Q1. (20pts.)

Fill the hash tables below for the given number series and their respective hashing and collision resolution algorithms. (0.66pts for each number)

a) The number series: **5, 18, 3, 7, 17, 0, 13**
Hash Table size: 7
Hash Function: Modular Arithmetic (for table size)
Collision Resolution method: Linear Probing

0	7
1	0
2	13
3	3
4	18
5	5
6	17

c) The number series: **15, 18, 1, 89, 17, 0, 4, 7, 21, 5, 6, 13**
Hash Table size: 7
Hash Function: Modular Arithmetic (for table size)
Collision Resolution method: Chaining

0	0->7->21
1	15->1
2	
3	17
4	18->4
5	89->5
6	6->13

(Burada Linked List olarak çizemedim)
Node -> Node şeklinde gösteriyorum array içinde

b) The number series: **55, 71, 78, 32, 45**
Hash Table size: 10
Hash Function: Mid-Square (pick middle 2 digits)
Collision Resolution method: Quadratic Probing

0	
1	
2	55
3	32
4	71
5	
6	45
7	
8	78
9	

d) The number series: **12, 13, 111, 4, 1125, 89**
Hash Table size: 10
Hash Function: **(Sum of digits)** mod 10
Collision Resolution Method: Random Probing
Random number list: 1, 3, 4, 8, 12, 23, 36, 55

0	
1	
2	
3	12
4	13
5	4
6	111
7	89
8	
9	1125

Q2. (15 pts.)

a) Sort the following array using the ***selection sort*** algorithm. Show the content of the array for the first 2 passes.

69	45	3	99	15	17

3	45	69	99	15	17

3	15	69	99	45	17

b) Sort the following array using ***quicksort*** algorithm. Show the content of the array for the first 2 passes. Use **first element** of the array as **pivot**.

22	13	36	6	15	30

13	6	15	22	36	30

6	13	15	22	30	36

b) Sort the following array using ***insertion*** algorithm. Show the content of the array for the first 2 passes. Use first element of the array as pivot.

61	55	27	134	1	25

55	61	27	134	1	25

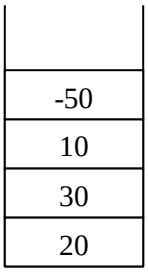
27	55	61	134	1	25

Q3. (10 pts.) Given a binary tree, you are to write a function that returns the minimum of all keys stored in the tree. Return 0 if the tree is empty. Assume that a BinaryTreeNode contains fields {key, left, right} with obvious meanings.

(**Hint:** Function name: int Minimum(BinaryTreeNode root){}).

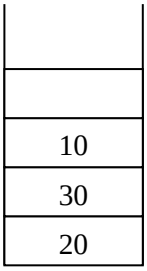
```
int Minimum (BinaryTreeNode root){
        if (root.key == null)
        return 0;
        BinaryTreeNode p = root;
        while (p.left != null){
                        p = p.left;
                        } //end-while
         return p.key;
} //end-Min
```

Q4. (10 pts.) Assume that the following Stack is given. Draw the final state of the Stack after calling all the generic stack functions listed below one after another.
Hint: STop() function can also be named as ShowTop().

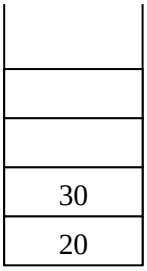



```
int x=25,y=60,z=40,w=15;
pop();
STop();
pop();
STop();
push(w);
STop();
z=pop();
push(y);
push(z);
STop();
pop();
STop();
push(x);
```

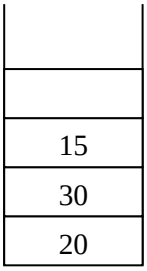
Stop=10

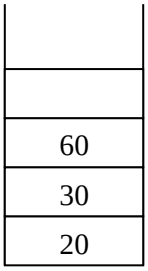


Stop=30

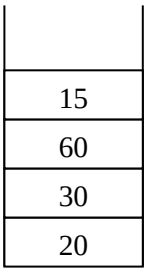


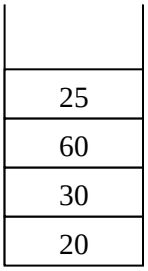
Stop=15 , z=pop->15





Stop=15, pop->15, Stop=60



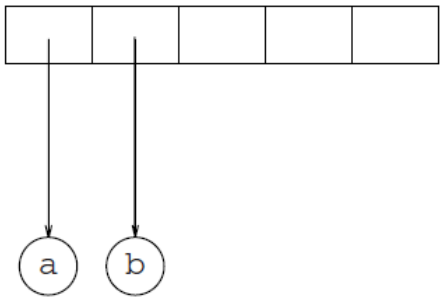


Q 5. (10 points) Convert the given postfix expression into an expression (binary) tree.

