Array
(pointer to array, array to pointer)

By : Chandrashekhar and Chaitanya

1. What does sizeof(array) return for int array[10];?
a) Size of one element
b) Number of elements
c) Total size of the array in bytes
d) Size of the pointer to the array

Answer: c) Total size of the array in bytes
Explanation: sizeof(array) gives the total memory occupied by the array.

2. Which of the following is the correct syntax to send an array as a parameter to a C function?
a) Both func(&array) and func(*array);
b) Both func(#array) and func(&array);
c) Both func(array) and func(&array);
d) Both func(array[size]) and func(*array);

Answer: c
Explanation: None.

3.What is the default value of an uninitialized array in C?
a) 0
b) garbage value
c) NULL
d) undefined

Answer: b) garbage value
Explanation: In C, uninitialized variables (including arrays) hold garbage values.

4. What are the elements present in the array of the following C code?
int array[5] = {5};
a) 5, 5, 5, 5, 5
b) 5, 0, 0, 0, 0
c) 5, (garbage), (garbage), (garbage), (garbage)
d) (garbage), (garbage), (garbage), (garbage), 5

Answer: b
Explanation: None.

5. Which of the following declaration is illegal?
a) int a = 0, b = 1, c = 2; int array[3] = {a, b, c};
b) int size = 3; int array[size];
c) int size = 3; int array[size] = {1, 2, 3};
d) All of the mentioned

Answer: c
Explanation: This is illegal in both C99 and C11 because you cannot initialize a variable-length array (VLA). VLAs cannot have an initializer list.

For option B : Explanation:
Variable-Length Arrays (VLAs):
In C89 (ANSI C): This is illegal because array sizes must be constant expressions.
In C99 and later standards: This is legal because VLAs were introduced, allowing the size of an array to be determined at runtime.
In C++: VLAs are not allowed. Array sizes must be compile-time constants.
Conclusion:
Legal in C99 and C11 but illegal in C89 and C++.

6. Which of the following c statement will calculate the correct size of an array of 10 integers? (Assuming the declaration as int a[10];)
a) sizeof(a[10]);
b) sizeof(*a);
c) sizeof(a);
d) sizeof(&a);

Answer: c
Explanation: None.

7. #include <stdio.h>
void main() {
int arr[] = {1, 2, 3, 4, 5);
int *p = arr;
printf("%d", *p++);
printf("%d", (p + 1));
}
Find the output of the above code?
1. 1,2
2. 1,3
3. 2,3
4. 1,4

Answer : 2
Explanation : Let's break down the code step by step:
Step 1:
Initially, arr' is an array with elements (1, 2, 3, 4, 5) and 'p' is a pointer pointing to the first element of the array, i.e., arr[0].
Line 1:
printf("%d", *p++);
Here, ""p" gives the value at 'p', which is '1' (the first element of the array). Then 'p' is incremented to point to the next element of the array (arr[1]). Therefore, the output of this line is 1.

Step 2:
After Line 1, 'p' now points to arr[1] (which is 2).
Line 2:
printf("%d", *(p + 1));
Here, p+ 1 points to arr[2] (since 'p' is currently pointing to arr[1]'). Therefore, **(p + 1) gives the value at arr[2], which is 3 Therefore, the output of this line is '3.
Hence the correct answer is:
The output of the above code is 1, 3.

8. How can the number of elements in an array be determined in C?
a) Using the sizeof operator

b) Using the length() function
c) By manually counting the elements
d) Using the size() function
answer : a
Explanation:
In C, the number of elements in an array can be determined using the sizeof operator, provided the array's size is known at compile time. Here's how it works:

Formula:
To calculate the number of elements in an array:
Number of elements = sizeof(array) / sizeof(array[0])
sizeof(array) gives the total size of the array in bytes.
sizeof(array[0]) gives the size of a single element in the array.

9. What is a multidimensional array in C?
a) An array with elements of different types
b) An array with functions as elements
c) An array of arrays
d) An array with more than 100 elements

Answer : c
A multidimensional array in C is an array whose elements are themselves arrays.

10. What is the output of the following C code?
```
int arr[3] = {0};
printf("%d %d %d", arr[0], arr[1], arr[2]);
```
a) 0 0 0
b) Garbage values
c) Error
d) 0 followed by garbage values

Answer : a
The array is initialized with all elements set to 0, so the output will be 0 0 0.

11. What will be output of the following program
```
int main()
{
int b[4]={5,1,32,4);
int k,l,m;
k=++b[1];
l=b[1]++;
m=b[k++];
printf("%d,%d, %d",k,l,m);
return 0;
}
```
a) 2.2,4
b) 3, 2, 32
c) 3, 2, 4
d) 2, 3, 32

Answer : b
Explanation: Here, ++b[1] means that firstly b[1] will be incremented so, b(1) = 2 then assigned to kie, k = 2

b[1]++ means firstly b[1] will be assigned to variable 1 Le. 1-2, Then value stored in b[1] will be incremented ie. b(1) = 3
b(k + 1/2) means first b[k] will be assigned to mie. m = 32 then value of k will be incremented le. k = 3

12. How do you declare a pointer to an integer array of size 5?
a) int (*ptr)[5];
b) int *ptr[5];
c) int **ptr;
d) int ptr[5];

Answer: a) int (*ptr)[5];
Explanation: The parentheses ensure ptr is a pointer to an array of 5 integers.

