**Stacks-**

A stack is an ordered list in which insertion and deletion are done at one end, called *top*. The last element inserted is the first one to be deleted. Hence, it is called the **L**ast **i**n **F**irst **O**ut (**LIFO**) or **F**irst **i**n **L**ast **o**ut (**FILO**) list.

* When an element is inserted, it is called *push*.
* When an element is removed, it is called *pop.*
* Trying to pop empty stack is called *underflow* and trying to push an element in full stack is called *overflow*.

**Application** –

* Balancing of symbols
* Infix-to-postfix conversion
* Evaluation of postfix expression
* Implementing function calls (including recursion)
* Finding of spans (Stock markets)
* Page-visited history in a Web Browser [Back button]
* Undo sequence in a text editor
* Matching Tags in HTML and XML
* Tree traversal algorithms
* Simulating queues

**Limitation-**

The maximum size of the stack must first be defined and it can’t be changed. Trying to push a new element into a full stack causes an implementation-specific exception.

**Repeated Doubling in Dynamic Array-**

If the array is full, create a new array of twice the size, and copy the items.

Note- Too many doublings may cause memory exception.

**Array implementation-**

* Operations take constant time
* Expensive doubling operation every once in while
* Any sequence of n operations (starting from empty stack) – “amortized” bound takes time proportional to n.

**Linked List implementation**

Push operation is implemented by inserting element at the beginning of the list. Pop operation is implemented by deleting the node from the beginning(head).

* Grows and Shrinks gracefully
* Every operation takes constant time O(1)
* Every operation uses extra space and time to deal with references.

**Problems**

Checking balancing of symbols, Infix to postfix conversion, Postfix evaluation, getMin() in O(1)

**Checking balancing of symbols**

*Approach-1*

1. Create a stack
2. While (end of input is not reached){
   1. If the character read is not a symbol to be balanced, ignore it
   2. If the character is an opening symbol like (,[,{, push it onto the stack
   3. If it is a closing symbol like ),],}, then if the stack is empty report an error, otherwise pop the stack
   4. If the symbol popped is not like corresponding opening symbol, report an error.}
3. At end of input, if the stack is not empty report an error.

Time Complexity: O(n). Only one time scanning

Space Complexity: O(n). [for stack]

**Infix to postfix conversion**

**Infix**: An infix expression is a single letter, or an operator, proceeded by one infix string and followed by another infix string

A

A+B

(A+B)+(C-D)

**Prefix:** A prefix expression is a single letter, or an operator, followed by two prefix strings. Every prefix string longer than a single variable contains an operator, first operand and second operand

A

+AB

++AB-CD

**Postfix:** A postfix expression (Reverse polish notation) is a single letter or an operator, preceded by two postfix strings. Every postfix string longer than a single variable contains first and second operands followed by an operator.