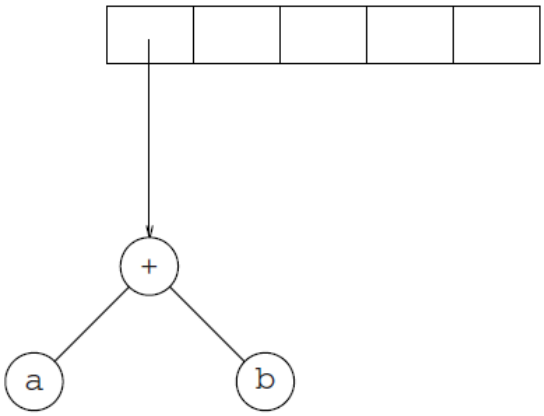
Postfix expression: a b + c d e + * *

(Hint: You have already known an algorithm to convert infix to postfix, you can generate expression trees from the two common types of input.**)**

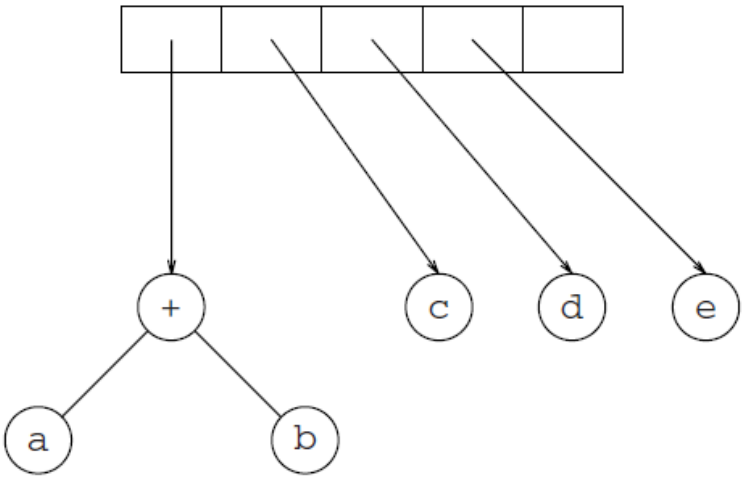
The first two symbols are operands, so we create one-node trees and push pointers to them onto a stack.[2]



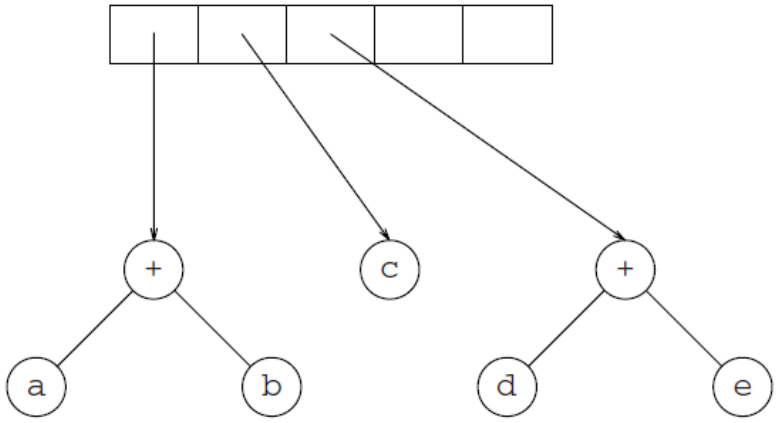
Next, a + is read, so two pointers to trees are popped, a new tree is formed, and a pointer to it is pushed onto the stack.



