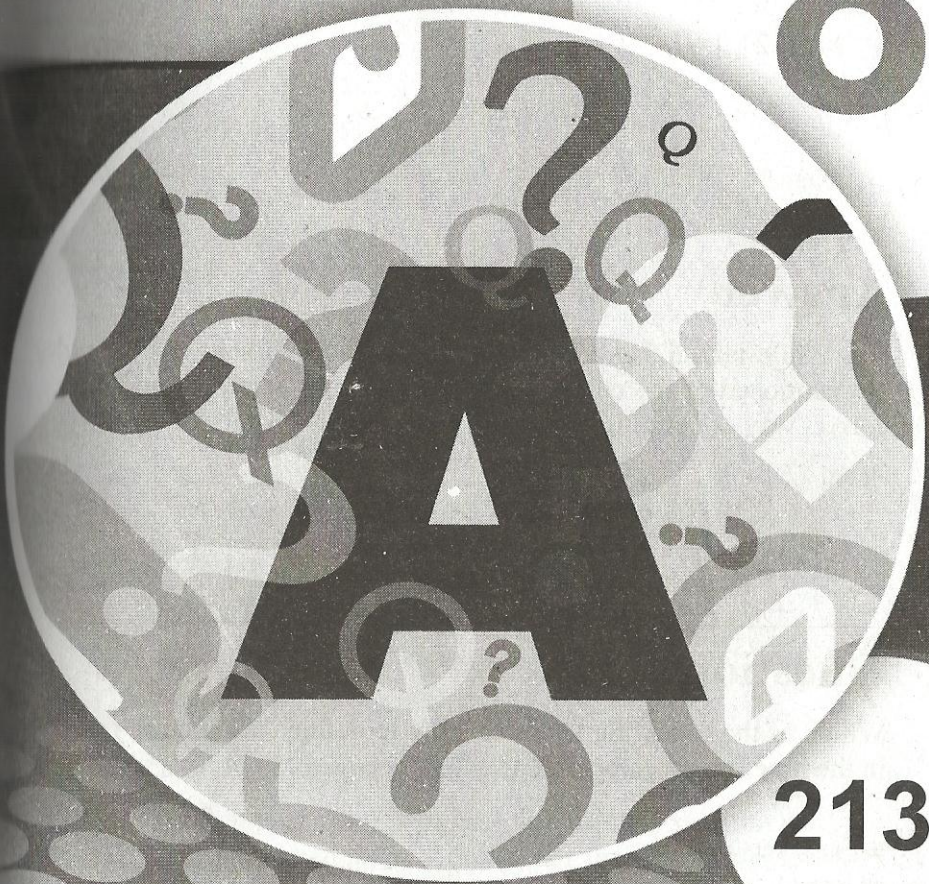
8 Arrays



Question 8.1

Is *arr* correctly defined in the following program?

```
#include <stdio.h>
int main( )
{
    int arr[ 2 ][ ] = {
                        { 1, 2, 3, 4, 5 },
                        { 6, 7, 8, 9, 10 }
                      };
    return 0 ;
}
```

Answer

No. While initializing a 2-D array, either both dimensions should be mentioned or the column dimension should be mentioned. The correct way of defining *arr* would be:

```
int arr[ ][ 5 ] = {
                    { 1, 2, 3, 4, 5 },
                    { 6, 7, 8, 9, 10 }
                  };
```

Question 8.2

Which of the following is correct way to define the function *fun( )* in the program given below?

```
#include <stdio.h>
int main( )
{
    int a[ 3 ][ 4 ] ;
    fun ( a ) ;
    return 0 ;
}
```


A.
```
void fun ( int p[ ][ 4 ] )
{
}
```

B.
```
void fun ( int *p )
{
}
```

C.
```
void fun ( int *p[ 4 ] )
{
}
```

D.
```
void fun ( int *p[ ][ 4 ] )
{
}
```

E.
```
void fun ( int *p[ 3 ][ 4 ] )
{
}
```

Answer

A

Question 8.3

Which of the following statements are correct about an array?

A. The array *int num[26]* ; can store twenty-six elements.
B. The expression *num[1]* designates the very first element in the array.
C. It is necessary to initialize the array at the time of declaration.
D. The expression *num[27]* designates the twenty-eighth element in the array.
E. The declaration *num[SIZE]* is allowed if *SIZE* is a macro.

Answer

A, D, E

Question 8.4

What do 3, 4, and 5 signify in the following definition?

```
int arr[ 3 ][ 4 ][ 5 ] ;
```

Answer

arr is a collection of 3 2-D arrays, each containing 4 rows and 5 columns of integers.

Question 8.5

Which of the following statements are correct about 6 used in the following C expressions?

```
int num[ 6 ] ;
num[ 6 ] = 21 ;
```

A. In the first statement 6 specifies a particular element, whereas in the second statement it specifies a type.
B. In the first statement 6 specifies the array size, whereas in the second statement it specifies a particular element of the array.
C. In the first statement 6 specifies a particular element, whereas in the second it specifies the array size.
D. In both the statements 6 specifies array size.
E. In the first statement 6 specifies array size, whereas in the second statement it specifies that the array size be increased from 6 to 21.

