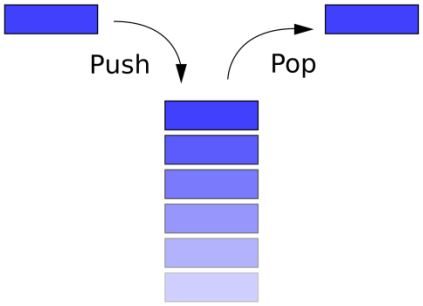
STACKS

1. Explain in detail about Stack ADT.

Stack ADT is an ordered list in which all insertions and deletions are made at one end called TOP. Insertion of an element is known as *push* and removal of an element is known as *pop.* Stack is a Last-In First-Out (LIFO) data structure. In a LIFO data structure, the last element added to the structure must be the first one to be removed.



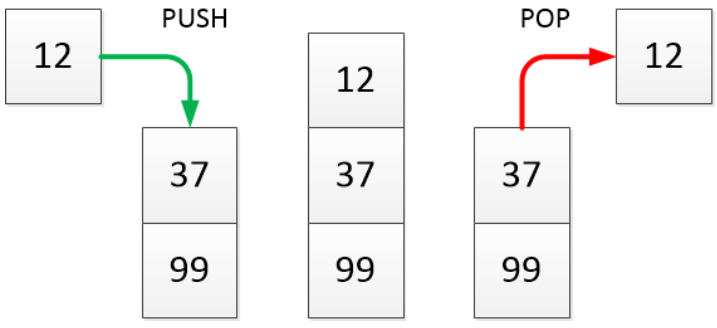
Following operations can be performed on a stack.

1. PUSH: PUSH operation will add elements onto the stack. It is carried out in following steps:

- Ensure that stack is not full, to avoid "STACK OVERFLOW"
- Increment the variable top so that it refers to location in which element is to be inserted.
- Add element onto the location of stack pointed by top.

2. POP: This operation will remove elements from stack. It is carried out in the following steps.

- Check whether there is at least one element in the stack, to avoid STACK UNDERFLOW.
- Store element pointed by top in a temporary variable
- Decrement top by 1
- Return value stored in temporary variable



2. Write the algorithm to convert an infix expression to postfix expression and explain it with an example.

Infix Expression: Any expression in the standard form like "2*3-4/5" is an Infix (Inorder) expression.

Postfix Expression: The Postfix (Postorder) form of the above expression is "23*45/-".

Infix to Postfix Conversion:

In normal algebra we use the infix notation like a+b*c. The corresponding postfix notation is abc*+. The algorithm for the conversion is as follows :

1. Scan the Infix string from left to right.
2. Initialize an empty stack and a Postfix string.
3. If the scanned character is an operand, add it to the Postfix string.
4. If the scanned character is an operator,
    a. compare the precedence of the character with the element on top of the stack(call it topStack).
    b. If topStack has higher precedence over the scanned character, Pop the stack else Push the scanned character to stack.
    c. Repeat this step as long as stack is not empty and topStack has precedence over the character.
5. Repeat the steps 3 & 4 till all the characters are scanned.

(After all characters are scanned, we have to add any character that is in the stack to the Postfix string.)

6. If stack is not empty add topStack to Postfix string and Pop the stack. Repeat this step as long as stack is not empty.
7. Return the Postfix string.

Example :

Let us see how the above algorithm will be implemented using an example.

Infix String: a+b*c-d

Initially the Stack is empty and our Postfix string has no characters. Now, the first character scanned is 'a'. 'a' is added to the Postfix string. The next character scanned is '+'. It being an operator, it is pushed to the stack.(Since stack is empty initially, precedence of top of stack is considered to be less than precedence of '+')

Next character scanned is 'b' which will be placed in the Postfix string. Next character is '*' which is an operator. Now, the top element of the stack is '+' which has lower precedence than '*', so '*' will be pushed to the stack.