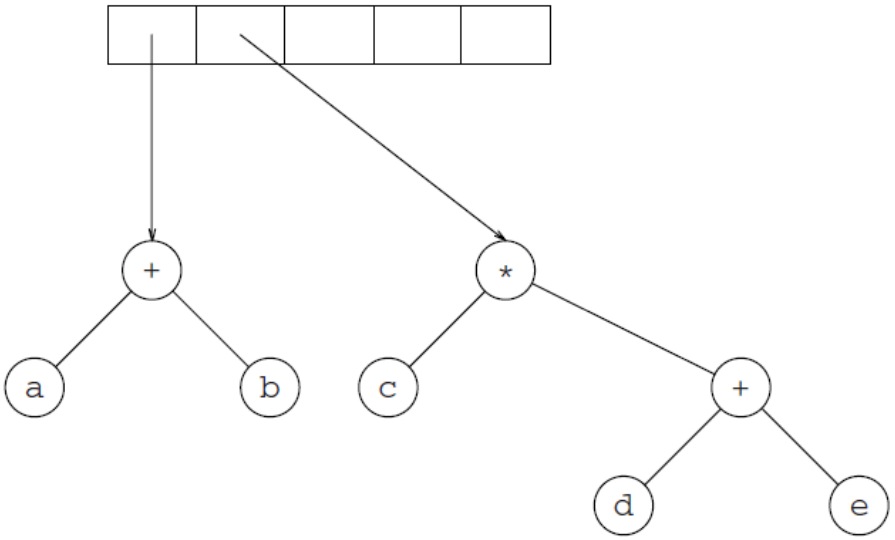
Next, `c`, `d`, and `e` are read, and for each a one-node tree is created and a pointer to the corresponding tree is pushed onto the stack.



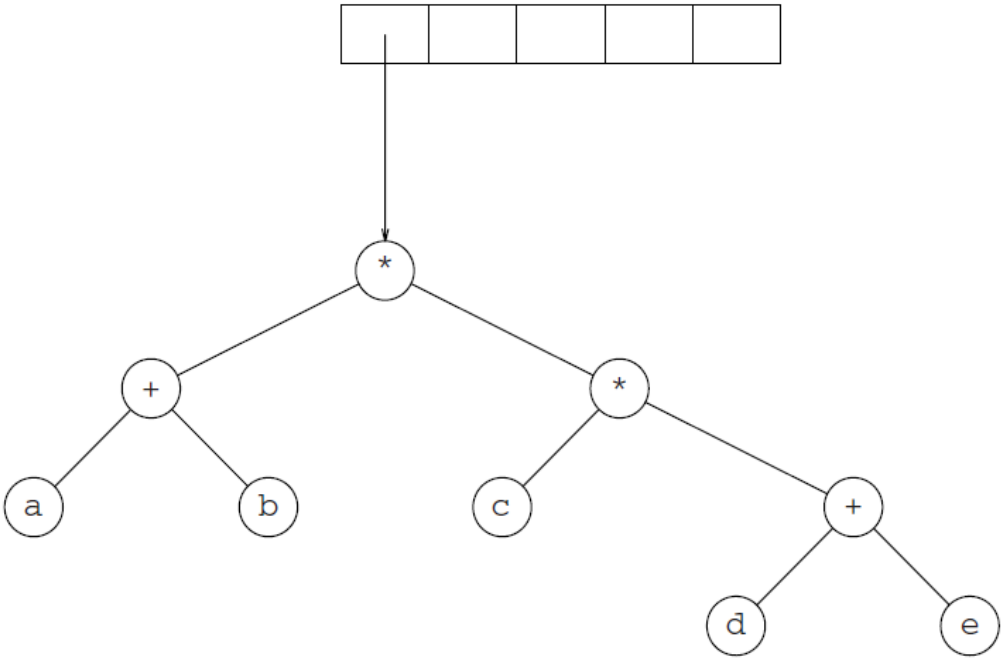
Now a + is read, so two trees are merged.



Continuing, a * is read, so we pop two tree pointers and form a new tree with a * as root.



Finally, the last symbol is read, two trees are merged, and a pointer to the final tree is left on the stack.

