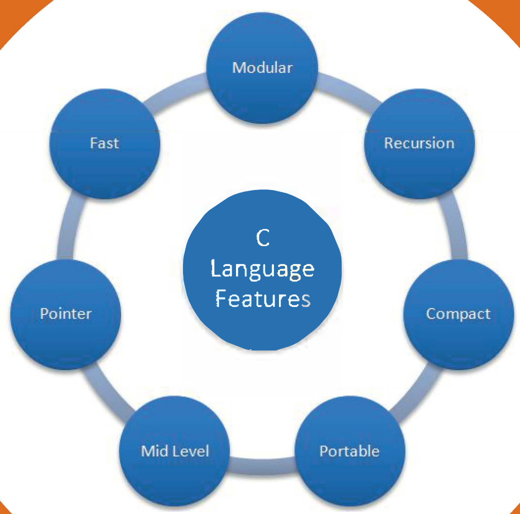
GATE | PSUs

COMPUTER SCIENCE & INFORMATION TECHNOLOGY

Programming Languages

(**Text Book :** Theory with worked out Examples and Practice Questions)



4 Programming Languages

(Solutions for Text Book Practice Questions)

01. Ans: (c)

Sol: Switch statement case A matches initially and all other cases are executed from there on as there is 'break' in cases.

$\therefore$ Output (c) $\rightarrow$ Choice A

Choice B No Choice

There is no break in between the case statements.

02. Ans: (a)

Sol: Initially matrix 'A' is empty and after performing the operations defined in the program then again matrix 'A' itself will be printed.

03. Ans: (b)

Sol: While loop will be terminated if $r < y$

By the time, when it reaches the condition $r < y$

The content in 'r' is $x - qy$

$\therefore r = x - qy \Rightarrow x = qy + r$

$\therefore x = (qy+r) \wedge r < y$

04. Ans: 10

Sol: j = ((((2 * 3) / 4) + (2.0/5)) + (8/5))

After evaluating above expression we have

$j = 2$

$k = -1$

When

$i = 0$, $i + k = -1 \rightarrow 1$ time printf statement executed

$i = 1$, $i + k = 0 \rightarrow 1$ time printf statement executed

$i = 2$, $i + k = 1 \rightarrow 3$ times printf statement executed

$i = 3$, $i + k = 2 \rightarrow 3$ times printf statement executed

$i = 4$, $i + k = 3 \rightarrow 2$ times printf statement executed

$\therefore$ Total 10 times printf statement executed.

05. Ans: (c)

Sol: In this, we are comparing (a>=b) && (c<b), if both are true then only we return b, that means we are finding middle number of a,b,c. Again by calling Trial function with different parameters, we are finding middle number of a, b, c.

06. Ans: (b)

Sol: When $p = 1, i = 1$

$$p = p * \frac{x}{i} \Rightarrow p = 1 * x ;$$

$p = x ;$

$s = s + p ;$

$= 1 + x ;$

When $p = 1, i = 2$

$$p = x * \frac{x}{2} = \frac{x^2}{2}$$

$$s = 1 + x + \frac{x^2}{2}$$

If we continue we get

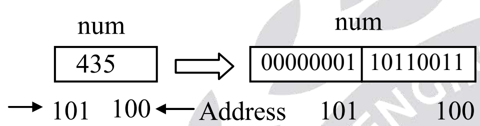
$$s = 1 + \frac{x}{1} + \frac{x^2}{2!} + \frac{x^3}{3!} \ldots\ldots\ldots$$
$$= e^x$$

07. Ans: (d)

Sol: Function defined later to the call and not defined in the program requires prototype.

08. Ans: 9

Sol:



The expression num>>=1; interprets that the content in variable num is shifted one bit right for every while loop. [Note that a bitwise right shift operator is same as integer division by 2.] So after "9" times of while loop, the content in num is zero.

09. Ans: (d)

Sol: Since function prototype is void f(int, short) i.e., f is accepting, arguments int, short and its return type is void. So f(i, *p) is correct answer.

