CLASS TEST

Sl. : 01SK_CS_ABCD_070722

MADE EASY
India's Best Institute for IES, GATE & PSUs

Delhi | Bhopal | Hyderabad | Jaipur | Lucknow | Pune | Bhubaneswar | Kolkata | Patna

Web: www.madeeasy.in | **E-mail:** info@madeeasy.in | **Ph:** 011-45124612

PROGRAMMING & DATA STRUCTURES

COMPUTER SCIENCE & IT

Date of Test : 07/07/2022

ANSWER KEY ➤

1.	(a)	7.	(b)	13.	(b)	19.	(b)	25.	(d)
2.	(d)	8.	(c)	14.	(a)	20.	(a)	26.	(d)
3.	(b)	9.	(a)	15.	(d)	21.	(b)	27.	(c)
4.	(d)	10.	(a)	16.	(b)	22.	(a)	28.	(c)
5.	(a)	11.	(d)	17.	(d)	23.	(d)	29.	(c)
6.	(d)	12.	(a)	18.	(c)	24.	(a)	30.	(d)

DETAILED EXPLANATIONS

1. (a)

In case of full or complete binary tree,

$$\text{Minimum height } (h_{min}) = \lceil \log_2 (n+1) \rceil$$

Hence, last element will be stored at $2^{h_{min}} - 1$

$$\text{Minimum size} = 2^{\lceil \log_2 (n+1) \rceil} - 1.$$

2. (d)

(a) 6, 8, 4, 7, 5

After popping element 6, only 4 can be popped, hence this permutation is not possible.

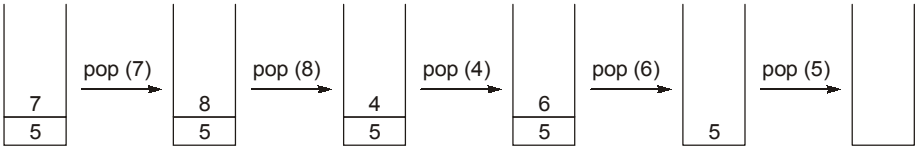
(b) 6, 4, 5, 7, 8

After performing pop operation on element 6, 4 now only element 8 can be popped.

(c) 6, 4, 7, 8, 5

After 6, 4 elements are popped, now element 7 can only be popped iff 8 has already been popped.

(d) 7, 8, 4, 6, 5



3. (b)

Here m represent the number of rows and n represents the number of column.

m = 2, n = 3

∗ (A[0] + 0) = A[0][0] = 1

∗ (A[1] + 0) = A[1][0] = 4

Similarly it will access all the element.

∴ 1 4 2 5 3 6 is the output printed by the program.

4. (d)

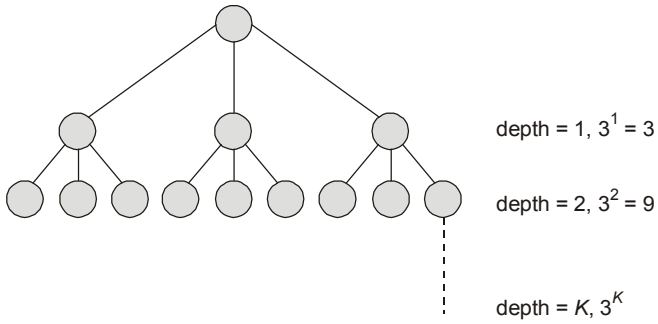
- Void pointers can't be used for dereferencing because each variable type takes different amount of memory.
- Since compiler can not know, after how many bytes is the next variables located, hence arithmetic operations can't be performed.
- Default value of extern storage class is 0.

5. (a)

Number of children by every node = n

depth of tree = k

Let $n = 3$



Hence the maximum number of leaves that 'T' an have in n^k.

