Technical Papers C Questions: C Questions Part V

Examrace Placement Series prepares you for the toughest placement exams to top companies.

1. main () { int i = 0; j = 0; if (i && j + +) printf (“%d. %d” i + +, j “); printf (” %d. %d, i, j); } Answer: 0. 0 Explanation: The value of i is 0. Since this information is enough to determine the truth value of the boolean expression. So the statement following the if statement is not executed. The values of i and j remain unchanged and get printed.

2. main () { int i; i = abc (); printf (“%d” i); } abc () { _AX = 1000; } Answer: 1000 Explanation: Normally the return value from the function is through the information from the accumulator. Here _AH is the pseudo global variable denoting the accumulator. Hence, the value of the accumulator is set 1000 so the function returns value 1000.

3. int i; main () { int t; for (t = 4; scanf (“%d” &i) -t; printf (“%d\n” i)) printf (“%d--” t--); }//If the inputs are 0, 1, 2, 3 find the o/p Answer: 4--0 3--1 2--2 Explanation: Let us assume some x = scanf (“%d” &i) -t the values during execution will be, t i × 4 0 – 4 3 1 – 2 2 2 0

4. main () { int a = 0; int b = 20; char x = 1; char y = 10; if (a, b, x, y) printf (“hello”); } Answer: Hello Explanation: The comma operator has associativity from left to right. Only the rightmost value is returned and the other values are evaluated and ignored. Thus the value of last variable y is returned to check in if. Since it is a non zero value if becomes true so, “hello” will be printed.

5. main () { unsigned int i; for (i = 1; i>-2; i--) printf (“c aptitude”); } Explanation: i is an unsigned integer. It is compared with a signed value. Since the both types doesn't match, signed is promoted to unsigned value. The unsigned equivalent of-2 is a huge value so condition becomes false and control comes out of the loop.

6. In the following pgm add a stmt in the function fun such that the address of ‘a’ gets stored in ‘j’ Main () { int * j; void fun (int * *); fun (&j); } void fun (int * * k) { int a = 0;/* add a stmt here */} Answer: * k = &a Explanation: The argument of the function is a pointer to a pointer.

7. What are the following notations of defining functions known as?

    a. int abc (int a, float b) {/* some code */}

    b. int abc (a, b) int a; float b; {/* some code */}

    Answer: ANSI C notation and Kernighan & Ritche notation

8. main () { char * p; p = “%d\n” p + +; p + +; printf (p – 2, 300); } Answer: 300 Explanation: The pointer points to % since it is incremented twice and again decremented by 2, it points to ‘%d\n’ and 300 is printed.

9. main () { char a[100]; a[0] = ‘a’ a[1] = ‘b’ a[2] = ‘c’ a[4] = ‘d’ abc (a); } abc (char a[]) { a + +; printf (“%c” * a); a + +; printf (“%c” * a); } Explanation: The base address is modified only in function and as a result a points to ‘b’ then after incrementing to ‘c’ so bc will be printed.

10. func (a, b) int a, b; { return (a = (a = = b)); } main () { int process (), func (); printf (“The value of process is %d! \n” process (func, 3, 6)); } process (pf, val1, val2) int (* pf) (); int val1, val2; { return ((* pf) (val1, val2)); } Answer: The value if process is 0! Explanation: The function ‘process’ has 3 parameters-1, a pointer to another function 2 and 3, integers. When this function is invoked from main, the following substitutions for formal parameters take place: Func for pf, 3 for val1 and 6 for val2. This function returns the result of the operation performed by the function ‘func’ The function func has two integer parameters. The formal parameters are substituted as 3 for a and 6 for b. Since 3 is not equal to 6, a = = b returns 0. Therefore the function returns 0 which in turn is returned by the function ‘process’

11. void main () { static int i = 5; if (--i) { main (); printf (“%d” i); } } Answer: 0 0 0 0 Explanation: The variable “I” is declared as static, hence memory for I will be allocated for only once, as it encounters the statement. The function main () will be called recursively unless I becomes equal to 0, and since main () is recursively called, so the value of static I i.e.. 0 will be printed every time the control is returned.

12. void main () { int k = ret (sizeof (float)); printf (“\n here value is %d” + + k); } int ret (int ret) { ret + = 2.5; return (ret); } Answer: Here value is 7 Explanation: The int ret (int ret), i.e.. the function name and the argument name can be the same. Firstly, the function ret () is called in which the sizeof (float) i.e.. 4 is passed, after the first expression the value in ret will be 6, as ret is integer hence the value stored in ret will have implicit type conversion from float to int. The ret is returned in main () it is printed after and preincrement.