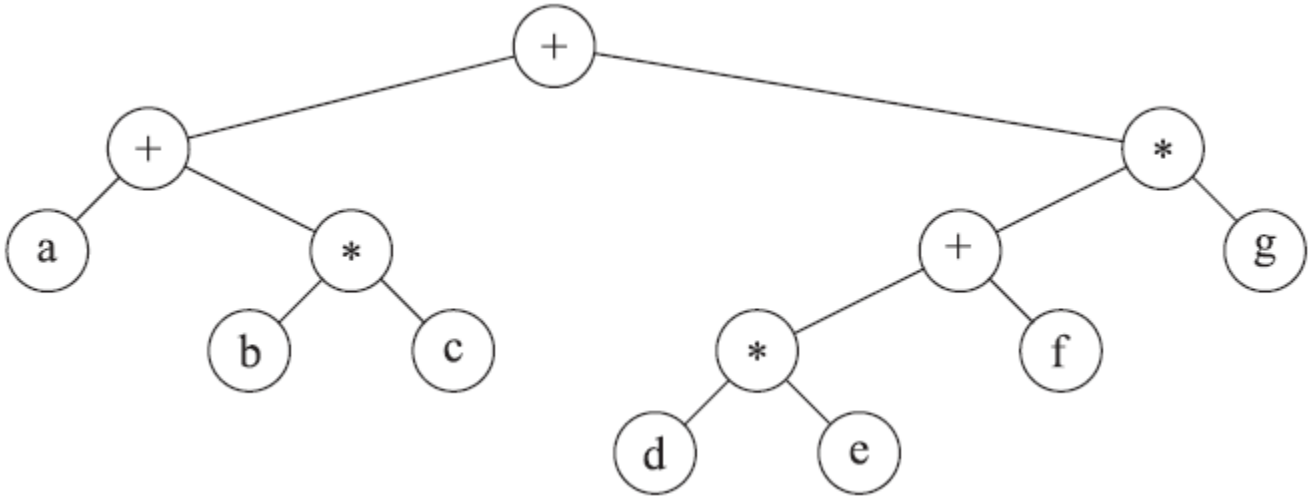


Q 6. (10 points) Convert the given infix expression into an expression (binary) tree.

Infix expression: Expression tree for (a + b * c) + ((d * e + f) * g)

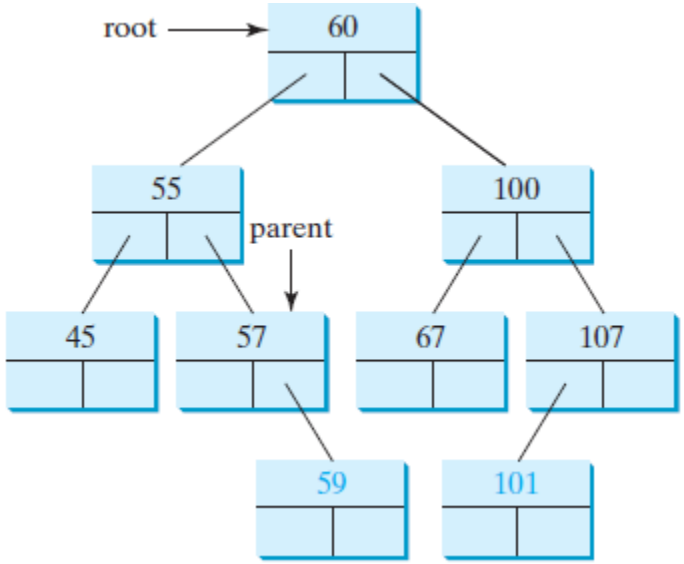
Firstly convert infix to postfix expression, then draw the expression (binary) tree.

Postfix expression: a b c * + d e * f + g * +



What is the prefix expression of this infix expression?
Prefix expression: + + a * b c * + * d e f g

Q 7. (10 points) Find the traversal path of the given (binary) tree.



The inorder traversal is: 45 55 57 59 60 67 100 101 107

The postorder traversal is: 45 59 57 55 67 101 107 100 60

The preorder traversal is: 60 55 45 57 59 100 67 107 101

Q8. (20 pts.) Make the following conversions.

a. **Infix:** a + b * c - (d + e) / f **Prefix: - + a * b c / + d e f**

b. **Postfix:** a d + b c / * e - **Infix: (a +d) * (b / c)- e**

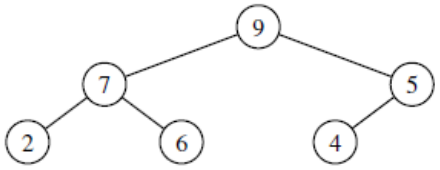
c. **Prefix:** - + a / b c * d - e f **Postfix: a b c / + d e f - * -**

d. **Postfix:** a b + c / d * e f + - **Prefix: - * / + a b c d + e f**

Q 9. (10 points) Apply heapsort to given initial array; A[N]=[5 2 6 7 9 4].

