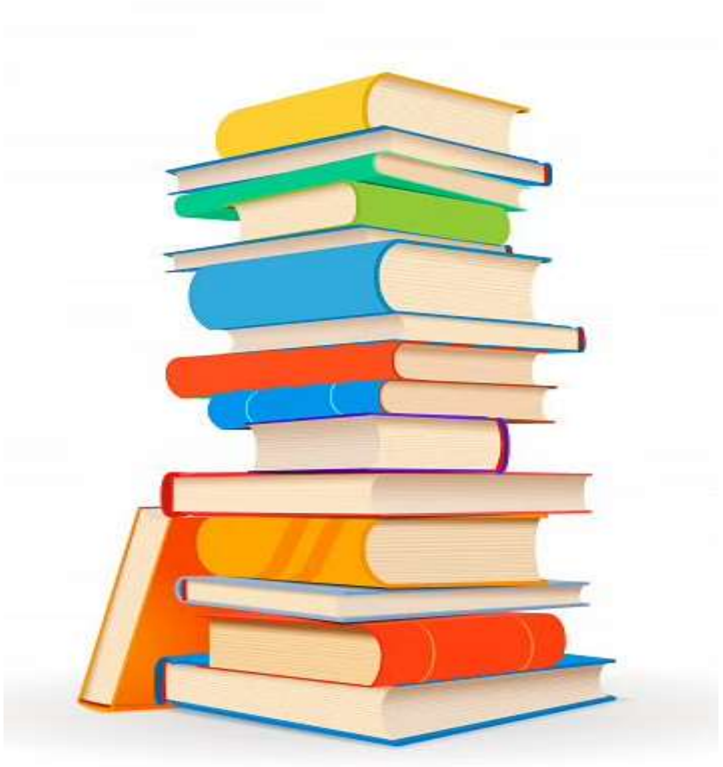
STACK

Stacks in Data Structures is a linear type of data structure that follows the LIFO (Last-In-First-Out) principle and allows insertion and deletion operations from one end of the stack data structure, that is top.

Implementation of the stack can be done by contiguous memory which is an array, and non-contiguous memory which is a linked list. Stack plays a vital role in many applications.

The stack data structure is a linear data structure that accompanies a principle known as LIFO (Last In First Out) or FILO (First In Last Out).

Real-life examples of a stack are a deck of cards, piles of books, piles of money, and many more.



This example allows you to perform operations from one end only, like when you insert and remove new books from the top of the stack.

It means insertion and deletion in the stack data structure can be done only from the top of the stack. You can access only the top of the stack at any given point in time.

Inserting a new element in the stack is termed a push operation.

Removing or deleting elements from the stack is termed pop operation.

Following operations can be performed in stack:

push: Inserts a new element at the top of the stack, above its current top element.

pop: Removes the top element on the stack, thereby decrementing its size by one.

isEmpty: Returns true if the stack is empty, i.e., its size is zero; otherwise, it returns false.

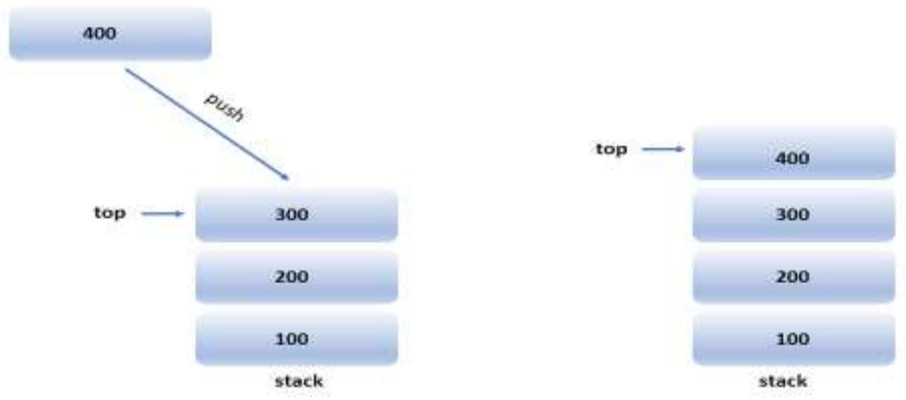
isFull: Returns true if the stack is full, i.e., its size has reached maximum allocated capacity; otherwise, it returns false.

peek: Returns the top element present in the stack without modifying the stack.

size: Returns the count of elements present in the stack.