Answer

B

Question 8.6

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    int arr[1] = { 10 } ;
    printf ( "%d\n", 0[ arr ] ) ;
    return 0 ;
}
```

A. 1
B. 10
C. 0
D. None of the above

Answer

B

Question 8.7

What will be the output of the following program, if the array begins at address 2752244 in memory?

```
#include <stdio.h>
int main( )
{
    int arr[ ] = { 2, 3, 4, 1, 6 } ;
    printf ( "%u %u %u\n", arr, &arr[ 0 ], &arr ) ;
```

```
    return 0 ;
}
```

Answer

2752244 2752244 2752244

Question 8.8

In which of the following cases mentioning the name of the array DOES NOT yield the address of the zeroth element of the array?

A. When array name is used with the *sizeof* operator.
B. When array name is operand of the & operator.
C. When array name is passed to *scanf( )* function.
D. When array name is passed to *printf( )* function.
E. When array name is used in arithmetic operations.

Answer

A, B

Question 8.9

Does mentioning the array name gives the address of zeroth element of the array in all the contexts?

Answer

No. Whenever mentioning the array name gives its base address it is said that the array has decayed into a pointer. This decaying doesn't take place in two situations:

- When array name is used with *sizeof* operator.
- When the array name is the operand of the & operator.

Question 8.10

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    static int arr[ ] = { 0, 1, 2, 3, 4 } ;
    int  *p[ ] = { arr, arr + 1, arr + 2, arr + 3, arr + 4 } ;
    int  **ptr =  p ;
    ptr++ ;
    printf ( "%d %d %d\n", ptr - p, *ptr - arr, **ptr ) ;
    *ptr++ ;
    printf ( "%d %d %d\n", ptr - p, *ptr - arr, **ptr ) ;
    *++ptr ;
    printf ( "%d %d %d\n", ptr - p, *ptr - arr, **ptr ) ;
    ++*ptr ;
    printf ( "%d %d %d\n", ptr - p, *ptr - arr, **ptr ) ;
    return 0 ;
}
```

A.
```
0 0 0
1 1 1
2 2 2
3 3 3
```

B.
```
1 1 2
2 2 3
3 3 4
4 4 1
```

C.
```
1 1 1
2 2 2
3 3 3
3 4 4
```

D. 0 1 2
1 2 3
2 3 4
3 4 Garbage

Answer

C

Question *8.11*

What will be the output of the following program, if the array begins at address 1898320?

```
#include <stdio.h>
int main( )
{
    int arr[ ] = { 12, 14, 15, 23, 45 } ;
    printf ( "%u %u %u %u\n", arr, &arr, arr + 1, &arr + 1 ) ;
    return 0 ;
}
```

Answer

1898320 1898320 1898324 1898340

Question *8.12*

Are the expressions *arr* and *&arr* same for an array of 10 integers?

Answer

No. Even though both may give the same addresses (refer 8.11), they mean two different things. *arr* gives the address of the first *int*, whereas *&arr* gives the address of array of *int*s. Since these

addresses happen to be same the results of the expressions *arr* and *&arr* are same. However, the expression *arr + 1* will give the address of the next integer, whereas *&arr + 1* will give the address of next array of 5 integers.

Question *8.13*

Which of the following statements are correct about the C program given below?

```
#include <stdio.h>
int main( )
{
    int size ;
    scanf ( "%d", &size ) ;
    int arr[ size ] ;
    for ( i = 1 ; i <= size ; i++ )
    {
        scanf ( "%d", arr[ i ] ) ;
        printf ( "%d\n", arr[ i ] ) ;
    }
    return 0 ;
}
```

A. The code is erroneous since the subscript for array used in *for* loop is in the range 1 to *size*.
B. The code is erroneous since the values of array are getting scanned through a loop.
C. The code is erroneous since the statement declaring array is invalid.
D. The code is erroneous since the type declaration statement *int arr[size]* ; is done after *scanf( )*.
E. The code is correct and runs successfully.

Answer

C, D

Question 8.14

What will be the output of the following program, if the array begins at 2424256 and each integer occupies 4 bytes?

```
#include <stdio.h>
int main( )
{
    int arr[ ][ 5 ] = {
                        { 12, 14, 15, 23, 45 },
                        { 10, 20, 30, 40, 50 }
                      };
    printf ( "%u %u %u\n", arr, arr + 1, &arr + 1 ) ;
    return 0 ;
}
```

Answer

2424256 2424276 2424296

Question 8.15

What will be the output of the following program?

```
#include <stdio.h>
int main( )
{
    float a[ ] = { 12.4, 2.3, 4.5, 6.7 } ;
    printf ( "%d\n", sizeof ( a ) / sizeof ( a[0] ) ) ;
    return 0 ;
}
```


Answer

4

Question 8.16

A pointer to a block of memory is effectively same as an array. [True/False]

Answer

True

Question 8.17

What will be the output of the following program if the array begins at 1965476 and each integer occupies 4 bytes?

```
#include <stdio.h>
int main( )
{


    int a[ ][3][4] = {
                        {
                            { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }
                        },
                        {
                            { 1, 1, 1, 1 }, { 2, 2, 2, 2 }, { 3, 4, 30, 40 }
                        }
                      };
    printf ( "%u %u %u\n", a, a + 1, &a + 1 ) ;
    return 0 ;
}
```

Answer

1965476 1965524 1965572

Question 8.18

In C, if you pass an array as an argument to a function, what actually gets passed?