10. Ans: (d)

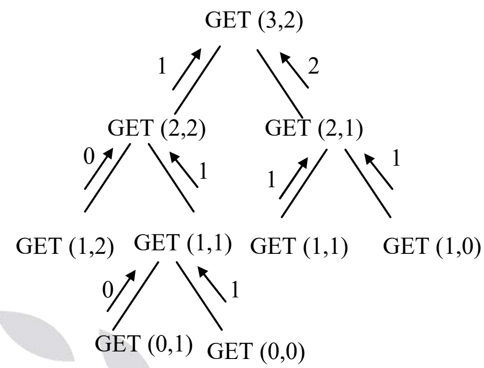
Sol: If b! = a we get maximum element of an integer

11. Ans: (c)

Sol: $X^Y = res * a^b$

12. Ans: 3

Sol:



13. Ans: (c)

Sol: Parameter is passed by reference

14. Ans: (c)

15. Ans: (b)

Sol: When the function call occurs, then the statements followed by function calls will be stored into stack in the form of activation record. So number of activation records depends on number of function calls.

16. Ans: (d)

Sol: The function foo is recursive function when we call foo (a, sum) = foo(2048, 0)

k = 2048% 10 = foo(204, 8)

foo (204,8)

k = 204% 10 = 4

foo (20, 4)

k = 20% 10 = 0

foo (2, 0)

k = 2% 10 = 2

j = 2048/10

10 = 204
j = 204/10
10 = 20
j = 2/10 = 0
sum = 0 + 8 = 8
sum = 8 + 4 =12
sum = 12 + 2= 14
foo (0,14) function will be terminated and value of k will print in stack way i.e.2, 0, 4 8 and sum = 0
Since sum is local variable in the main function so the print sequence is
2, 0, 4, 8, 0.

17. Ans: (d)

Sol: Here we are using the '=' operator which has high priority than '!=' operator.
So (c = getchar()) has to be in brackets and after reversing the string we use function putchar(c) for printing the character.

18. Ans: (b)

Sol: foo(345, 10)
= 345 % 10 = 5, 345/10 = 34
foo(34,10)
= 34%10 = 4, 34/10 = 3
foo(3,10)
= 3%10 = 0, 3/10 = 0
foo(0,10)

Stack	
3+(0)	3+0 = 3
4+(3)	4+3 = 7
5+(7)	5+7 = 12 → therefore output is 12

19. Ans: (d)

Sol: foo (513, 2)
= 513 % 2 = 1, 513/2 = 256
foo(256,2)
= 256%2 = 0, 256/2 = 128
foo(128,2)
= 128%2 = 0, 128/2 =64
foo(64,2)
= 64%2 = 0, 64/2 = 32
foo(32,2)
= 32%2 = 0, 32/2 = 16
foo(16,2)
= 16%2 = 0, 16/2 = 8
foo(8,2)
= 8%2 = 0, 8/2 = 4
foo(4,2)
= 4%2 = 0, 4/2 = 2
foo(2,2)
= 2%2 = 0, 2/2 = 1
foo(1,2)
= 1%2 = 1, 1/2 = 0
foo(0,2)

Stack
1+(0)
1+(0)
1+(0)
1+(0)
1+(0)
1+(0)
1+(0)
1+(0)
1+(0)
1+(1)

therefore output is 2 →

20. Ans: 51

Sol: $x = x + \sum_{K=1}^{4} f(k) \times f(5-k)$

f(1) = 1, f(2) = 2, f(3) = 5, f(4) = 15

$\therefore$ x = x+ [f(1) * f(4) + f(2) * f(3) + f(3) * f(2) + f(4) * f(1)]

$\therefore$ x = 51

21. Ans: (a)

Sol:

i	0	1	2	3	4
j	0	1	3	6	10

22. Ans: (c)

Sol: n is incremented by one in each iteration.

23. Ans: (d)

Sol: n $\boxed{5}$ r $\boxed{0}$

return f(n-2)+2 = 5-2+2 = 5

r = n = 5

( )+2

(1)+5	1+5 = 6
(6)+5	6+5 = 11
(11)+5	11+5 = 16
(16)+2	16+2 = 18 → therefore output is 18

return f(n–2)+r = 5 – 2 + 0 = 3

return f(n–1)+r = 3 – 1 + 0 = 2

return f(n–1)+r = 2 – 1 + 0 = 1

24. Ans: (c)

Sol: If the variables are static then, it is persisting previous state value from the destruction of various function calls.

The variable 'a' in prtFun() is static, i.e its life time is global and hence retains its value always, meaning history sensitive.

25. Ans: (d)

Sol: If the variables are auto, these variables will be reinitialized in every function call.

Now the variables are all auto storage class. Their lifetime is local.

