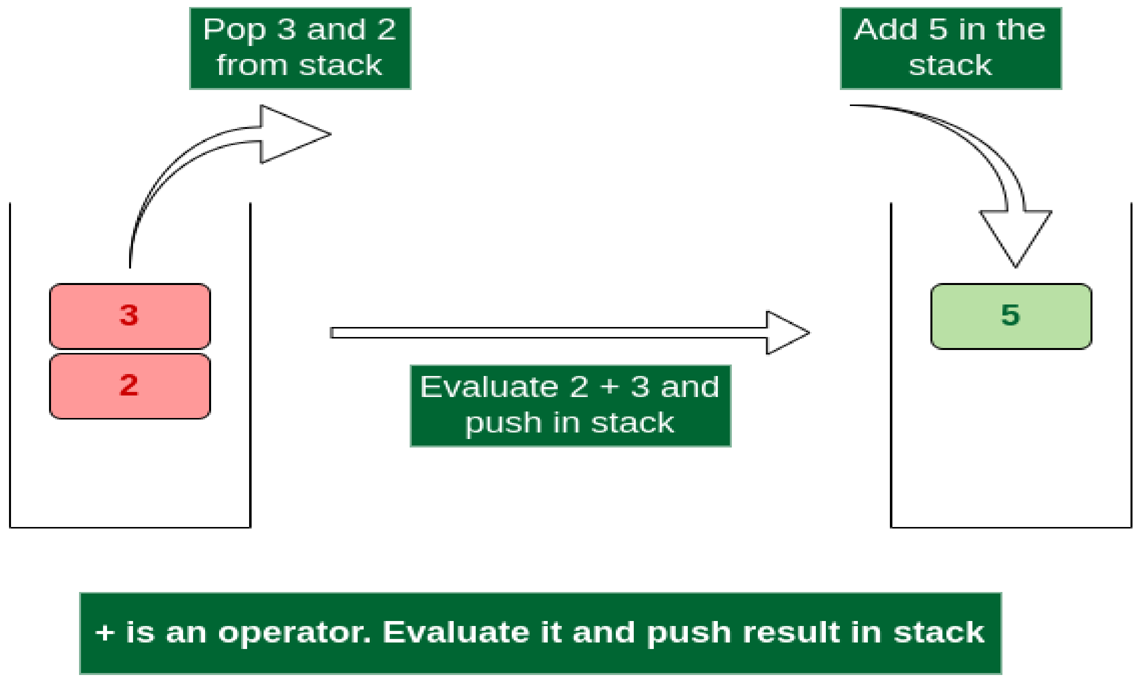
*Push 1 into stack*

- *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'.*


Pop 1 and 3 from stack
Add 3 in the stack
1
3
2
Evaluate 3 * 1 and push in stack
3
2
* is an operator. Evaluate it and push result in stack

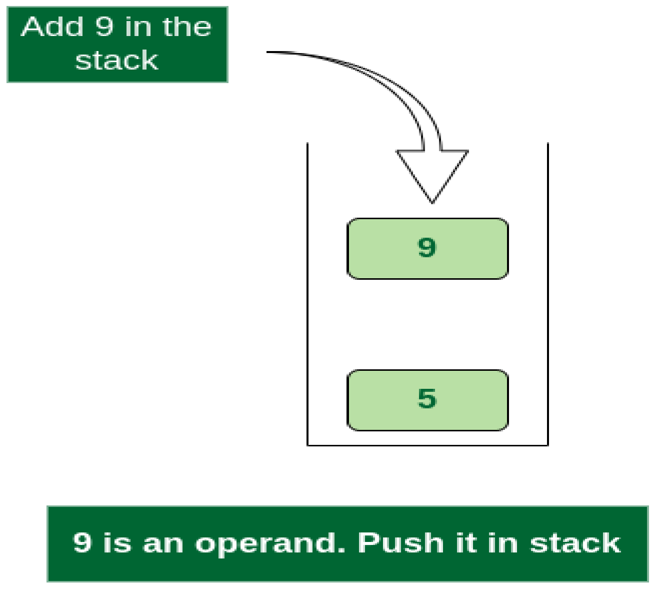
*Evaluate * operator and push result in stack*

- *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'.*


Pop 3 and 2 from stack
Add 5 in the stack
3
2
5
Evaluate 2 + 3 and push in stack
+ is an operator. Evaluate it and push result in stack

