|  |
| --- |
| 1.#include<stdio.h>  void fun(int\*, int\*);  int main()  {  int i=5, j=2;  fun(&i, &j);  printf("%d, %d", i, j);  return 0;  }  void fun(int \*i, int \*j)  {  \*i = \*i\*\*i;  \*j = \*j\*\*j;  } |
| |  |  |  |  | | --- | --- | --- | --- | | [A.](javascript: void 0;) | 5, 2 | [B.](javascript: void 0;) | 10, 4 | | [C.](javascript: void 0;) | 2, 5 | [D.](javascript: void 0;) | 25, 4 |  |  | | --- | | What does the following declaration mean? *int (\*ptr)[10];* | | |  |  | | --- | --- | | [A.](javascript: void 0;) | *ptr* is array of pointers to 10 integers | | [B.](javascript: void 0;) | *ptr* is a pointer to an array of 10 integers | | [C.](javascript: void 0;) | *ptr* is an array of 10 integers | | [D.](javascript: void 0;) | *ptr* is an pointer to array | |   Ans-B   |  | | --- | | In C, if you pass an array as an argument to a function, what actually gets passed? | | |  |  | | --- | --- | | [A.](javascript: void 0;) | Value of elements in array | | [B.](javascript: void 0;) | First element of the array | | [C.](javascript: void 0;) | Base address of the array | | [D.](javascript: void 0;) | Address of the last element of array | |   Ans-c   |  | | --- | |  | | |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | | [C.](javascript: void 0;) |  |  |  | |   Ans-C   |  | | --- | | What will be the output of the program ?  #include<stdio.h>  int main()  {  char p[] = "%d\n";  p[1] = 'c';  printf(p, 65);  return 0;  } | | |  |  |  |  | | --- | --- | --- | --- | | [A.](javascript: void 0;) | A | [B.](javascript: void 0;) | a | | [C.](javascript: void 0;) | c | [D.](javascript: void 0;) | 65 | |   Ans-A   |  | | --- | | What will be the output of the program ?  #include<stdio.h>  int main()  {  printf(5+"Good Morning\n");  return 0;  } | | |  |  |  |  | | --- | --- | --- | --- | | [A.](javascript: void 0;) | Good Morning | [B.](javascript: void 0;) | Good | | [C.](javascript: void 0;) | M | [D.](javascript: void 0;) | Morning | |   Ans-D   |  |  | | --- | --- | |  | What will be the output of the program ?  #include<stdio.h>  #include<string.h>  int main()  {  static char s[] = "Hello!";  printf("%d\n", \*(s+strlen(s)));  return 0;  } | | |  |  |  |  | | --- | --- | --- | --- | | [A.](javascript: void 0;) | 8 | [B.](javascript: void 0;) | 0 | | [C.](javascript: void 0;) | 16 | [D.](javascript: void 0;) | Error | |   Ans-B |

#include <stdio.h>

int main(void)

{

char p;

char buf[10] = {1, 2, 3, 4, 5, 6, 9, 8};

p = (buf + 1)[5];

printf("%d\n", p);

return 0;

}

What is the output of this program?

(a)  5  
(b)  6  
(c)  9  
(d)  none of the above

[Hide answer](javascript:void(hide(13)))

Answer: **(c)**

Remember that x[i] is equivalent to \*(x + i), so (buf + 1)[5] is \*(buf + 1 + 5), or buf[6].

#include <stdio.h>

int main(void)

{

int a, b, c, d;

a = 3;

b = 5;

c = a, b;

d = (a, b);

printf("c=%d ", c);

printf("d=%d\n", d);

return 0;

}

What is the output of this program?

(a)  c=3 d=3  
(b)  c=5 d=3  
(c)  c=3 d=5  
(d)  c=5 d=5

[Hide answer](javascript:void(hide(6)))

Answer: **(c)**

The comma operator evaluates both of its operands and produces the value of the second.  It also has lower precedence than assignment.  Hence c = a, b is equivalent to c = a, while d = (a, b) is equivalent to d = b.

#include <stdio.h>

int main(void)

{

int i = 3;

int j;

j = sizeof(++i + ++i);

printf("i=%d j=%d\n", i, j);

return 0;

}

What is the output of this program on an implementation where int occupies 2 bytes?

(a)  i=4 j=2  
(b)  i=3 j=2  
(c)  i=5 j=2  
(d)  the behavior is undefined

[Hide answer](javascript:void(hide(9)))

Answer: **(b)**

Evaluating ++i + ++i would produce undefined behavior, but the operand of sizeof is not evaluated, so i remains 3 throughout the program.  The type of the expression (int) is deduced at compile time, and the size of this type (2) is assigned to j.