6. (d)

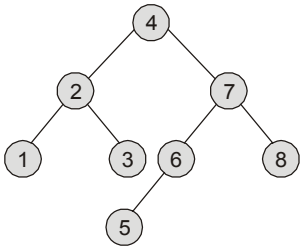
Let $n = 3$

$$r = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & -3 & -8 \\ -3 & 2 & -5 \\ -8 & -5 & 3 \end{bmatrix} \end{array}$$

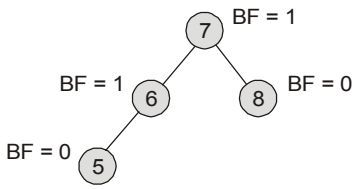
$$1 + 2 + 3 + 2 \times (-3 - 8 - 5) = -26 \neq \frac{n(n+1)}{2}$$

7. (b)

The tree constructed will be,



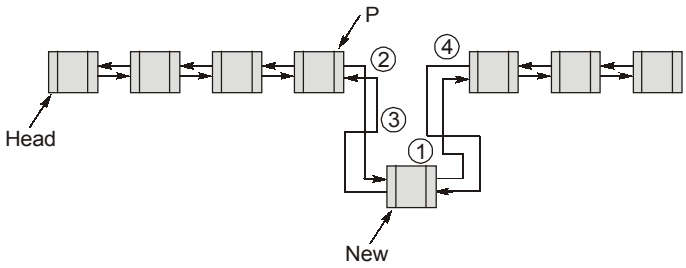
- 'T' has 4 leaf nodes.
- Subtree rooted at node '7' satisfies the AVL property.



ΘBF =Balancing Factor

- In a heap, nodes are started from by most pointer, hence node '5' is violating the basic heap property.

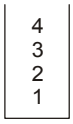
8. (c)



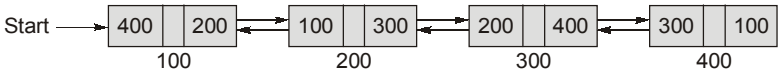
1. new → next = P → next;
2. P → next = new;
3. new → prev = P;
4. (new → next) → prev = new;

9. (a)

Consider a stack



The characteristic of the stack is both insertions and deletions are performed from one end.
If, it is implemented with a link lists, then both insertions and deletions are needed to be performed from the end.
Since, the linked list is a doubly circular linked list, hence the start node will have address of last node.



So, both the operations can be performed in $O(1)$ time.

10. (a)

Initial value are $p = -3$
$q = 2$
$r = 0$

&& has more priority than ++

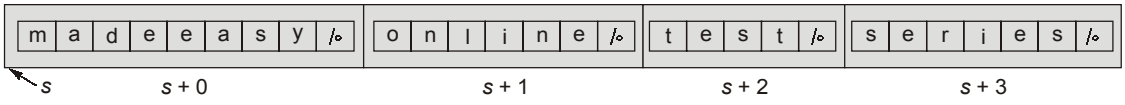
$++p = -2$
$++q = 3$

Since, both are non zero, hence expression becomes true. r++ need not be checked for calculating 's' because it's an OR operation so $s = 1$ i.e. the truth value of the expression.

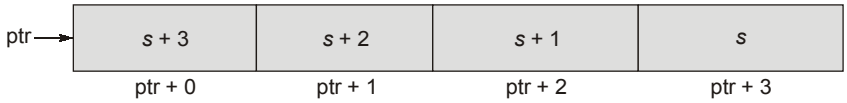
$$t = p + q + s++$$
$$= -2 + 3 + 1$$
$$= 2$$

11. (d)

In this problem we have an array of char pointers pointing to start of 4 strings i.e.,



We have ptr which is pointer to a pointer of type char and a variable *p* which is a pointer to a pointer of type char.



p = ptr; *p* ptr

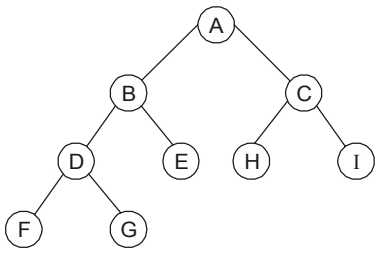
++*p*; *p* ptr+1

Printf("%s", ∗ – – ∗ ++ *p* + 3);