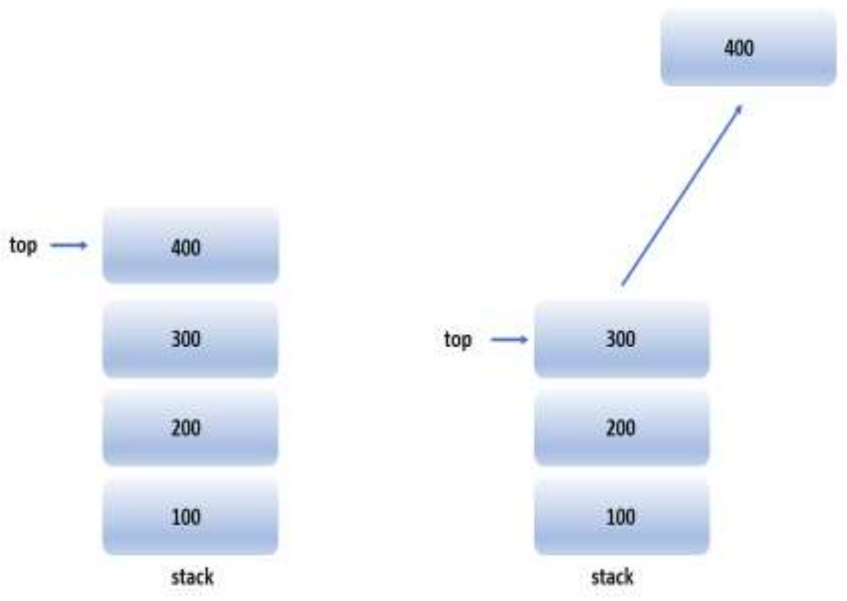
Push Operation

Push operation involves inserting new elements in the stack. Since you have only one end to insert a unique element on top of the stack, it inserts the new element at the top of the stack.



Pop Operation

Pop operation refers to removing the element from the stack again since you have only one end to do all top of the stack. So removing an element from the stack again since you have only one end to do all top of the stack. So removing an element from the top of the stack is termed pop operation.



isFull()

isFull function is used to check whether or not a stack is empty.

The implementation of the isFull() function is as follows:

```
Bool isFull()
{
 if(top == maxsize)
 return true;
else
 return false;
}
```

isEmpty()

isEmpty function is used to check whether or not a stack is empty.
The implementation of the isEmpty() function is:

```
Bool isEmpty()
{
 if(top = = -1)
 return true;
else
 return false;
}
```

Push Operation

Push operation includes various steps, which are as follows:
Step 1: First, check whether or not the stack is full
Step 2: If the stack is complete, then exit
Step 3: If not, increment the top by one
Step 4: Insert a new element where the top is pointing
Step 5: Success

```
if(! isFull ())
{
top = top + 1;
stack[top] = item;
}
else
{
 cout<<“stack is full”;
}
```

Pop Operation

Step 1: First, check whether or not the stack is empty
Step 2: If the stack is empty, then exit
Step 3: If not, access the topmost data element
Step 4: Decrement the top by one
Step 5: Success

```
void pop()

    {

        if (top < 0) {

            cout << "stack underflow";

            return;

        }

        cout << "deleted " << stk[top--];

    }
```

```
void display()
```

```
   {
       if (top < 0)
{
           cout << " stack empty"; return;
 }
for (int i = top; i >= 0; i--)
           cout << stk[i] << " ";
   }}
```

2. **Stack using Linked List**:

We can represent a stack as a linked list. In a stack push and pop operations are performed at one end called top. We can perform similar operations at one end of list using top pointer. The linked stack looks as shown in figure.