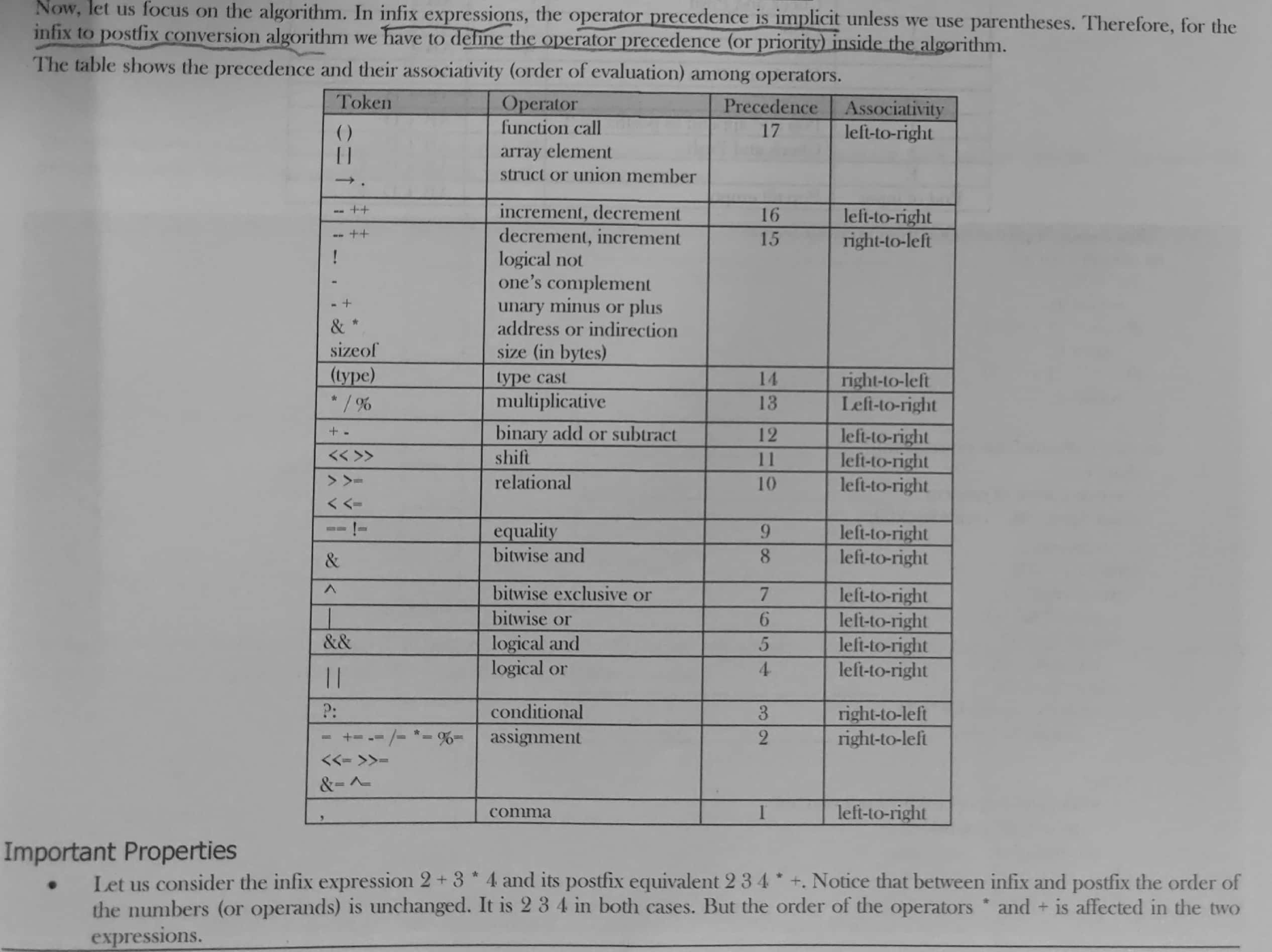
A

AB+

AB+CD-+

Time to evaluate a postfix and prefix expression is O(n), where n is number of elements in the array.

|  |  |  |
| --- | --- | --- |
| **Infix** | **Prefix** | **Postfix** |
| A+B | +AB | AB+ |
| A+B-C | -+ABC | AB+C- |
| (A+B)\*C-D | -\*+ABCD | AB+C\*D- |



* Only one stack is enough to convert an infix expression to postfix expression. The stack that we use in the algorithm will be used to change the order of operators from infix to postfix. The stack we use will only contain operators and the open parentheses symbol ‘(’.

Postfix expressions don’t contain parentheses. We shall not output the parentheses in the postfix output.

*Approach-1*

1. Create a stack
2. For each character t in the input stream{

If (t is an operand)