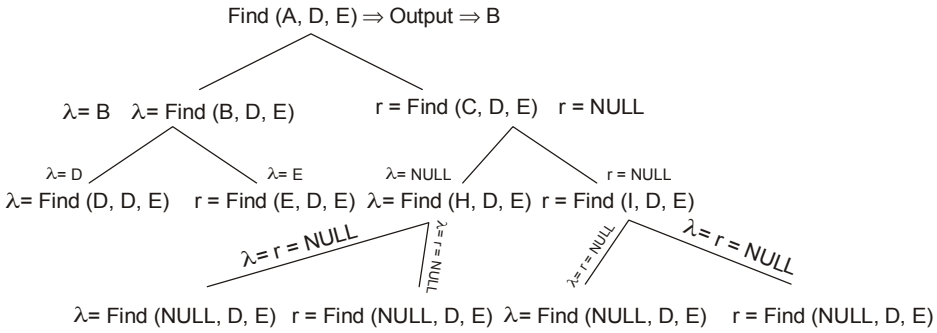
In printf statement the expression is evaluated ∗++*p* cause gets value (*s*+1) then now pre-decrement is executed and we get (*s*+1) – 1 = *s*. The indirection pointer now gets the value from the array of *s* and add 3 to the starting address. The string is printed starting from this position. Thus, the output is 'eeasy'.

12. (a)

Let's traverse the code on the given tree.



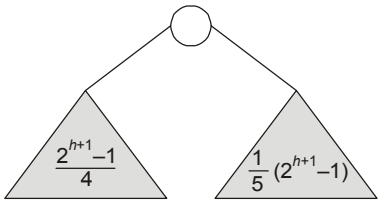
Let 'p' be D and 'q' be E.



Consider nodes 'D' and 'E', they have two ancestors node 'A' and node 'B'. 'B' is the lowest common ancestor and that is the output.

Hence, the value returned by the function is lowest common ancestor of two nodes.

13. (b)

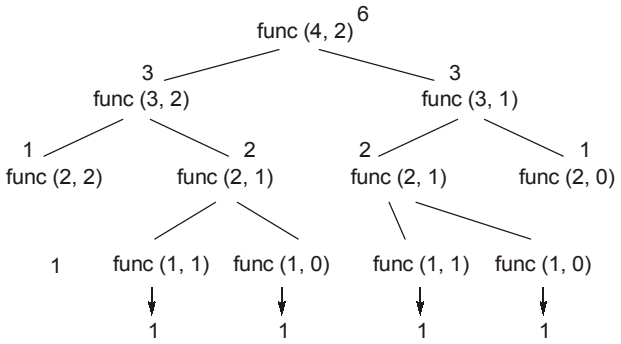


$$\text{Total} = \left(\frac{2^{h+1}-1}{4}\right) + \frac{1}{5}\left(2^{h+1}-1\right) + 1$$

$$= \frac{9}{20}\left(2^{h+1}-1\right)+1$$

14. (a)

Take m = 4 and n = 2



So, correct vaue of E is func (m – n, n) + func (m – 1, n – 1).

15. (d)

- **print 1( ):** x = 10 + 5 = 15; since the variable is of static storage class, hence it will retain its value between different function calls.
- **print 1( ):** x = 15 + 5 = 20; since it has retained its value 15.
- **print 2( ):** x is defined again inside the function and hence will print, $x = x + 5 = 10 + 5 = 15$. Again when the function will be called, x = 10 + 5 = 15. Here second time also x = 10 will be there because it is not initialized at the time of definition.

Hence output 15, 20, 15, 15.

16. (b)

By comparing string1 and string2, we are not comparing the actual data of the string, instead the starting address of both strings will be compared.

Hence, it will print "Two strings are unequal".