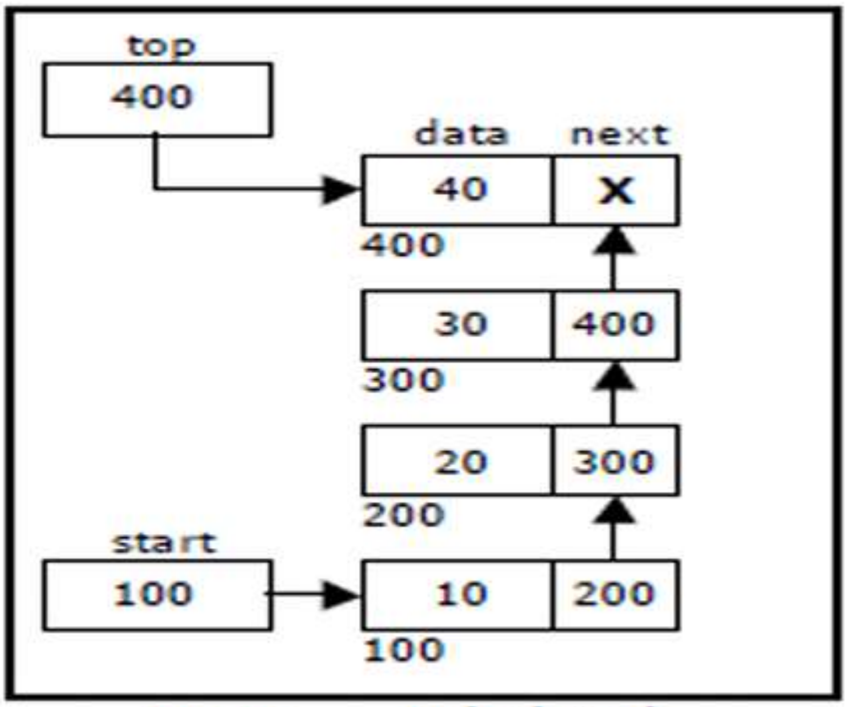


Figure Linked stack representation

Converting and evaluating Algebraic expressions:

An algebraic expression is a legal combination of operators and operands. Operand is the quantity on which a mathematical operation is performed. Operand may be a variable like x, y, z or a constant like 5, 4, 6 etc. Operator is a symbol which signifies a mathematical or logical operation between the operands. Examples of familiar operators include +, -, *, /, ^ etc.

An algebraic expression can be represented using three different notations. They are **infix, postfix and prefix notations**:

Infix: It is the form of an arithmetic expression in which operator in between the two operands. Example: A + B

Prefix: It is the form of an arithmetic notation in which we fix (place) the arithmetic operator before (pre) its two operands. The prefix notation is called as **polish notation**. Example: + A B

Postfix: It is the form of an arithmetic expression in which we fix (place) the arithmetic operator after (post) its two operands. The postfix notation is called as also referred to **reverse polish notation** Example: A B +

Conversion from infix to postfix:

Procedure to convert from infix expression to postfix expression is as follows:

1. Scan the infix expression from left to right.
2. a) If the scanned symbol is left parenthesis, push it onto the and is stack.

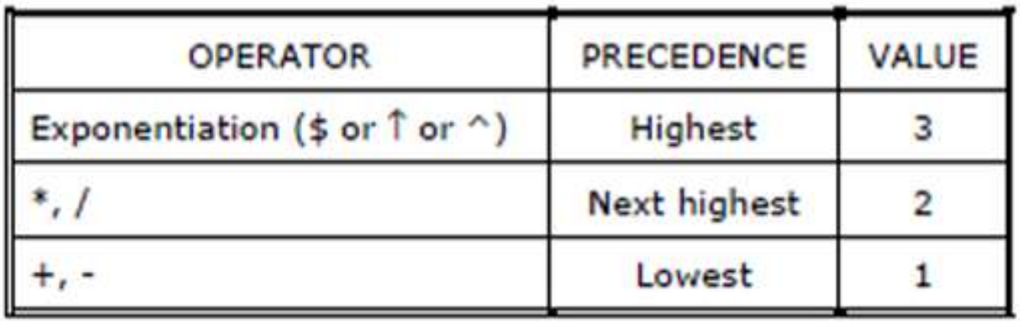
   b) If the scanned symbol is an operand, then place directly in the postfix expression (output).

   c) If the symbol scanned is a right parenthesis, then go on popping all the items from the stack and place them in the postfix expression till we get the matching left parenthesis.