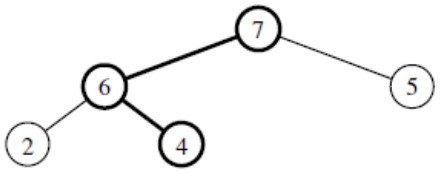
(a)

9	7	5	2	6	4



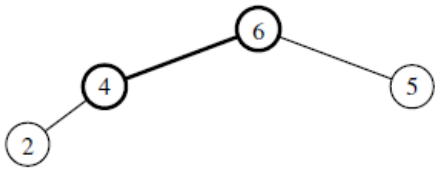
(b)

7	6	5	2	4	9



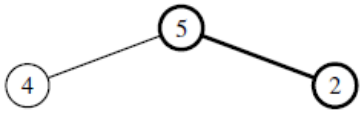
(c)

6	4	5	2	7	9



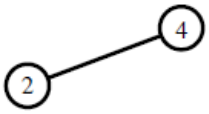
(d)

5	4	2	6	7	9



(e)

4	2	5	6	7	9



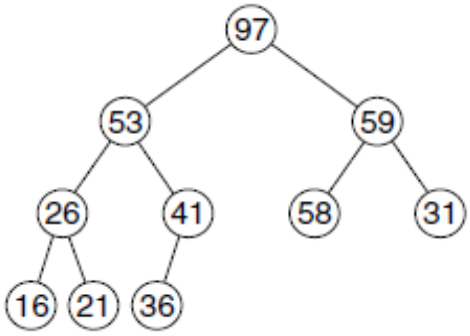
(f)

2	4	5	6	7	9

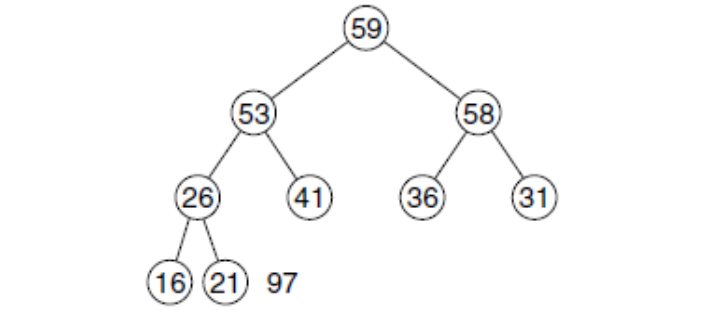
Q 10. (10 points) Apply heapsort to given initial array: H[N]=[59, 36,58, 21, 41, 97, 31, 16, 26, 53].

figure 21.25

Max heap after the `buildHeap` phase



97	53	59	26	41	58	31	16	21	36				
0	1	2	3	4	5	6	7	8	9	10	11	12	13



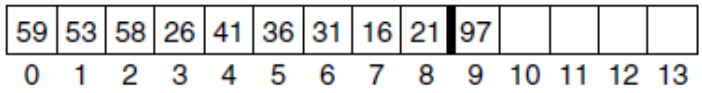
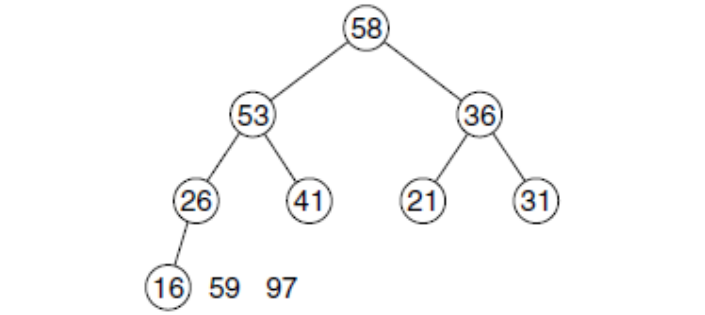


figure 21.26

Heap after the first deleteMax operation



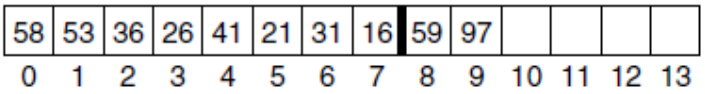


figure 21.27

Heap after the second deleteMax operation

Q11. (20pts.)

Fill the hash tables below for the given number series and their respective hashing and collision resolution algorithms. (0.66pts for each number)

a) The number series: **89, 18, 49, 58, 09,** Hash Table size: 10, Hash Function: Modular Arithmetic (for table size), Collision Resolution method: Linear Probing

```
hash ( 89, 10 ) = 9
hash ( 18, 10 ) = 8
hash ( 49, 10 ) = 9
hash ( 58, 10 ) = 8
hash (  9, 10 ) = 9
```

	After insert 89	*After insert 18*	*After insert 49*	*After insert 58*	*After insert 9*
0			49	49	49
1				58	58
2					9
3					
4					
5					
6					
7					
8		18	18	18	18
9	89	89	89	89	89

figure 20.5

Linear probing hash table after each insertion

b) Collision Resolution method: Quadratic Probing

```
hash ( 89, 10 ) = 9
hash ( 18, 10 ) = 8
hash ( 49, 10 ) = 9
hash ( 58, 10 ) = 8
hash (  9, 10 ) = 9
```