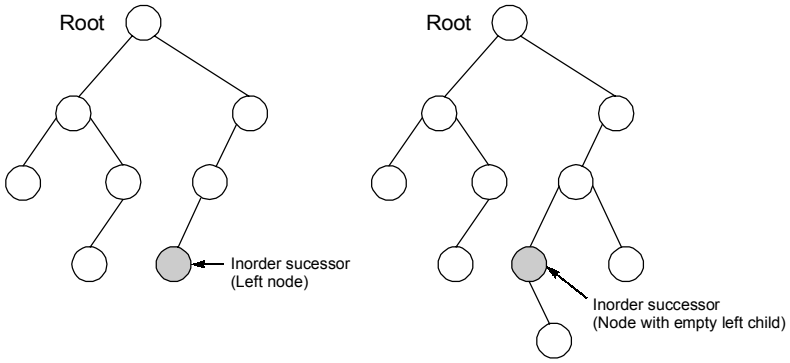
To compare the actual data of the strings,

if (∗ string1 == ∗ string2)

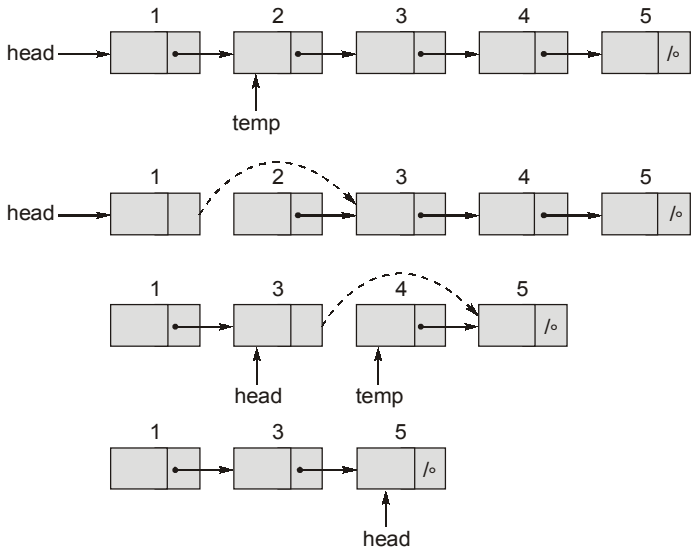
then will get the output: "Two strings are equal".

17. (d)

Successor of Root element is always the smallest element of the Right subtree. Because it will be the next largest element after the element to be deleted.

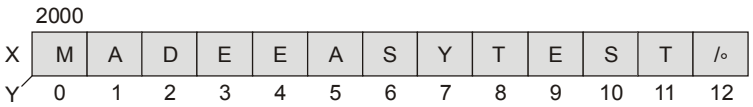


18. (c)



The above program deletes every even number node in the linked list (In particular second, fourth, sixth... soon nodes will be deleted)

19. (b)



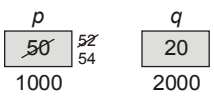
X = 2000
Y = 2000
y[7] = *y* (90 in ASCII)
y[10] = *s* (83 in ASCII)
= X + Y[7] – y[10]
= 2000 + 90 – 83 = 2007

Therefore it prints from the array starting at address; 2007 to the end i.e., "MADEEASYTEST".

20. (a)

If S_2 stack is empty and S_1 stack is not empty then we have to pop the element from stack S_1 and push that element into stack S_2 and return the stack S_2 which contain newly inserted element.

21. (b)

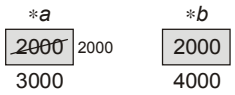


1st function call pass parameters as call by reference

a = *b* ⇒ Now a is also pointing to 2000 address.
**a*+ = 2 ⇒ **a* = **a* + 2
⇒ **a* = 20 + 2 = 22
a = *b* ⇒ Now a is also pointing to 2000 address.
2nd function call by reference

$a = b$ 'a' will now store b value which is 1000, already contain by 'a'.

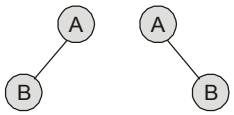
- $*a+ = 2;$
  $*a = *a + 2;$
  $*a = 50 + 2$
  $*a = 52$
- $a = b$
  'a' will now store b value which is 1000, already contain by 'a'.



$a = b \Rightarrow$ Now a is also pointing to 2000 address.
$*a+ = 2 \Rightarrow *a = *a + 2$
$\Rightarrow *a = 22 + 2 = 24$
$a = b \Rightarrow$ Now a is also pointing to 2000 address.
So, output will be 52 and 24.