The next character is 'c' which is placed in the Postfix string. Next character scanned is '-'. The topmost character in the stack is '*' which has a higher precedence than '-'. Thus '*' will be popped out from the stack and added to the Postfix string. Even now the stack is not empty. Now the topmost element of the stack is '+' which has equal priority to '-'. So pop the '+' from the stack and add it to the Postfix string. The '-' will be pushed to the stack.

Next character is 'd' which is added to Postfix string. Now all characters have been scanned so we must pop the remaining elements from the stack and add it to the Postfix string. At this stage we have only a '-' in the stack. It is popped out and added to the Postfix string. So, after all characters are scanned, this is how the stack and Postfix string will be:

End result:

Infix String: a+b*c-d

Postfix String: abc*+d-

3. Write the algorithm for evaluating the given postfix expression and explain it.

1. Scan the Postfix string from left to right.
2. Initialize an empty stack.
3. If the scanned character is an operand, add it to the stack.
4. If the scanned character is an Operator, then pop the top most element of the stack(topStack) into a variable temp. Now evaluate topStack(Operator)temp. Let the result of this operation be retVal. Pop the stack and Push retVal into the stack.
5. Repeat steps 3 & 4 till all the characters are scanned.
6. After all characters are scanned, we will have only one element in the stack. Return topStack.

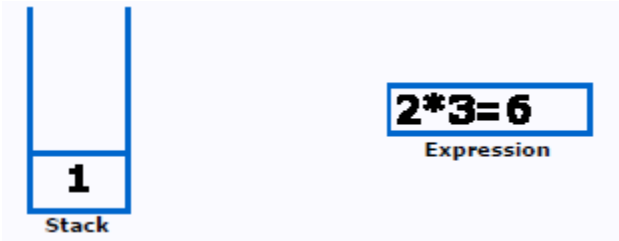
Example:

Let us see how the above algorithm will be implemented using an example.

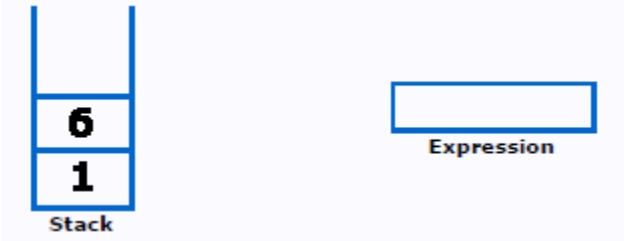
Postfix String : 123*+4-

Initially the Stack is empty. Now, the first three characters scanned are 1,2 and 3, which are operands. Thus they will be pushed into the stack in that order.

Next character scanned is "*", which is an operator. Thus, we pop the top two elements from the stack and perform the "*" operation with the two operands. The second operand will be the first element that is popped.

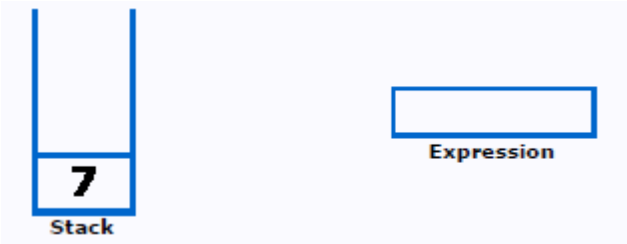


The value of the expression (2*3) that has been evaluated (6) is pushed into the stack.

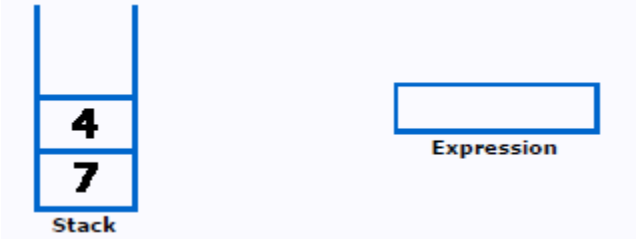


Next character scanned is "+", which is an operator. Thus, we pop the top two elements from the stack and perform the "+" operation with the two operands. The second operand will be the first element that is popped.