13. How do you access the second element of an array using a pointer?
a) *(arr + 2)
b) *(arr + 1)
c) arr[2]
d) arr[1]

Answer: b) *(arr + 1)
Explanation: arr + 1 points to the second element, and * dereferences it.

14. How do you pass a 2D array to a function?
a) int array[][]
b) int (*array)[]
c) int array[][cols]
d) int *array[]

Answer: c) int array[][cols]
Explanation: The number of columns must be specified to handle the array correctly.

15. What does int *ptr = arr; mean for int arr[5];?

a) ptr points to the first element of arr.
b) ptr is an array of pointers.
c) ptr points to the entire array.
d) ptr is invalid.
Answer: a) ptr points to the first element of arr.
Explanation: Arrays decay to pointers to their first element.

16. How do you declare a pointer to a 2D array of size [3][4]?
a) int *ptr[3][4];
b) int (*ptr)[4];
c) int ptr[3][4];
d) int **ptr;

Answer: b) int (*ptr)[4];
Explanation: The parentheses ensure ptr is a pointer to an array of 4 integers.

17. What is the output of the following code?
int arr[2][3] = {{1, 2, 3}, {4, 5, 6}};
int (*ptr)[3] = arr;
printf("%d", *(*(ptr + 1) + 2));

a) 3
b) 4
c) 5
d) 6
Answer: d) 6
Explanation: ptr + 1 points to the second row, and +2 accesses the third element in that row.

18. How do you pass a pointer to a 2D array to a function?
a) void func(int **arr);
b) void func(int arr[3][4]);
c) void func(int (*arr)[4]);
d) void func(int *arr[3]);

Answer: c) void func(int (*arr)[4]);
Explanation: A pointer to a 2D array requires the number of columns to be specified.

19. What is the output of the following pseudo-code?
int arr[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
int (*ptr)[3] = arr;
printf("%d", *(ptr[2] + 1));
a) 7
b) 8
c) 9
d) Undefined behavior

Answer: b) 8
Explanation: ptr[2] points to the third row, and +1 accesses the second element in that row.

20. What is the type of arr in the following declaration?
int *arr[5];
a) pointer to an array of integers
b) array of pointers to integers
c) pointer to a pointer to an integer
d) array of integers

Answer: b) array of pointers to integers
Explanation: arr is an array containing 5 pointers to integers.

21. What is the output of this pseudo-code?
char *arr[] = {"Hello", "World"};
printf("%s", arr[1]);
a) Hello
b) World
c) HelloWorld
d) Undefined behavior

Answer: b) World
Explanation: arr[1] points to the second string, "World".

22. Which function is used to allocate memory dynamically for an array in C?
a) malloc
b) calloc

c) realloc
d) All of the above

Answer: d) All of the above
Explanation: malloc, calloc, and realloc can all be used to allocate memory dynamically.

23. Which function is used to allocate zero-initialized memory blocks dynamically for an array in C ?
a) malloc
b) calloc
c) realloc
d) All of the above

Answer: b) calloc
Explanation: malloc() allocates uninitialized memory, whereas calloc() allocates zero-initialized memory blocks, making dynamic data structures like linked lists and arrays flexible.

24. What is the correct way to allocate memory for an array of 10 integers?
a) int *arr = malloc(10);
b) int *arr = malloc(10 * sizeof(int));
c) int arr = malloc(10 * sizeof(int));
d) int *arr = malloc(10 * sizeof(char));

Answer: b) int *arr = malloc(10 * sizeof(int));
Explanation: malloc requires the size in bytes, so 10 * sizeof(int) is used.

25. What is the correct way to free a dynamically allocated array?
a) delete arr;
b) free(arr);
c) delete[] arr;
d) clear(arr);

Answer: b) free(arr);
Explanation: The free function is used to deallocate memory allocated by malloc or calloc.

26. How do you dynamically allocate memory for a 2D array of size [3][4]?
a) int **arr = malloc(3 * 4 * sizeof(int));
b) int **arr = malloc(3 * sizeof(int *));
c) int **arr = malloc(3 * sizeof(int *)); for (int i = 0; i < 3; i++) { arr[i] = malloc(4 * sizeof(int));}
d) int *arr = malloc(3 * 4 * sizeof(int));

Answer: c)
Explanation: A 2D array is allocated as an array of pointers, where each pointer points to a dynamically allocated row.