26. Ans: (d)

Sol: For every function call, the auto variable j is recreated and reinitialized. If we take j = 50, then every time, if condition is true, so we have to call f(i) every time in that case the statements reference are stored into the stack, and stack continuously growing, so after some extent, stack overflow error occurs.

27. Ans: 230

Sol: x = x + f1 ( ) + f2 ( ) + f3( ) + f2 ( )

f1 ( ) returns 26

f2 ( ) returns 51

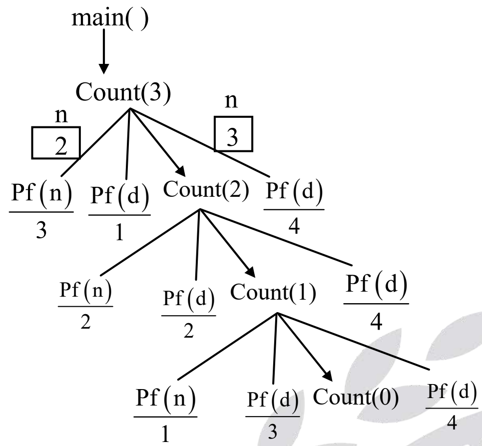
f3 ( ) returns 100

f2 ( ) returns 52

x = 1 + 26 + 51 + 100 + 52 = 230

28. Ans: (a)

Sol:



Output = 3, 1, 2, 2, 1, 3, 4, 4, 4

29. Ans: (d)

30. Ans: (a)

Sol: It is an array of pointers and each pointer is pointing to structure

31. Ans: (c)

Sol: ‘P1’creates dangling pointer problem.
‘P2’creates uninitialized pointer problem

32.

Sol: (a) 332 332 1
(b) $(2^n – 1)$

33. Ans: (a)

Sol: In main () function, we are passing address of x = 5, to the function P( ) and in P( ), we are passing x = 7 to Q( ). So print(z) displays output as 12 and print(x) in P( ), will print 7, and print(x) in main will print 6.

34. Ans: (a)

Sol: Since B[10][10] represents two-dimensional array so, B[1] represents address we can not write it as left hand side of assignment operators, however, remaining I, II, IV are representing values, so we can write them left hand side of assignment operators

35. Ans: (d)

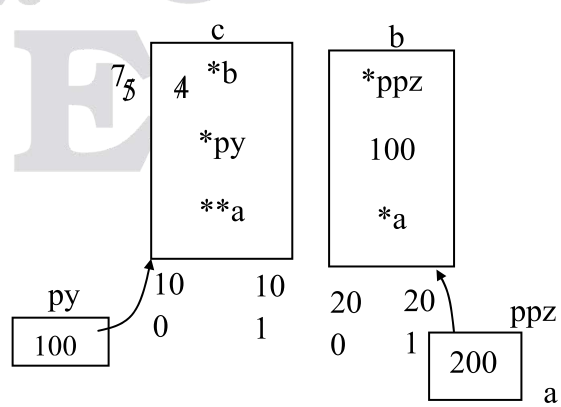
Sol: Since first character in the array p[20], contains null character, so while compiler executing the array p[20], it reads first character (i.e null character) and assumes that it is the end of the string, so no output printed.

36. Ans: (c)

Sol: int(*f)(int*) ;
Syntax pointer to function is for declaration of
return_type (*ptr variable)
(List of arguments);

37. Ans: (b)

Sol:



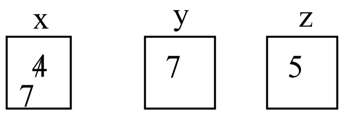
f(c, b, a)

↓

int f(int x, int *py, int **ppz)

↓

(4, 100, 200)



return x+y+z

∴ x + y + z = 7+7+5 = 19.

Output is (b)

38. Ans: (d)

Sol: In this function int *p, int *q these are two pointer variables (global function) and f(&i,&j) are the local values of the pointer variables. First, p is storing 200 address and q is storing 300 address (we are assuming the address). &i is pointed to p address and &j is pointed to q address.

∴ p = q then address of p&q both are storing at same address then i = 0, j = 2 and *p = 2. after *p, *q are both storing at same location that's why the values are same as printed. First *p = 0 after *p = 2.