22. (a)

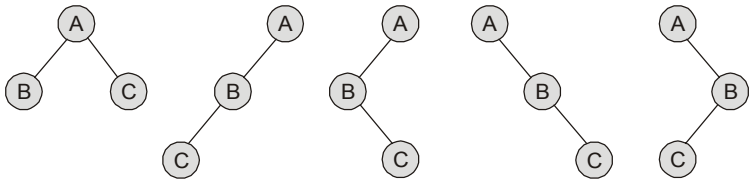
Consider 2 nodes "A", "B"
Consider preorder = A . B



only 2 Tree possible

So, $\frac{4!}{3! \times 2!} = \frac{4}{2} = 2$

Consider 3 nodes A. B. C
Consider preorder = A, B, C

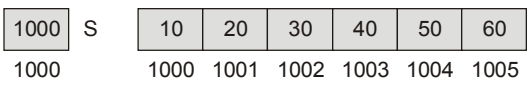


Only 5 trees present

So, $\frac{6!}{4! \times 3!} = \frac{6 \times 5}{3 \times 2} = 5$

So option (c) is true.

23. (d)



str 1006 $= (\text{int} *)(\&S + 1)$

$\Rightarrow = (\text{int} *)(\text{Base address of (S)} + 1 * (\text{size of (S)})$

$\Rightarrow$ = (1000 + 6 B)
$\Rightarrow$ = (1000 + 6)
$\Rightarrow$ = 1006

Now, *(S + 2) print 3rd element from start.
*(Str – 2) print *(1006 –2) = *(1004) element at address 1004 i.e. 50.

24. (a)

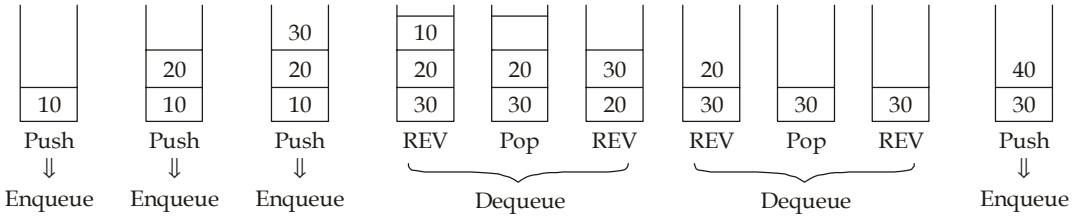
arr[ ] =	GATE%	CAT%	IES%	IAS%	PSU%	IFS%
	1000	1004	1008	1012	1016	1020

**ptr = arr $\Rightarrow$ **ptr = 1000;
*ptr1 = (ptr+ = size of (int)) [–2];
= (1000 + 4) [–2]
= [1000 + 4 × 4] [–2]
= [1016] [–2]
= [1015 – 2 × 4]
*ptr1 = [1008]
print(*ptr1) = IES

25. (d)

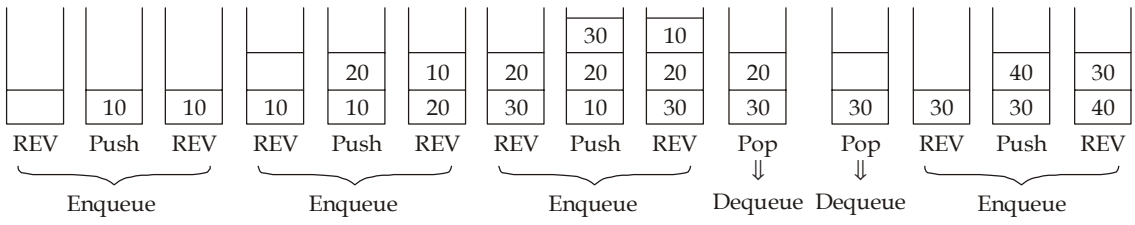
Enqueue: PUSH $\Rightarrow$ 1 operation
Dequeue: REVERSE, POP, REVERSE $\Rightarrow$ 3 operation
Example: Enqueue (10), Enqueue (20), Enqueue (30)
Dequeue, Dequeue, Enqueue (40)