27. What is the output of the following pseudo-code?
```
int **arr = malloc(2 * sizeof(int *));
for (int i = 0; i < 2; i++) {
    arr[i] = malloc(3 * sizeof(int));
}
arr[0][0] = 5;
printf("%d", arr[0][0]);
```
a) 5

b) 0
c) Undefined behavior
d) Compile-time error

Answer: a) 5
Explanation: The memory is allocated correctly, and arr[0][0] is set to 5.

28. How do you free a dynamically allocated 2D array?
a) free(arr);
b)for (int i = 0; i < rows; i++) {   free(arr[i]); } free(arr);
c) delete arr;
d) clear(arr);

Answer: b)
Explanation: Each row must be freed individually, and then the array of pointers must be freed.

29. What is the output of this pseudo-code?
```
int arr[] = {1, 2, 3, 4};
int *ptr = arr;
printf("%d", *(++ptr));
```
a) 1
b) 2
c) 3
d) Undefined behavior

Answer: b) 2
Explanation: ++ptr increments the pointer to point to the second element.

30. What is the output of this pseudo-code?
```
char *arr[] = {"C", "Programming", "Language"};
printf("%c", arr[1][0]);
```
a) C
b) P
c) L
d) Undefined behavior

Answer: b) P
Explanation: arr[1] points to "Programming", and [0] accesses the first character.

31. What is the output of this pseudo-code?
```
void func(int arr[]) {
   arr[0] = 10;
}
int main() {
   int arr[3] = {1, 2, 3};
   func(arr);
   printf("%d", arr[0]);
}
```
a) 1
b) 2
c) 10
d) Undefined behavior
Answer: c) 10

Explanation: Arrays are passed by reference, so changes in the function affect the original array.

32. What is the output of this pseudo-code?
int arr[] = {1, 2, 3, 4};
printf("%p", arr);
a) Address of the first element
b) Address of the array
c) Address of the last element
d) Undefined behavior

Answer: a) Address of the first element
Explanation: arr decays to a pointer to its first element. Pointers Format Specifier In C (%p) In other words, the %p format specifier is used to print the memory address of a variable.

33. What is the output of this pseudo-code?
int arr[3] = {10, 20, 30};
int *ptr = &arr[1];
printf("%d", *(ptr - 1));
a) 10
b) 20
c) 30
d) Undefined behavior

Answer: a) 10
Explanation: ptr - 1 points to the previous element in the array.

34. What is the size of a pointer in C on a 64-bit system?
a) 4 bytes
b) 8 bytes
c) Depends on the data type it points to
d) 16 bytes

Answer: b) 8 bytes
Explanation: On a 64-bit system, pointers are typically 8 bytes, regardless of the type they point to.

35. What is the difference between int *ptr and int (*ptr)[5]?
a) int *ptr points to a single integer, while int (*ptr)[5] points to an array of 5 integers.
b) Both are the same.
c) int *ptr is a pointer, and int (*ptr)[5] is an array.
d) int (*ptr)[5] is invalid syntax.

Answer: a) int *ptr points to a single integer, while int (*ptr)[5] points to an array of 5 integers.
Explanation: The parentheses in int (*ptr)[5] indicate that ptr is a pointer to an array.

36. What does sizeof(*arr) return if arr is a pointer to a 1D array of integers?
a) The size of the pointer.
b) The size of the array.
c) The size of one integer.
d) Undefined behavior.
Answer: c) The size of one integer.
Explanation: Dereferencing arr gives the type of the first element, which is an integer.

37. What is the difference between sizeof(arr) and sizeof(ptr) where arr is an array and ptr is a pointer to the array?
a) Both return the size of the array.
b) sizeof(arr) gives the total size of the array, while sizeof(ptr) gives the size of the pointer.
c) Both return the size of the pointer.
d) sizeof(arr) gives the size of the first element, while sizeof(ptr) gives the size of the array.
Answer: b) sizeof(arr) gives the total size of the array, while sizeof(ptr) gives the size of the pointer.
Explanation: sizeof(arr) considers the entire array, while sizeof(ptr) only considers the pointer.

38. Which of the following is not a valid way to declare a pointer to an array?
a) int *ptr;
b) int (*ptr)[5];
c) int ptr[5];
d) int **ptr;

Answer: c) int ptr[5];
Explanation: int ptr[5]; declares an array, not a pointer to an array.

39. Which of the following is valid syntax for accessing elements in a dynamically allocated 2D array?
a) ptr[i][j]
b) *(*(ptr + i) + j)
c) ptr[i * cols + j]
d) All of the above

Answer: d) All of the above
Explanation: Depending on how the memory is allocated, all methods can be valid.

40. Which of the following is true for pointer arrays like char *arr[]?
a) arr stores the addresses of strings.
b) arr can dynamically grow in size.
c) Both a) and b).
d) Neither a) nor b).

Answer: a) arr stores the addresses of strings.
Explanation: char *arr[] is an array of pointers, each pointing to a string.