**자료구조론 CC343\_2207**

**Reading assignment 3**

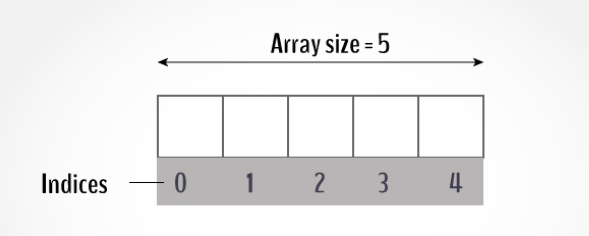
**경기대학교 컴퓨터공학부**

**201511837 이상민**

**Review Questions**

1. What are arrays and why are they needed?  
Arrays란 무엇이며 왜 필요한가?

* Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type.
* An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
* It is used to represent multiple data items of the same type by using only a single name.
* Arrays are more efficient and beneficial when compared to linked lists and hash tables.
* Faster and can be utilized anywhere.
* Store data of similar data types together and can be used anywhere in the code.

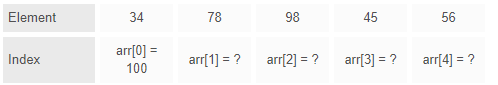


2. How is an array represented in the memory?  
기억 속에 배열은 어떻게 표현되는가?

Mainly arrays are categorized in

* One dimensional array
* Multidimensional array

One dimensional arrays are the simple arrays that we have so far discussed. In previous examples we have shown how to initialize and declare one dimensional array, now we will discuss how it is represented in the memory. As we know now 1-d array are linear array in which elements are stored in the successive memory locations. The element at the very first position in the array is called its base address. Now consider the following example :

Int arr[5]:  


**Base Address (B)+ No. of bytes occupied by element (C) \* index of the element (i)**

/\* Here C is constant integer and vary according to the data type of the array, for e.g. for integer the value of C will be 4 bytes, since an integer occupies 4 bytes of memory. \*/

Now, we can calculate the starting address of second element of the array as :

arr[1] = 100 + 4 \* 1 = 104/\*Thus starting address of second element of array is 104 \*/

Similarly other addresses can be calculated in the same manner as :

arr[2] = 100 + 4 \* 2 = 108

arr[3] = 100 + 4 \* 3 = 112

arr[4] = 100 + 4 \* 4 = 116

3. How is a two-dimensional array represented in the memory?  
기억 속에 2차원 배열은 어떻게 표현되는가?

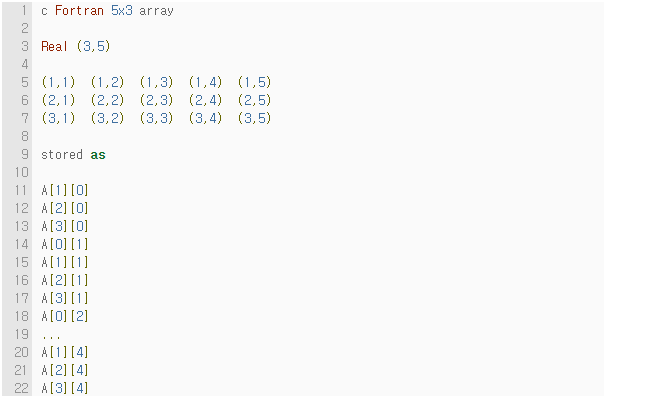
Generally the memory for the values are laid out linearly.

For an element data size of B bytes,  
a leading dimension of N ,  
and a secondary dimension of M,  
A single block of B \* N \* M bytes will be allocated.

In a row-major language, such as C or Java, there will be no “Gaps” between the end of a row and the beginning of the next row. The syntax of access to the 2D array is actually a calculated “2D view” into a single continuous array, and the math is straightforward if not easy to do in your head. It is easy for the compiler, and usually is replaced by a direct offset from the beginning of the array when compilation is done.



Column-major languages such as Fortran do much the same thing, with a slight twist.



4. What is the use of multi-dimensional arrays?  
다차원 어레이의 용도는 무엇인가?

**Multidimensional arrays** are often known as array of the array. In multidimensional arrays the array is divided into rows and columns, mainly while considering multidimensional arrays we will be discussing mainly about two dimensional arrays and a bit about three dimensional arrays. In 2-D array we can declare an array as :

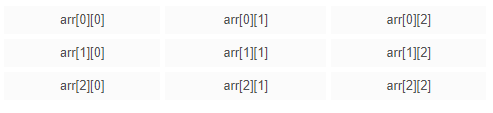
int arr[3][3];

where first index value shows the number of the rows and second index value shows the no. of the columns in the array. We will learn about the 2-D array in detail in the next section, but now emphasize more on how these are stored in the memory.