	After insert 89	*After insert 18*	*After insert 49*	*After insert 58*	*After insert 9*
0			49	49	49
1					
2				58	58
3					9
4					
5					
6					
7					
8		18	18	18	18
9	89	89	89	89	89

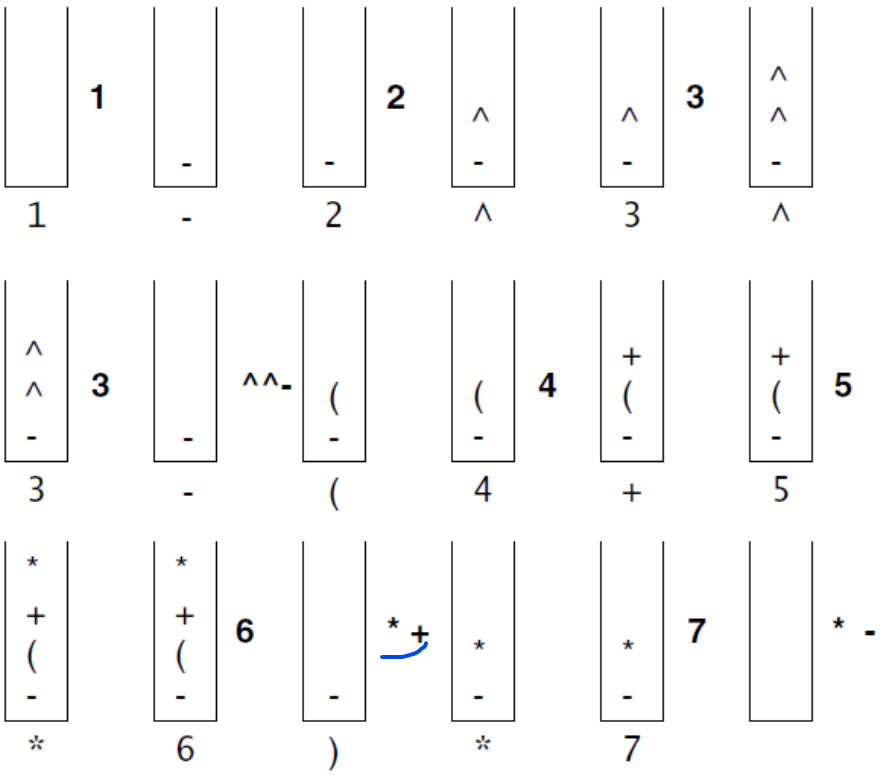
figure 20.7

A quadratic probing hash table after each insertion (note that the table size was poorly chosen because it is not a prime number).

Q 12. (10 points) Convert the given infix expression to a postfix expression and evaluate it.

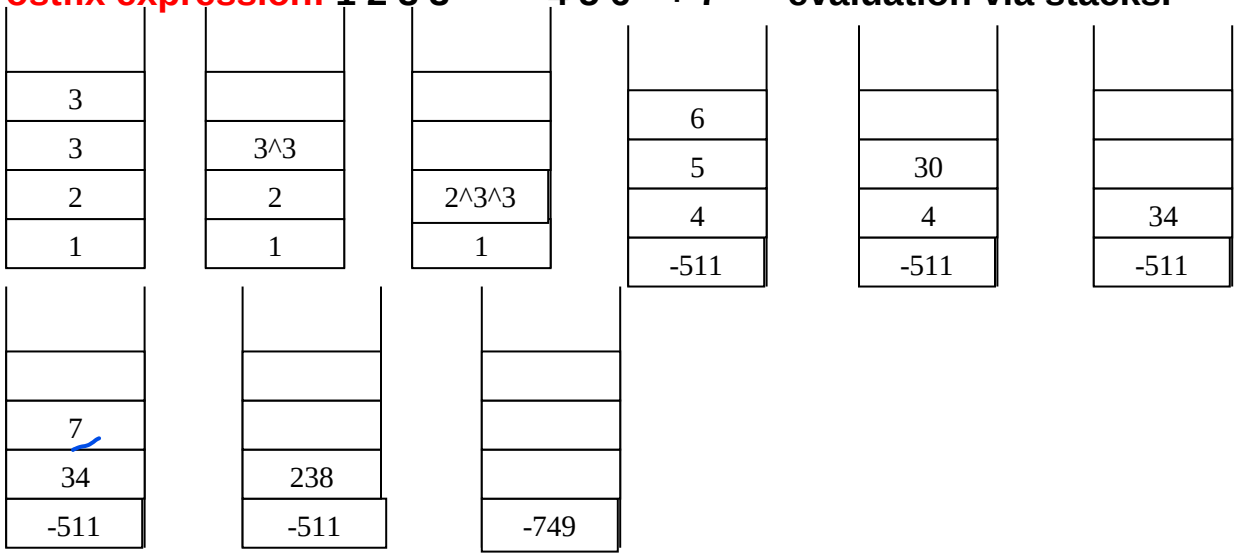
Infix expression: 1 - 2 ^ 3 ^ 3 - (4 + 5 * 6) * 7