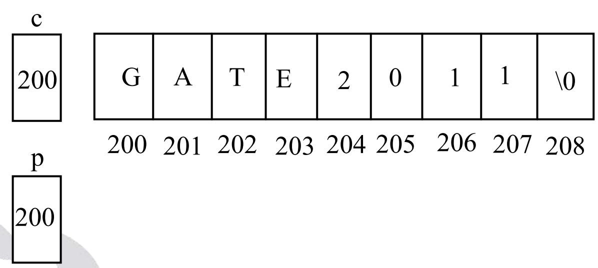
Therefore output is 0 2.

39. Ans: (c)

Sol: The ASCII values of p[3], which is E & p[1] which is 'A' gets subtracted i.e the difference from 'A' to 'E' is 4

Therefore p+4 is '2004' assuming 'p' to be 2000;

Therefore o/p is 2011



ASCII value of A = 65

ASCII value of C = 69

printf("%s", p + p[3] – p[1])

printf("%s", 200 + 69 – 65)

printf("%s", 204)

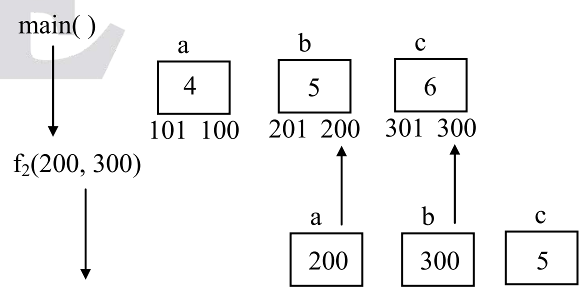
Output = 2011

40. Ans: (d)

Sol: Scanf function reads input from the user and stores it in variable 'i'. On execution, the value printed is '5' more than the integer value entered.

41. Ans: –5

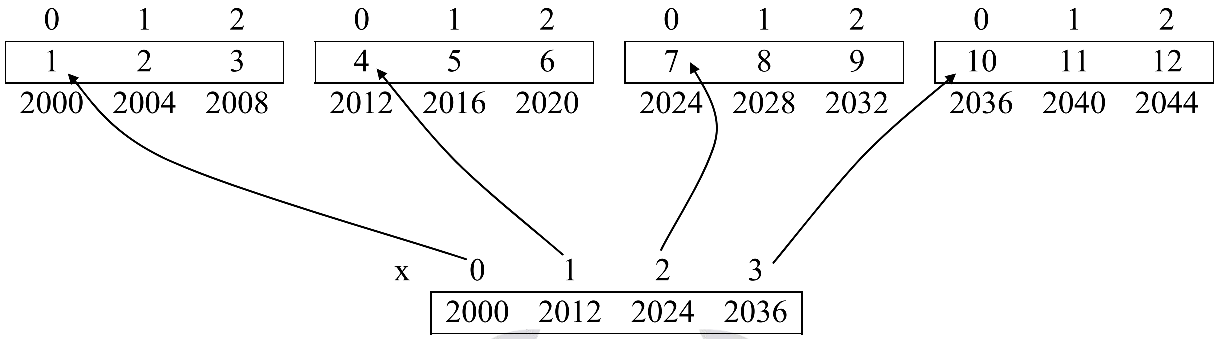
Sol:



∴ printf("%d", c –b –a) prints output –5.

42. Ans: (a)

Sol:



x+3 = 2000 + (3+size of each one dimensional array)

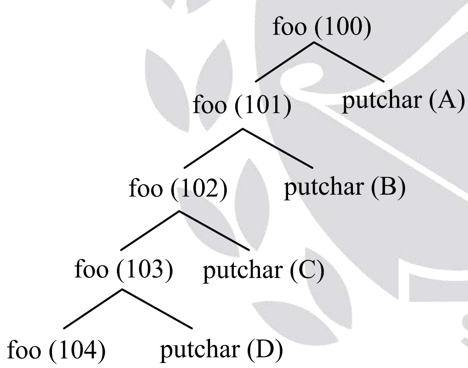
= 2000 + (3 × 12) = 2036

∴ printf("%d%u%u", x+3, *(x+3), *(x+2)+3);

= 2036, 2036, 2036

43. Ans: (d)

Sol: Assume that base address of string constant "ABCD EFGH" is 100.



∴ Output: DCBA

44. Ans: 15

Sol: When we call stkFunc(–1,10), the variable size is initialized to 10.

When we call stkFunc(0,5), the array contains element 5.