The value of the expression (1+6) that has been evaluated (7) is pushed into the stack.

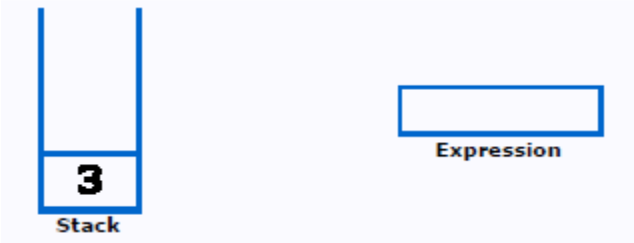


Next character scanned is "4", which is added to the stack.



Next character scanned is "-", which is an operator. Thus, we pop the top two elements from the stack and perform the "-" operation with the two operands. The second operand will be the first element that is popped.

The value of the expression (7-4) that has been evaluated (3) is pushed into the stack.



Now, since all the characters are scanned, the remaining element in the stack (there will be only one element in the stack) will be returned.

End result:

Postfix String: 123*+4-

Result: 3

4. Explain how to use a stack for delimiter matching?

Arithmetic expressions such as (2 + 3 / [4+3 – {6 * 4}] + 8) uses delimiters like ( ), [ ], { } to specify the order of operations. These symbols (parentheses or brackets) must appear in a balanced fashion. Matched delimiters in this context means that each opening symbol has a corresponding closing symbol and the pairs of symbols are properly nested. So, for example, the inputs (x(y) (z)) and a ({(b)}c) are balanced, while the inputs w)(x) and p({(q)r)} are not. Following are the steps of the algorithm for delimiter matching using a stack:

1. Scan the Input expression from left to right.
2. Initialize an empty stack.
3. If the scanned character is an opening parenthesis or an opening bracket, push it onto the stack.
4. If the scanned character is a closing parenthesis, check if the symbol on the top of the stack is an opening parenthesis. If it is not report error and terminate the process. Otherwise, pop the top element and continue.
5. If the scanned character is a closing square bracket, check if the symbol on the top of the stack is an opening square bracket. If it is not report error and terminate the process. Otherwise, pop the top element and continue.
6. If the scanned character is a closing curly bracket, check if the symbol on the top of the stack is an opening curly bracket. If it is not report error and terminate the process. Otherwise, pop the top element and continue.
7. Repeat the steps 3, 4, 5 & 6 till all the characters in the input expression are scanned.
8. If stack is not empty report error and terminate the process. Else, report that the expression has delimiters matched.

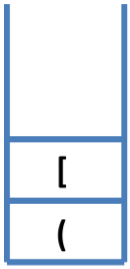
Example:

Input Expression: (a+b/[f*d/{g+h}+r])

Initially the Stack is empty. The first character scanned is '('. This will be pushed onto the stack.

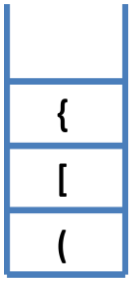
Stack

The next four characters scanned are 'a', '+', 'b', and '/'. These will be ignored since they are neither parenthesis nor brackets. Next character scanned is '['. Since this is a delimiter, it will be pushed onto the stack.



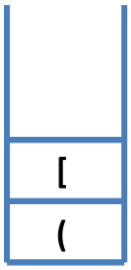
Stack

After the next four characters, which are not delimiter characters, character '{'will be pushed onto the stack.



Stack

The next three characters 'g', '+', and 'h' will be ignored. The next character scanned is '}'. Since the top element is an opening curly bracket, the top element of the stack will be popped.



After '+' and 'r', the next character scanned is ']'. Since the top element is an opening square bracket, the top element of the stack will be popped.

The next character scanned is ')', which is matching delimiter for the symbol on the top of the stack. So stack is popped once again, yielding an empty stack.

Since all the characters of the input expression are scanned, and stack is empty, the message 'DELIMETERS MATCHED' will be reported and the process gets terminated.