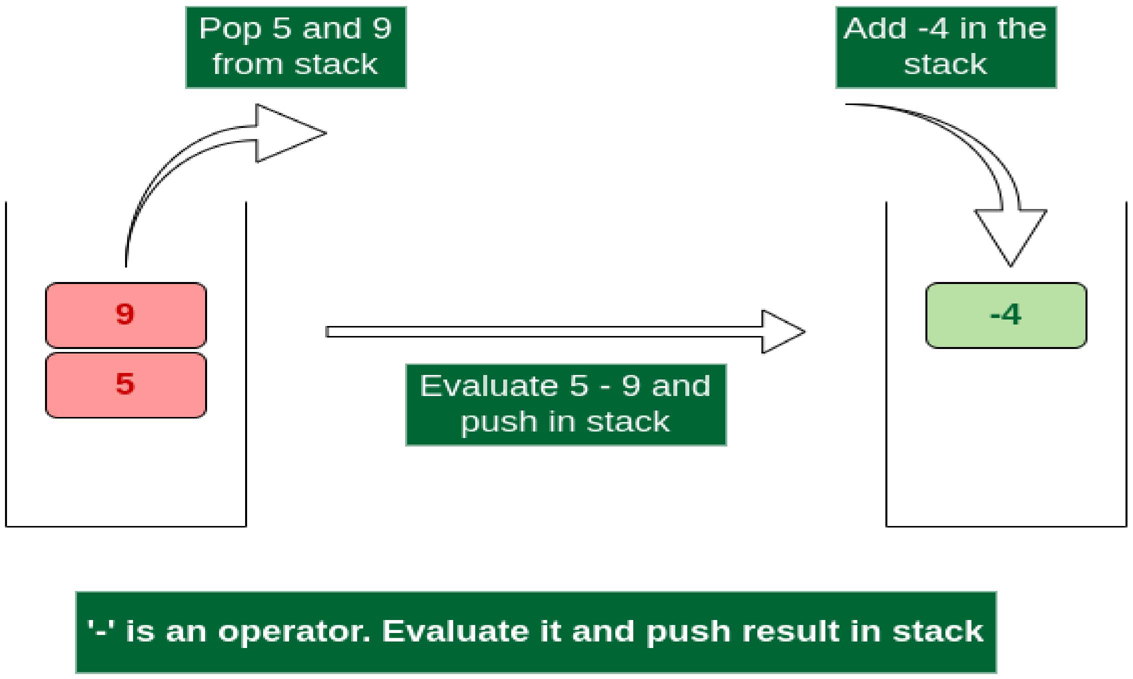
*Evaluate + operator and push result in stack*

- *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'.*


Pop 5 and 9 from stack
Add -4 in the stack
9
5
-4
Evaluate 5 - 9 and push in stack
'-' is an operator. Evaluate it and push result in stack

*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**.*

Follow the steps mentioned below to evaluate postfix expression using stack:

- Create a stack to store operands (or values).
- Scan the given expression from left to right and do the following for every scanned element.
    - If the element is a number, push it into the stack.
    - If the element is an operator, pop operands for the operator from the stack. Evaluate the operator and push the result back to the stack.
- When the expression is ended, the number in the stack is the final answer.