When we call stkFunc(0,10), the array contains elements 5, 10

The function stkFunc(1,0), returns 10 and the function stkFunc(1, 0) returns 5

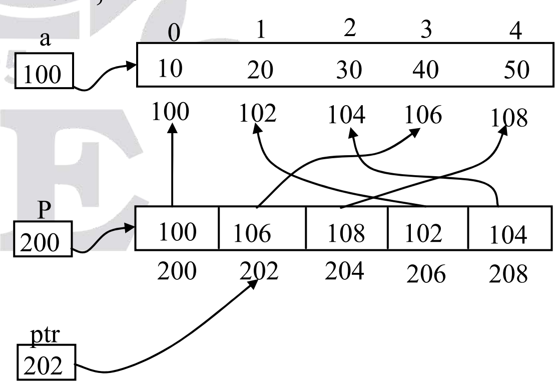
∴ Print statement will print result 15

45. Ans: (c)

Sol: By using pointer 'p' the content of array is updated to 1204. i.e p = s1 + 2, will point to the location of 3rd element of array s1 and that element is replaced by '0'.

46. Ans: 1, 40

Sol:



ptr – p = 202 – 200 = 2 (i.e., 1 element)

ptr = * 106 = 40

∴ Output = 1, 40

47. Ans: 2016

Sol: Whatever modifications are performed in mystery( ) function, those modifications are not reflected in main( ) function so it will print 2016.

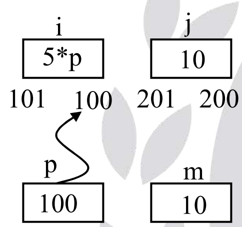
48. Ans: 30

Sol: m = m + 5; // m = 15

*p = *p + m; // * p = 5 + 15

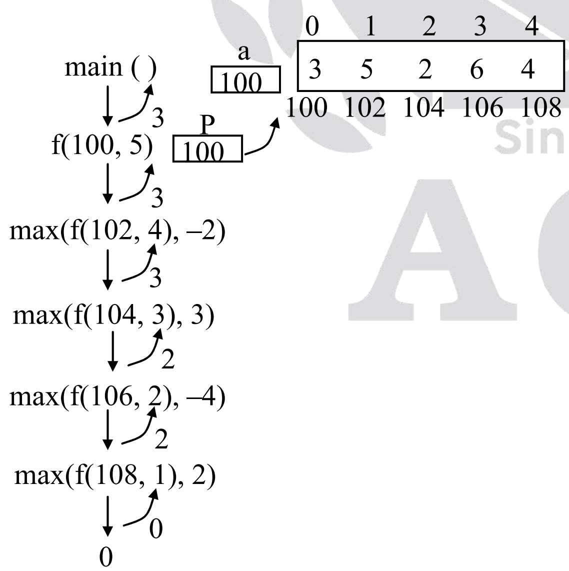
* p = 20 (i.e., i = 20)

i + j = 20 + 10 = 30



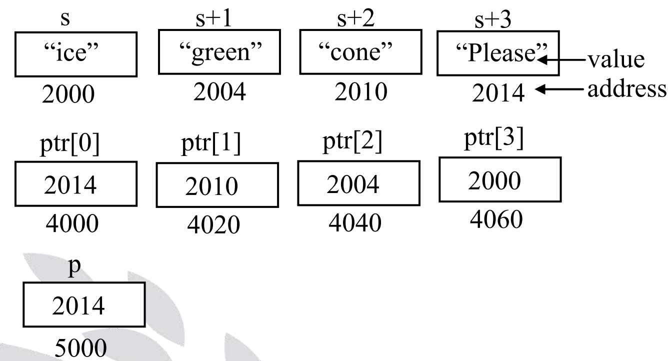
49. Ans: 3

Sol:



50. Ans: (a)

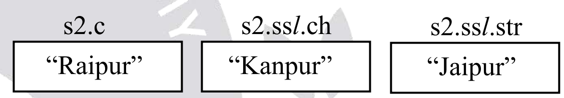
Sol: Char ***p = ptr



It prints "cone", "ase", "reen"

51. Ans: (b)

Sol:



1st printf prints Raipur Jaipur

2nd printf prints aipur aipur

because s2.c begins from R

++s2.c begins from a

52. Ans: (b)

Sol: a[0] = {"Nagpur", 1, a + 1}

a[1]= {"Raipur", 2, a + 2}

a[2] = {"Kanpur", 3, a}

a[0].z → prints Nagpur

ptr.z → prints Nagpur

a[2].p ⇒ a[2].p = a

a[2].p→ z ⇒ a → z prints Nagpur

53. Ans: (d)

Sol:
- ptr → z points to a[0].z
  ++(ptr →z) points to a[1].z which prints agpur.
- a[(++ptr) → i].z = a[a[1].i].z
  = a[2].z which prints kanpur.
  Because *ptr = a
  ++ptr becomes a[1]
- ptr → p ⇒ a + 1
  ptr → p →i ⇒ a[1] → i = 2
  – – (ptr → p → i) =1
  a[1].z is Kanpur

54. Ans: (b)

Sol: Struct test *p = st
p = p +1
p points to st[0]
p = p + 1 points to st[1]
printf("%s", ++p → c) prints "etter"
p → c points to better
++ p → c points to etter
*++p → c ⇒ prints second character of Jungle, 'u'
p[0]. i ⇒ prints 6 because p points to st[2].
p → c ⇒ prints ungle.

55. Ans: (a)

Sol: * X[0] = a1
* X [1] = a2
* X[2] = a3
Print (int *a[]) implies *a[0] = *X[0]
*a[1] = *X[1]
*a[2] = *X[2]
a[0][2] = a1[2] = 8
*a[2] = a3[0] = – 12
*++a[0] = a[0][1] = a1[1] = 7
*(++a)[0] = *a[1] = a2[0] = 23
a[–1][+1] = a1[2] = 8

56. Ans: (a)

57. Ans: (b)

Sol: x = 15
fun(5, &x) ⇒ fun(n, *fp)
t = fun(4, fp), f = t + * fp, *fp =t
fun(4,fp) ⇒ t = fun(3, fp),
f = t + *fp, *fp = t
fun(3, fp) ⇒ t = fun(2, fp),
f = t + *fp, *fp = t
fun(2, fp) ⇒ t = fun(1, fp),
f = t + *fp, *fp = t
fun(1, fp) ⇒ *fp = 1
⇒ x =1 return 1 ⇒ t =1
fun(2, fp) ⇒ t =1,
f = 1 + * fp = 1 + 1 = 2,
*fp =1 return f(2)
fun(3, fp) ⇒ t = 2,
f = 2 + * fp = 2 + 1 = 3,
*fp = 2 return 3
fun(4, fp) ⇒ t = 3,
f = 3 +2 = 5, *fp = 3,
return 5
fun(5, fp) ⇒ t = 5,
f = 5 + 3 = 8, *fp = 5,
return 8

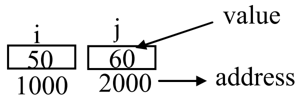
58. Ans: (d)

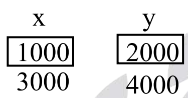
Sol: Call by value: No change in j value

There is a change in i value because i is global.

Call by reference:

i = 50, j = 60





f(&i, &j) ⇒

In procedure f ( ) i = 100 ⇒ $\underset{1000}{\overset{i}{\boxed{100}}}$

x =10 ⇒ $\underset{1000}{\overset{i}{\boxed{10}}}$

y = y + i = 60 + 10 = 70

59. Ans: (d)

Sol: If we call swap(x, y) then there is no interchange in the value of x and y because the parameters are passed by value. There is interchange in formal parameters a and b but not interchange in actual parameters x and y because the scope of ‘a’ and ‘b’ lie within the function but not in the main program.

60. Ans: (b)

Sol: 9 * 9 * 9 * 9 * 1 = 6561

61.

Sol: (i).Call-by-value: 1,100

(ii).Call-by-Reference: 2, 7
because ‘a’ refers to ‘x’ and ‘c’ refers to ‘z’

62.

Sol: (a) (i). Call-by value prints 30

(ii) 5 times

(b) Call-by-Reference prints 110

63.

Sol: (i). Call-by Value: 2

(ii). Call-by-Reference: 10

64. Ans: (b)

Sol: In called function func1, x refer to the value 3, y and z refers to 10 so the output is 31, 3.

65. Ans: (a)

Sol: Under static scoping the reference to free variable is in the environment of the immediate next outer Block (statically/ lexically) therefore the answer is (3, 6)

66. Ans: (b)

Sol: Under Dynamic scoping, the reference to free variable is at point of invocation in reverse order, therefore the answer is (6, 7)

67.

Sol: The referencing environment in procedure ‘s’ is that of ‘s’ and ‘P’

The referencing environment in procedure ‘q’ is that of ‘q’, ‘s’ and ‘P’

The referencing environment in procedure ‘r’ is that of ‘r’ , q, s and P.