A. Base address of the array
B. First element of the array
C. Address of the last element of the array
D. Number of elements of the array
E. Address of the last element of the array

Answer

A

Question 8.19

What does the following declaration mean?

```
int ( *ptr )[10] ;
```

Answer

ptr is a pointer to an array of 10 integers.

Question 8.20

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
```



```
    int a[2][2] = { 1, 2, 3, 4 } ;
    int  i, j ;
    int *p[ ] = { (int * ) a, ( int * ) a + 1, ( int * ) a + 2 } ;
    for ( i = 0; i < 2 ; i++ )
    {
        for ( j = 0 ; j < 2 ; j++ )
            printf ( "%d %d %d %d\n",
                *( *( p + i ) + j ),
                *( *( j + p ) + i ),
                *( *( i + p ) + j ),
                *( *( p + j ) + i ) ) ;
    }
    return 0 ;
}
```

A.
```
1 1 1 1
2 2 2 2
2 2 2 2
3 3 3 3
```

B.
```
1 2 1 2
2 3 2 3
3 4 3 4
4 2 4 2
```

C.
```
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4
```

D.
```
1 2 3 4
2 3 4 1
3 4 1 2
4 1 2 3
```

Answer

A

Question 8.21

If we pass the name of a 1-D *int* array to a function it decays into a pointer to an *int*. If we pass the name of a 2-D array of integers to a function what will it decay into?

Answer

It will decay into a pointer to an array and not a pointer to a pointer.

Question 8.22

How will you define the function *f( )* in the following program?

```
int arr[MAXROW][MAXCOL] ;
fun ( arr ) ;
```

Answer

```
void fun ( int a[ ][MAXCOL] )
{
}
```

OR

```
void fun ( int ( *ptr )[ MAXCOL] ) /* ptr is pointer to an array */
{
}
```


Question 8.23

What will be the output of the following program?

```
#include <stdio.h>
void fun ( int **p ) ;
int main( )
{
    int a[3][4] = {
                    1, 2, 3, 4,
                    4, 3, 2, 8,
                    7, 8, 9, 0
                  } ;
    int *ptr ;
    ptr = &a[0][0] ;
    fun ( &ptr ) ;
    return 0 ;
}
void fun ( int **p )
{
    printf ( "%d\n", **p ) ;
}
```

Answer

1

Question 8.24

What will happen if in a C program you assign a value to an array element whose subscript exceeds the size of array?

A. The element will be set to 0.
B. Nothing, it is done all the time.
C. The compiler would report an error.
D. The program may crash if some important data gets overwritten.

E. The array size would appropriately grow.

Answer

D

Question 8.25

Is there any difference in the following declarations?

```
int fun ( int  arr[ ] ) ;
int fun ( int arr[2] ) ;
```

Answer

No. Though the formal arguments given in the function prototype are different, there isn't any difference in the two statements. This is because, when we pass an array to a function we assume that an array is passed, and the function receives the array. But, since, the array decays immediately into pointers, the declaration

```
int fun ( int  *arr[ ] )
```

is treated as

```
int fun ( int  *arr )
```

Question 8.26

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    void fun ( int, int [ ] ) ;
    int arr[ ] = { 1, 2, 3, 4 } ;
    int i ;
    fun ( 4, arr ) ;
    for ( i = 0 ; i < 4 ; i++ )
        printf ( "%d ", arr[ i ] ) ;
    printf ( "\n" ) ;
    return 0 ;
}
void fun ( int n, int arr[ ] )
{
    int *p ;
    int i ;
    for ( i = 0 ; i < n ; i++ )
        p = &arr[ i ] ;
    *p = 0 ;
}
```

A. 2 3 4 5
B. 1 2 3 0
C. 0 1 2 3
D. 3 2 1 0

Answer

B

Question 8.27

Which of the following is the correct output for the program given below?

```
#include <stdio.h>
int main( )
{
    int a[ 5 ] = { 5, 1, 15, 20, 25 } ;
    int i, j, m ;
    i = ++a[ 1 ] ;
    j = a[ 1 ]++ ;
    m = a[ i++ ] ;
```

```
    printf ( "%d %d %d\n", i, j, m ) ;
    return 0 ;
}
```

A. 2 1 15
B. 1 2 5
C. 3 2 15
D. 2 3 20
E. 2 2 2

Answer

C