Output t

Else if(t is a right parenthesis){

Pop and output tokens until a left parenthesis is popped (but not output)}

Else {

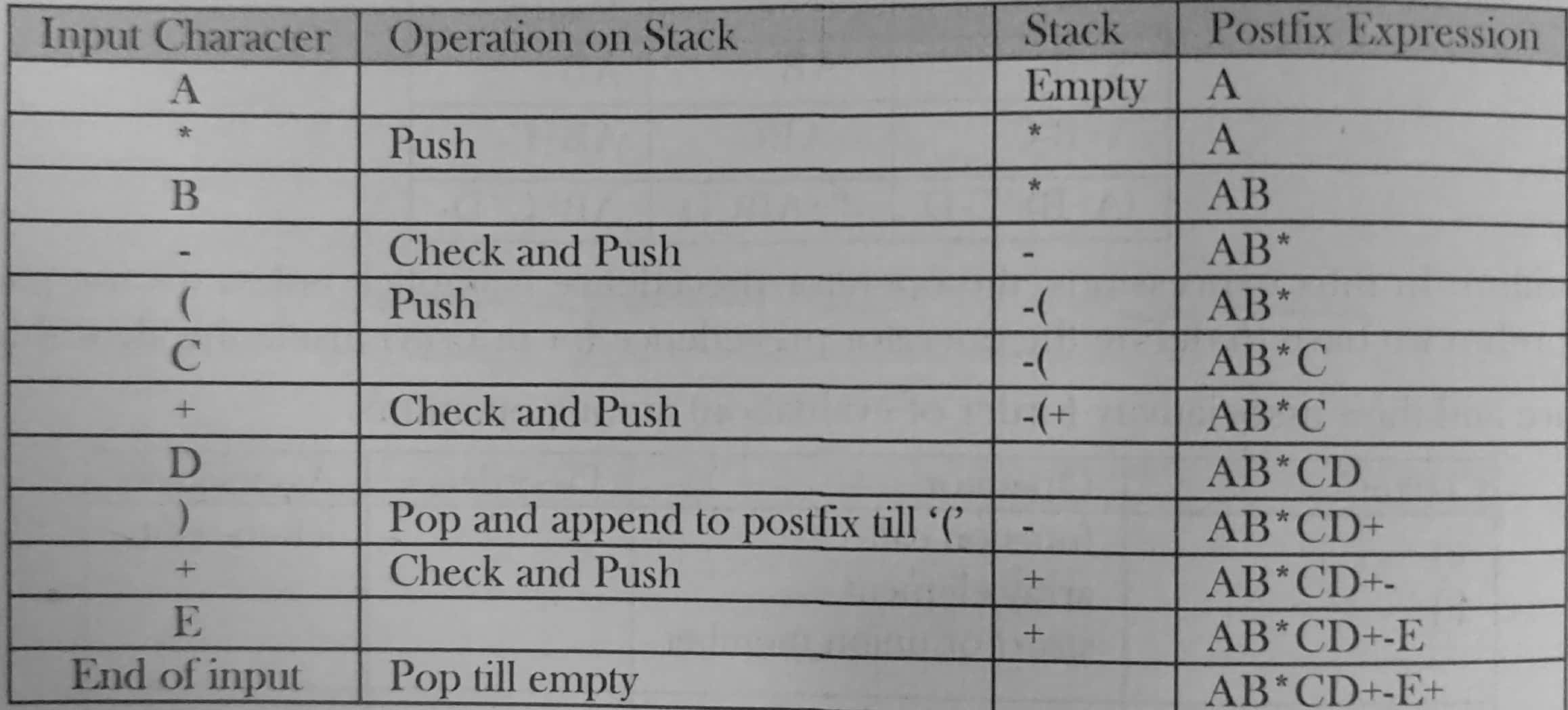
Pop and output tokens until one of lower priority than t is encountered or a left parenthesis is encountered or the stack is empty

Push t}

}

1. Pop and output tokens until the stack is empty

Example – A \* B – (C + D) + E



**Postfix evaluation**

*Approach-1*

1. Scan the postfix string from left to right
2. Initialize an empty stack
3. Repeat steps 4 and 5 till all characters are scanned
4. If the scanned character is an operand, push it onto the stack
5. If the scanned character is an operator and if the operator is a unary operator, then pop an element from the stack. If the operator is a binary operator, then pop two elements from stack. After popping the elements, apply the operator to those popped elements. Let the result of this operation be retVal onto the stack
6. After all characters are scanned we will have only one element in the stack
7. Return top of the stack as result

*Approach-2*

Using 2 stacks we can evaluate an infix expression in 1 pass without converting to postfix.

1. Create an empty operator stack
2. Create an empty operand stack
3. For each token in the input string:
   1. Get the next token in the infix string
   2. If next token is an operand, place it on the operand stack
   3. If next token is an operator
      1. Evaluate the operator
4. While operator stack is not empty, pop operator and operands, evaluate left operator right and push result onto operand stack
5. Pop result from operator stack