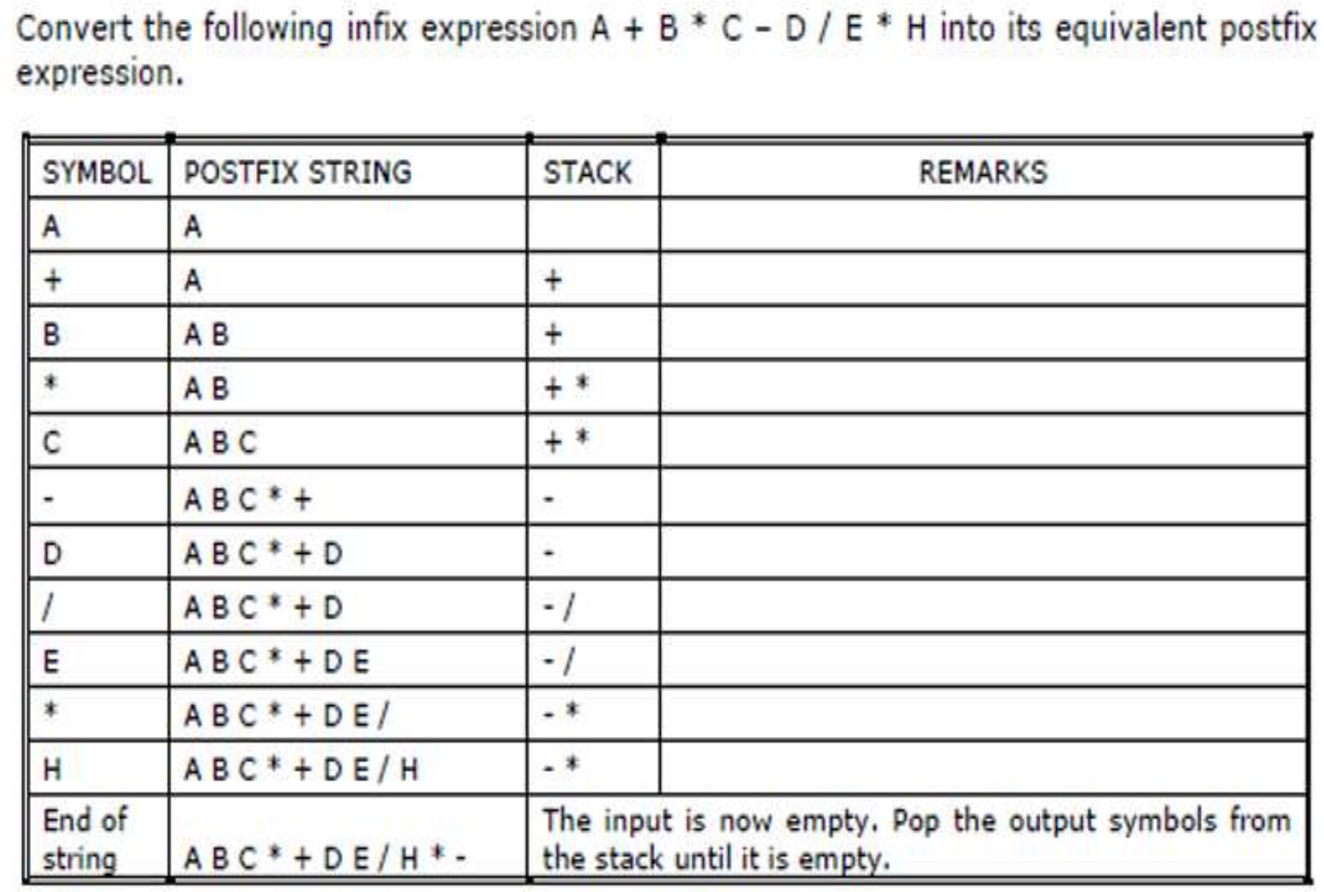
   d) If the scanned symbol is an operator, then go on removing all the operators from the stack and place them in the postfix expression, if and only if the precedence of the operator which is on the top of the stack is greater than (or greater than or equal) to the precedence of the scanned operator and push the scanned operator onto the stack otherwise, push the scanned operator onto the stack.

The three important features of postfix expression are:

1. The operands maintain the same order as in the equivalent infix expression.
2. The parentheses are not needed to designate the expression unambiguously.
3. While evaluating the postfix expression the priority of the operators is no longer relevant. We consider five binary operati ons: +, , *, / and $ or ↑ (exponentiation). For these binary operations, the following in the order of precedence (highest to lowest):



OPERATOR	PRECEDENCE	VALUE
Exponentiation ($ or ↑ or ^)	Highest	3
*, /	Next highest	2
+, -	Lowest	1