Infix: 1 - 2 ^ 3 ^ 3 - (4 + 5 * 6) * 7



Postfix: 1 2 3 3 ^ ^ - 4 5 6 * + 7 * -

Postfix expression: 1 2 3 3 ^ ^ - 4 5 6 * + 7 * - evaluation via stacks.



Q 13. (10 points) Convert the given infix expression to a postfix expression and evaluate it.

Infix expression: (1 - 2) - ((((4 ^ 5) * 3) * 6) / (7 ^ (2 ^ 2)))
Postfix expression: 1 2 - 4 5 ^ 3 * 6 * 7 2 2 ^ ^ / -

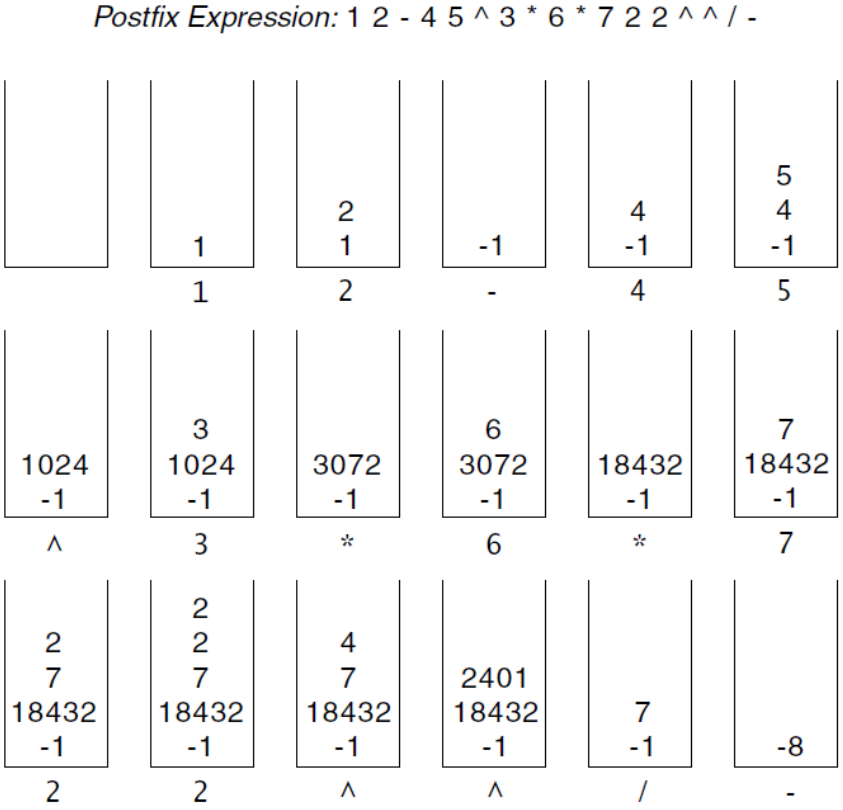


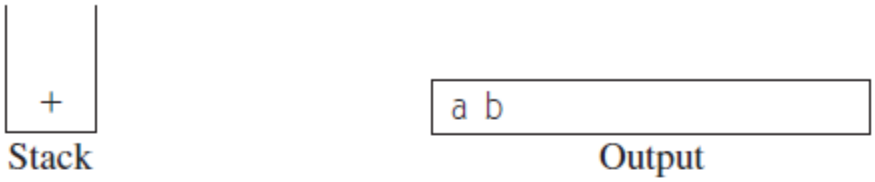
figure 11.11

Steps in the evaluation of a postfix expression

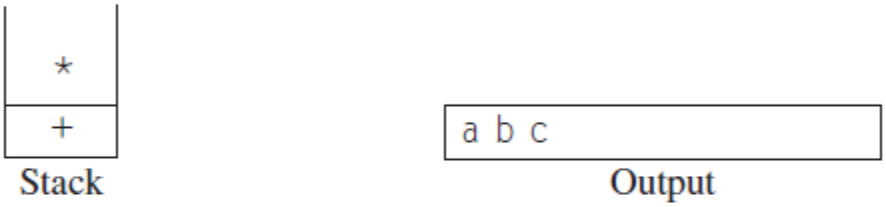
Q 14. (10 points) Convert the given infix expression to a postfix expression via stacks.