Queue: ~~10~~ ~~20~~ 30 40



Enqueue: REV, PUSH, REV
Dequeue: POP
Example: Enqueue (10), Enqueue (20), Enqueue (30)
Dequeue, Dequeue, Enqueue (40)

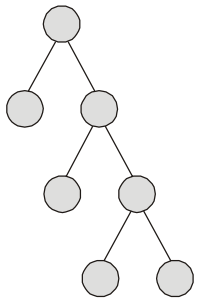
Queue: ~~10~~ ~~20~~ 30 40



So, either 1 Enqueue and 3 Dequeue or 3 Enqueue and 1 dequeue operation possible.

26. (d)

- Rotation operation in always preserves the inorder numbering so 1st is true.
- AVL tree doesnot guarantee that both left and right subtree has equal number of nodes, so statement is false.
- Consider



satisfying the property of statement 3, in this tree if element present is at last level the time complexity will be $c \times n/2 \simeq O(n)$. So S_3 is false.

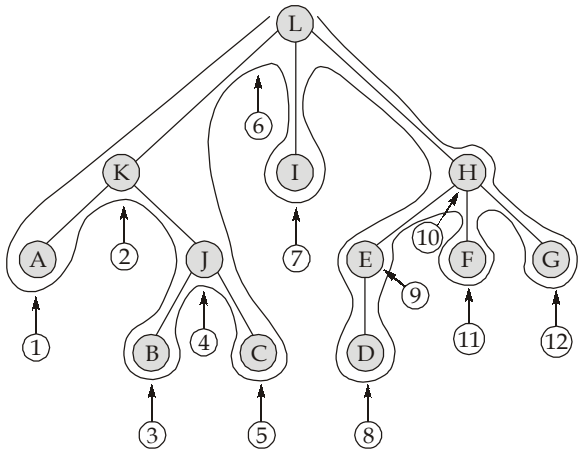
- Total nodes = 3 × internal nodes + 1
  = 3 × 20 + 1 = 61
  and 20 + 41 = 61
  (Leaf + internal = total) so S_4 is true.

27. (c)

Given function call is recursive.
Before calling any recursive call, it decrements length. So at leaf node recursive call will decrement by 1 even there exist no path.
∴ B is (len == –1)
Before traversing its child it decrements the length, whenever a length reaches to –1 and node is leaf then it implies there exist a path with given length.
∴ A is path($x \rightarrow$ left, len – 1)||path($x \rightarrow$ right, len – 1)
If one of the path returns non-zero then it recursively returns back.
So option (c) is correct.

28. (c)

- Inorder of tree is: Left, Root, Right (any other)



So, Inorder: AKBJCLIDEHFG

- Postorder of tree is Left, Right (any other), Root
  So, Post order is: ABCJKIDEFGHL

29. (c)

All the 3 functions check if a given number is a power of 2.

Function 1: Checks if $\log_2(n)$ of a number is an integer. If yes, it returns true, else it returns false. So function 1 checks if a given number is a power of 2.

Function 2: The key here is that, a number which is a power of 2 has the bit pattern 10* (1 followed by any number of zeroes). So at every step we keep checking if the number is even and keep dividing the number by 2 (right shift); if except for the most significant bit, a bit is found to be 1 (the number is odd at any point of time while right shifting), then the function 2 returns false. Else it returns true. So function 2 also checks if a given number is a power of 2.

Function 3: The observation is that, if a number n is power of 2, then $(n-1)$ becomes the 1's complement of n. Hence function 3 also checks if a given number is a power of 2.

30. (d)

Since P_1 returns the address of a variable which is declared locally, P_1 may cause problems.

P_2 will cause a problem because px doesn't have any address and is being dereferenced.

P_3 also will cause problems because even though malloc has been used to allocate the memory into the heap, free( ) has been called and returning that address is simply asking for trouble.

■■■■