**Example 1:**

**Input**: S = "231*+9-"

**Output**: -4

**Example 2:**

**Input**: S = "123+*8-"

**Output**: -3

**Example 3:**

**Input**: S = 53+62/35+

**Output**: 68

**Example 4:**

**Input**: S = 10 5 60 6 / * 8 -
**Output**: 142

**Example 5:**

**Input**: S = 5 4 6 * 4 9 3 *
**Output**: 350

Below is the implementation of the above approach:

```
// C program to evaluate value of a postfix expression
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Stack type
struct Stack {
    int top;
    unsigned capacity;
    int* array;
};

// Stack Operations
struct Stack* createStack(unsigned capacity)
{
    struct Stack* stack
        = (struct Stack*)malloc(sizeof(struct Stack));

    if (!stack)
```

```
        return NULL;

    stack->top = -1;
    stack->capacity = capacity;
    stack->array
        = (int*)malloc(stack->capacity * sizeof(int));

    if (!stack->array)
        return NULL;

    return stack;
}

int isEmpty(struct Stack* stack)
{
    return stack->top == -1;
}

char peek(struct Stack* stack)
{
    return stack->array[stack->top];
}

char pop(struct Stack* stack)
{
    if (!isEmpty(stack))
        return stack->array[stack->top--];
    return '$';
}

void push(struct Stack* stack, char op)
{
    stack->array[++stack->top] = op;
}

// The main function that returns value
// of a given postfix expression
int evaluatePostfix(char* exp)
{
    // Create a stack of capacity equal to expression size
    struct Stack* stack = createStack(strlen(exp));
    int i;

    // See if stack was created successfully
    if (!stack)
```

```
        return -1;

    // Scan all characters one by one
    for (i = 0; exp[i]; ++i) {

        // If the scanned character is an operand
        // (number here), push it to the stack.
        if (isdigit(exp[i]))
            push(stack, exp[i] - '0');

        // If the scanned character is an operator,
        // pop two elements from stack apply the operator
        else {
            int val1 = pop(stack);
            int val2 = pop(stack);
            switch (exp[i]) {
            case '+':
                push(stack, val2 + val1);
                break;
            case '-':
                push(stack, val2 - val1);
                break;
            case '*':
                push(stack, val2 * val1);
                break;
            case '/':
                push(stack, val2 / val1);
                break;
            }
        }
    }
    return pop(stack);
}

// Driver code
int main()
{
    char exp[] = "231*+9-";

    // Function call
    printf("postfix evaluation: %d", evaluatePostfix(exp));
    return 0;
}
```

**Output**

```
postfix evaluation: -4
```

**Time Complexity:** O(N)
**Auxiliary Space:** O(N)

**There are the following limitations of the above implementation.**

- It supports only **4** binary operators **'+', '*', '-'** and **'/'**. It can be extended for more operators by adding more switch cases.
- The allowed operands are only single-digit operands.

## Postfix evaluation for multi-digit numbers:

*The above program can be extended for multiple digits by adding a separator-like space between all elements* ***(operators and operands)*** *of the given expression.*

Below given is the extended program which allows operands to have multiple digits.

```
// C program to evaluate value of a postfix
// expression having multiple digit operands
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Stack type
struct Stack {
    int top;
    unsigned capacity;
    int* array;
};

// Stack Operations
struct Stack* createStack(unsigned capacity)
{
    struct Stack* stack
        = (struct Stack*)malloc(sizeof(struct Stack));
    if (!stack)
        return NULL;

    stack->top = -1;
    stack->capacity = capacity;
    stack->array
        = (int*)malloc(stack->capacity * sizeof(int));

    if (!stack->array)
        return NULL;

    return stack;
}

int isEmpty(struct Stack* stack)
{
    return stack->top == -1;
}

int peek(struct Stack* stack)
{
    return stack->array[stack->top];
}

int pop(struct Stack* stack)
{
    if (!isEmpty(stack))
        return stack->array[stack->top--];
    return '$';
}

void push(struct Stack* stack, int op)
{
    stack->array[++stack->top] = op;
}

// The main function that returns value
// of a given postfix expression
int evaluatePostfix(char* exp)
{
    // Create a stack of capacity equal to expression size
    struct Stack* stack = createStack(strlen(exp));
    int i;
```

```
// See if stack was created successfully
if (!stack)
    return -1;

// Scan all characters one by one
for (i = 0; exp[i]; ++i) {
    // if the character is blank space then continue
    if (exp[i] == ' ')
        continue;

    // If the scanned character is an
    // operand (number here),extract the full number
    // Push it to the stack.
    else if (isdigit(exp[i])) {
        int num = 0;

        // extract full number
        while (isdigit(exp[i])) {
            num = num * 10 + (int)(exp[i] - '0');
            i++;
        }
        i--;

        // push the element in the stack
        push(stack, num);
    }

    // If the scanned character is an operator, pop two
    // elements from stack apply the operator
    else {
        int val1 = pop(stack);
        int val2 = pop(stack);

        switch (exp[i]) {
        case '+':
            push(stack, val2 + val1);
            break;
        case '-':
            push(stack, val2 - val1);
            break;
        case '*':
            push(stack, val2 * val1);
            break;
        case '/':
            push(stack, val2 / val1);
```

```
                break;
            }
        }
    }
    return pop(stack);
}

// Driver program to test above functions
int main()
{
    char exp[] = "100 200 + 2 / 5 * 7 +";

    // Function call
    printf("%d", evaluatePostfix(exp));
    return 0;
}

// This code is contributed by Arnab Kundu
```

**Output**

```
757
```

**Time Complexity:** O(N)
**Auxiliary Space:** O(N)