Infix expression: a + b * c + (d * e + f) * g
Postfix expression: a b c * + d e * f + g * +.

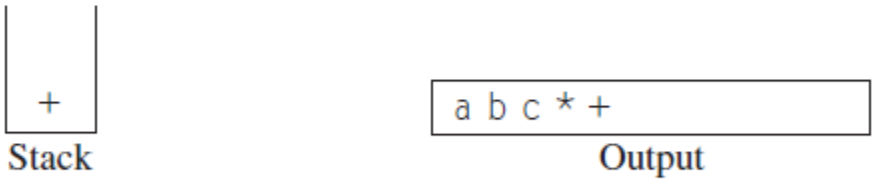
Then + is read and pushed onto the stack. Next b is read and passed through to the output. The state of affairs at this juncture is as follows:



Next, a * is read. The top entry on the operator stack has lower precedence than *, so nothing is output and * is put on the stack. Next, c is read and output. Thus far, we have

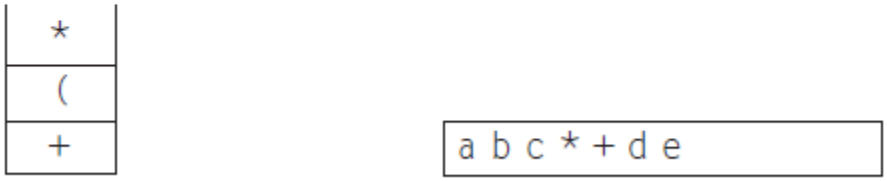


The next symbol is a +. Checking the stack, we find that we will pop a * and place it on the output; pop the other +, which is not of *lower* but equal priority, on the stack; and then push the +.

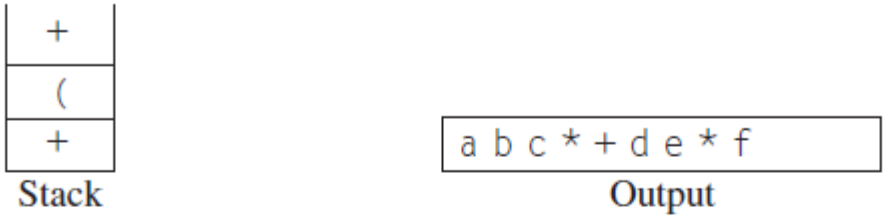


The next symbol read is a (. Being of highest precedence, this is placed on the stack. Then d is read and output.

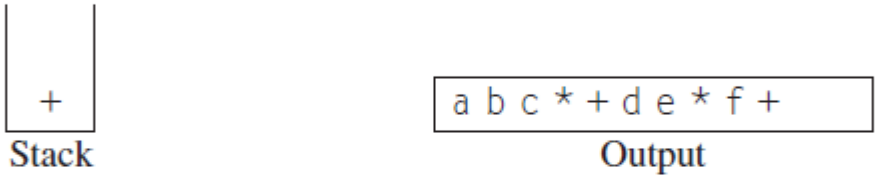
We continue by reading a *. Since open parentheses do not get removed except when a closed parenthesis is being processed, there is no output. Next, e is read and output.



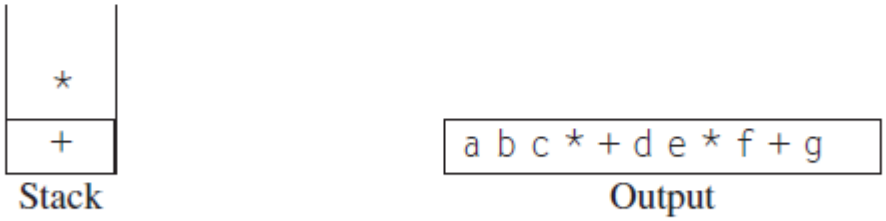
The next symbol read is a +. We pop and output * and then push +. Then we read and output f.



Now we read a), so the stack is emptied back to the (. We output a +.



We read a * next; it is pushed onto the stack. Then g is read and output.



The input is now empty, so we pop and output symbols from the stack until it is empty.