Mainly multidimensional arrays are stored in the memory in the following two ways :

* **Row-Major order Implementation**
* **Column-Major order Implementation**

**In Row-Major Implementation of the arrays,** the arrays are stored in the memory in terms of the row design, i.e. first the first row of the array is stored in the memory then second and so on. Suppose we have an array named arr having 3 rows and 3 columns then it can be stored in the memory in the following manner :



Thus an array of 3\*3 can be declared as follows :

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

and it will be represented in the memory with row major implementation as follows :



**In Column-Major Implementation of the arrays**, the arrays are stored in the memory in the term of the column design, i.e. the first column of the array is stored in the memory then the second and so on. By taking above eg. we can show it as follows :

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

and it will be represented in the memory with column major implementation as follows :



5. Explain sparse matrix.  
희박한 행렬에 대해 설명하라.

In and , a sparse matrix or sparse array is a in which most of the elements are zero. By contrast, if most of the elements are nonzero, then the matrix is considered dense. The number of zero-valued elements divided by the total number of elements (e.g., m × n for an m × n matrix) is called the sparsity of the matrix (which is equal to 1 minus the density of the matrix). Using those definitions, a matrix will be sparse when its sparsity is greater than 0.5.

Conceptually, sparsity corresponds to systems with few pairwise interactions. Consider a line of balls connected by springs from one to the next: this is a sparse system as only adjacent balls are coupled. By contrast, if the same line of balls had springs connecting each ball to all other balls, the system would correspond to a dense matrix. The concept of sparsity is useful in and application areas such as , which have a low density of significant data or connections.

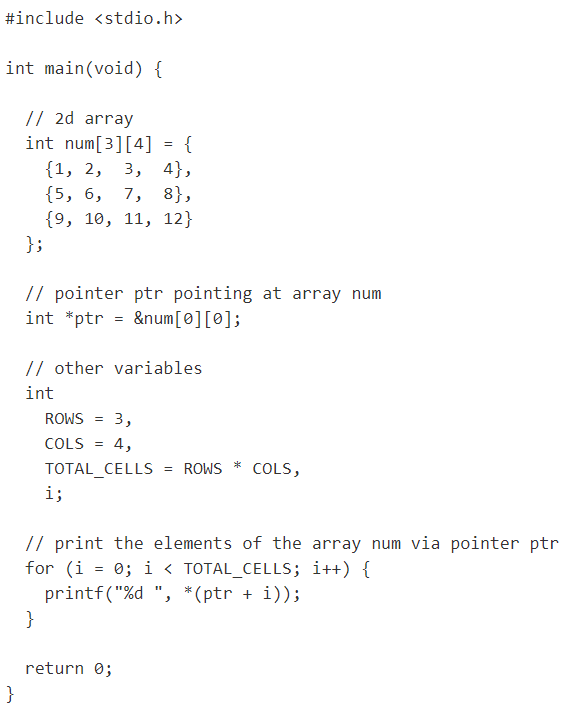
Large sparse matrices often appear in or applications when solving .

When storing and manipulating sparse matrices on a , it is beneficial and often necessary to use specialized and that take advantage of the sparse structure of the matrix. Specialized computers have been made for sparse matrices, as they are common in the machine learning field. Operations using standard dense-matrix structures and algorithms are slow and inefficient when applied to large sparse matrices as processing and are wasted on the zeros. Sparse data is by nature more easily and thus requires significantly less . Some very large sparse matrices are infeasible to manipulate using standard dense-matrix algorithms.

6. How are pointers used to access two-dimensional arrays?  
2차원 어레이에 액세스하는 데 포인터는 어떻게 사용하는가?

The two dimensional array num will be saved as a continuous block in the memory. So, if we increment the value of ptr by 1 we will move to the next block in the allocated memory.

In the following code we are printing the content of the num array using for loop and by incrementing the value of ptr.



Output :



7. Why does storing of sparse matrices need extra consideration? How are sparse matrices stored efficiently in the computer ’s memory?  
희소성 매트릭스의 저장이 특별한 고려가 필요한 이유는 무엇인가? 스파스 매트릭스는 어떻게 컴퓨터의 메모리에 효율적으로 저장되는가?