13. void main () { char a[] = “12345\0” int i = strlen (a); printf (“here in 3 %d\n” + + i); } Answer: Here in 3 6 Explanation: The char array ‘a’ will hold the initialized string, whose length will be counted from 0 till the null character. Hence the ‘I’ will hold the value equal to 5, after the pre-increment in the printf statement, the 6 will be printed.

14. void main () { unsigned giveit = -1; int gotit; printf (“%u” + + giveit); printf (“%u \n” gotit = --giveit); } Answer: 0 65535 Explanation:

15. void main () { int i; char a[] = “\0” if (printf (“%s\n” a)) printf (“Ok here \n”); else printf (“Forget it\n”); } Answer: Ok here Explanation: Printf will return how many characters does it print. Hence printing a null character returns 1 which makes the if statement true, thus “Ok here” is printed.

16. Some question on compiler error Answer: Compiler Error. We cannot apply indirection on type void *. Explanation: Void pointer is a generic pointer type. No pointer arithmetic can be done on it. Void pointers are normally used for, Passing generic pointers to functions and returning such pointers. As a intermediate pointer type. Used when the exact pointer type will be known at a later point of time.

17. void main () { int i = i + +, j = j + +, k = k + +; printf (%d%d%d, i, j, k); } Answer: Garbage values. Explanation: An identifier is available to use in program code from the point of its declaration. So expressions such as i = i + + are valid statements. The i, j and k are automatic variables and so they contain some garbage value. Garbage in is garbage out (GIGO).

18. void main () { static int i = i + +, j = j + +, k = k + +; printf (i = %d j = %d k = %d, i, j, k); } Answer: i = 1 j = 1 k = 1 Explanation: Since static variables are initialized to zero by default.

19. void main () { while (1) { if (printf (“%d” printf (“%d”))) break; else continue; } } Answer: Garbage values Explanation: The inner printf executes first to print some garbage value. The printf returns no of characters printed and this value also cannot be predicted. Still the outer printf prints something and so returns a non-zero value. So it encounters the break statement and comes out of the while statement.

20. main () { unsigned int i = 10; while (i → = 0) printf (“%u” i); } Answer: 10 9 8 7 6 5 4 3 2 1 0 65535 65534. Explanation: Since i is an unsigned integer it can never become negative. So the expression i → = 0 will always be true, leading to an infinite loop.

21. #include main () { int x, y = 2, z, a; if (x = y%2) z = 2; a = 2; printf (“%d %d” z, x); } Answer: Garbage-value 0 Explanation: The value of y%2 is 0. This value is assigned to x. The condition reduces to if (x) or in other words if (0) and so z goes uninitialized. Thumb Rule: Check all control paths to write bug free code.

22. main () { int a[10]; printf (“%d” * a + 1- * a + 3); } Answer: 4 Explanation: * a and- * a cancels out. The result is as simple as 1 + 3 = 4!

23. #define prod (a, b) a * b main () { int x = 3, y = 4; printf (“%d” prod (x + 2, y – 1)); } Answer: 10 Explanation: The macro expands and evaluates to as: x + 2 * y – 1 ⇒ x + (2 * y) -1 ⇒ 10

24. main () { unsigned int i = 65000; while (i + +! = 0); printf (“%d” i); } Answer: 1 Explanation: Note the semicolon after the while statement. When the value of i becomes 0 it comes out of while loop. Due to post-increment on i the value of i while printing is 1.

25. main () { int i = 0; while (+ (+ i--)! = 0) i- = i + +; printf (“%d” i); } Answer: 1 Explanation: Unary + is the only dummy operator in C. So it has no effect on the expression and now the while loop is, while (i--! = 0) which is false and so breaks out of while loop. The value 1 is printed due to the post-decrement operator.

26. main () { float f = 5, g = 10; enum { i = 10, j = 20, k = 50}; printf (“%d\n” + + k); printf (“%f\n” f<<2); printf (“%lf\n” f%g); printf (“%lf\n” fmod (f, g)); } Answer: Line no 5: Error: Lvalue required Line no 6: Cannot apply leftshift to float Line no 7: Cannot apply mod to float Explanation: Enumeration constants cannot be modified, so you cannot apply + +. Bit-wise operators and % operators cannot be applied on float values. Fmod () is to find the modulus values for floats as % operator is for ints.

27. main () { int i = 10; void pascal f (int, int, int); f (i + +, i + +, i + +); printf (“%d” i); }

void pascal f (integer: i, integer: j, integer: k) { write (i, j, k); }

Answer: Compiler error: Unknown type integer Compiler error: Undeclared function write