68.

Sol: (i). Static Scoping : 5, 10

(ii). Dynamic Scoping : 1, 2

69.

Sol: i) 2, 2, 2

In static scope the referencing environment of free variable is in the next immediate outer block

ii) 2, 5, 2

In dynamic scope the referencing environment of free variable is at point of invocation.

70. Ans: (c)

Sol: In dynamic scope, the reference to the free variable is at a point of invocation in reverse order.

71.

Sol: (a) 12, 7, 10, 5 with static scoping and call-by-value

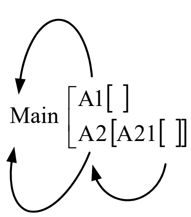
(b) 14, 14, 10, 10 with Dynamic scope and call-by-reference

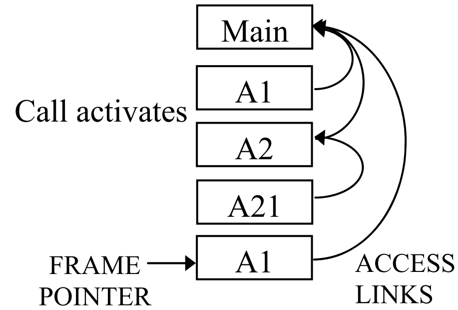
72. Ans: (d)

Sol: Output is 4, as 'x' refer to n.

73. Ans: (d)

Sol: Static logical scoping used a clean links are shown to statistically (Textually) enclosing blocks.

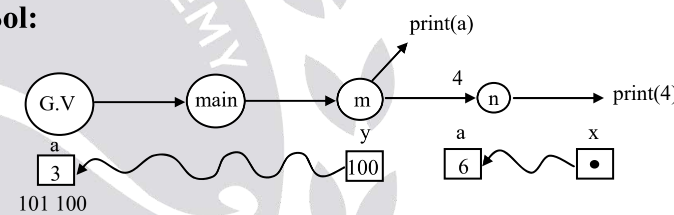




Lexical scoping refers to static scoping. The referencing environments of the statements are local scope plus parental scopes.

74. Ans: (d)

Sol:



75. Ans: (b)

Sol: Recursion requires stack, where as dynamic data structure required heap.

76. Ans: (c)

Sol: Data structures that are allocated space during run-time is done from the Heap portion.

77. Ans: (c) & (d)

Sol: Only alphanumeric characters and few special characters like '_' are allowed in variable name C. The special character @ is not allowed.

78. Ans: (a), (b) & (d)

Sol: Consider the following expressions:

if(a = = 7) ⇒ if(1) ⇒ correct.

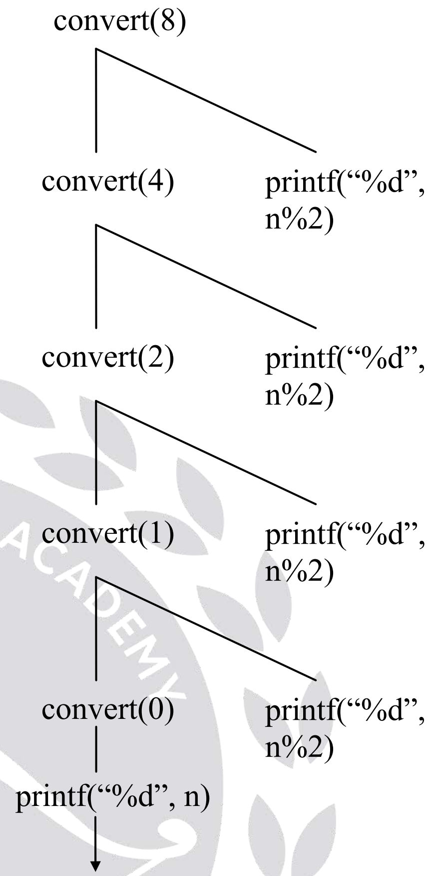
if(7 = = a) ⇒ if(1) ⇒ correct.

if(a = 7) ⇒ if(7) ⇒ correct (7 is a non-zero value).

if(7 = a) ⇒ incorrect, because value of a cannot assign in 7, L value required error will occur.

79. Ans: (a), (b) & (d)

Sol: Assume that input n = 8



It will be continued recursively and it will not print anything and not terminate