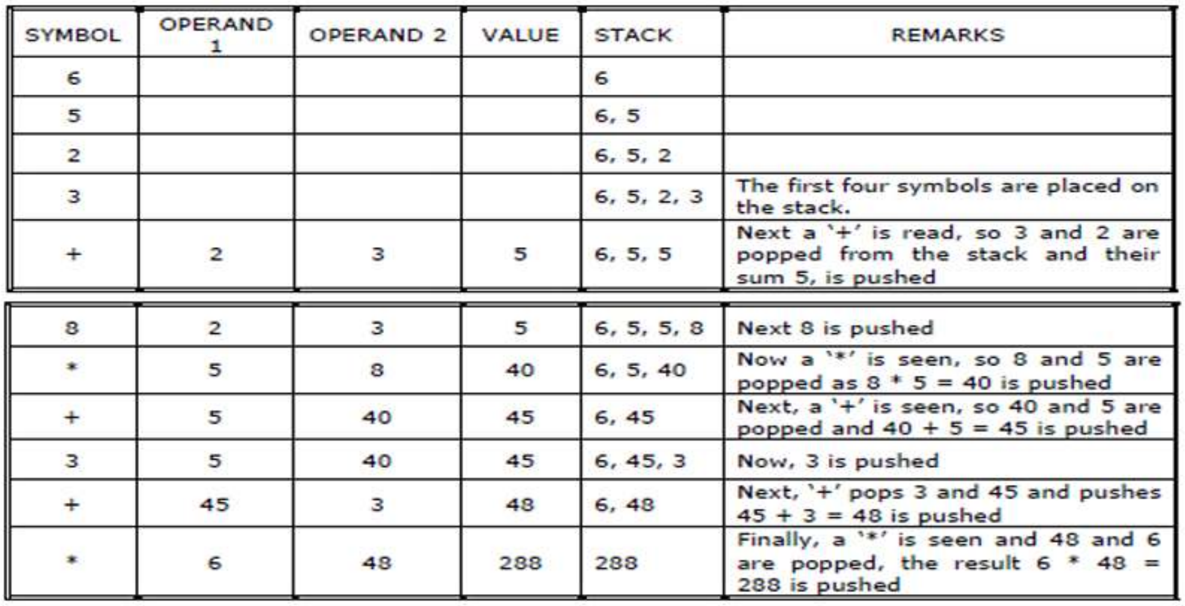
Convert the following infix expression A + B * C – D / E * H into its equivalent postfix expression.

SYMBOL	POSTFIX STRING	STACK	REMARKS
A	A		
+	A	+	
B	A B	+	
*	A B	+ *	
C	A B C	+ *	
-	A B C * +	-	
D	A B C * + D	-	
/	A B C * + D	- /	
E	A B C * + D E	- /	
*	A B C * + D E /	- *	
H	A B C * + D E / H	- *	
End of string	A B C * + D E / H * -	The input is now empty. Pop the output symbols from the stack until it is empty.	

Evaluation of postfix expression:

The postfix expression is evaluated easily by the use of a stack.
1. when a number is seen, it is pushed onto the stack;
2. When an operator is seen, the operator is applied to the two numbers that are popped from the stack and the result is pushed onto the stack.
3. When an expression is given in postfix notation, there is no need-to-know precedence rules; this is our obvious advantage.

Evaluate the postfix expression: 6 5 2 3 + 8 * + 3 + *



SYMBOL	OPERAND 1	OPERAND 2	VALUE	STACK	REMARKS
6				6	
5				6, 5	
2				6, 5, 2	
3				6, 5, 2, 3	The first four symbols are placed on the stack.
+	2	3	5	6, 5, 5	Next a '+' is read, so 3 and 2 are popped from the stack and their sum 5, is pushed
8	2	3	5	6, 5, 5, 8	Next 8 is pushed
*	5	8	40	6, 5, 40	Now a '*' is seen, so 8 and 5 are popped as 8 * 5 = 40 is pushed
+	5	40	45	6, 45	Next, a '+' is seen, so 40 and 5 are popped and 40 + 5 = 45 is pushed
3	5	40	45	6, 45, 3	Now, 3 is pushed
+	45	3	48	6, 48	Next, '+' pops 3 and 45 and pushes 45 + 3 = 48 is pushed
*	6	48	288	288	Finally, a '*' is seen and 48 and 6 are popped, the result 6 * 48 = 288 is pushed

Applications of Stack Data Structure

Although stack is a simple data structure to implement, it is very powerful. The most common uses of a stack are:

To reverse a word - Put all the letters in a stack and pop them out. Because of the LIFO order of stack, you will get the letters in reverse order.

In compilers - Compilers use the stack to calculate the value of expressions like 2 + 4 / 5 * (7 - 9) by converting the expression to prefix or postfix form.

In browsers - The back button in a browser saves all the URLs you have visited previously in a stack. Each time you visit a new page, it is added on top of the stack. When you press the back button, the current URL is removed from the stack, and the previous URL is accessed.