Obviously, sparse matrices are the special case of full matrices, but they are extremely useful in some, mostly practical, problems.

A good illustration of the role of sparse matrices is so-called Netflix problem, where you have a user-movie matrix with whole bunch of zeros and you want to build a recommender system. Storing this matrix (let alone building a recommender system) as it is requires #users \* #movies. Now there are around 100M users and over 10K titles, including movies, tv shows, documentaries, etc. Storing 1012 elements is not efficient, if possible at all. Here comes sparse matrices for efficiently storing this type of data. The first and most computationally costly step to build a recommender system is to compute the SVD of given sparse matrix. Once it was computed then it can be updated using efficient low-rank update techniques after new user or movie was added.

The matrices with O(n) non-zero entries reduce the complexity of any matrix operations by n, but this type of matrices are not very typical in practical problems. Nevertheless, there are algorithms that allow to solve sparse linear system of equation and compute few largest or smallest eigenvalues of huge matrices that cannot be stored as full matrices. These two problems are the fundamental problem in linear algebra. Moreover, it is also possible that LU, QR or other decompositions of a dense matrix produce sparse matrices which can be efficiently stored and be used afterwards.

There are 4 essential ways to store them:

* COO (Coordinate format)

The simplest format is to use coordinate format to represent the sparse matrix as positions and values of non-zero elements. You need only 3 arrays:

i, j, val

where i,j are integer array of indices, val is the real array of matrix elements. So storing 3nnz elements is enough for matrix description, where nnz is the number of non-zero elements.

* CSR (Compressed sparse row) or CSC(Compressed sparse row)

In the CSR format a matrix is stored as 3 different arrays:

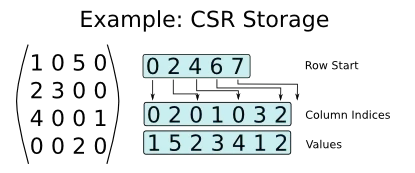


where:

* ia (row start) is an integer array of length
* ja (column indices) is an integer array of length nnz
* sa (values) is an real-value array of length nnz

So, we got 2 nnz + n + 1 elements.

Here is a simple example for understanding CSR:

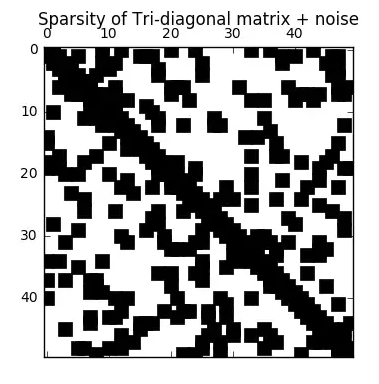


The CSR is optimal for matrix-by-vector product, this can be done as follows:



* LIL (List of lists)  
  LIL format stores one list per row, with each entry containing the column index and the value. A proper sorting (say, by column index) allows to find element faster than looking up for nnz elements.
* Block variants  
  Overall, the general format recommendation for sparse matrix storage is CSR or CSC. The main advantages of this format are optimal storage and optimal matrix-by-vector product. The latter is very important in linear algebra, optimization.

Here you can see an example of sparse matrix — tridiagonal matrix with some additive noise.



**8**. For an array declared as int arr[50], calculate the address of arr[35], if Base(arr) = 1000 and w = 2.  
in[50]으로 선언된 배열의 경우, base(ar)가 1000이고 w = 2인 경우, 부정[35]의 주소를 계산한다.

B = 1000, LB = 0, W = 2, I = 35

Address of A [ I ] = B + W \* ( I – LB )

= 1000 + 2\*(35-0)

= 1000 + 2\*35

= 1000 + 70

= 1070

**9**. Consider a two-dimensional array Marks[10][5] having its base address as 2000 and the number of bytes per element of the array is 2. Now, compute the address of the element, Marks[8][5] , assuming that the elements are stored in row major order.  
기본 주소를 2000으로 하는 2차원 배열 마크[10][5]와 배열 요소당 바이트 수는 2라고 간주한다. 이제 요소가 행의 주요 순서로 저장된다고 가정하여 요소 주소인 Marks[8][5]를 계산한다.

Address of Marks[i][j] along the row= Base Address + w\*(I\*C+j)

Address of Marks[8][5]= Base Address + 2\*(8\*100+5)

2000= Base Address + 2\*(805)

2000= Base Address + 1610

Base Address= 2000-1610

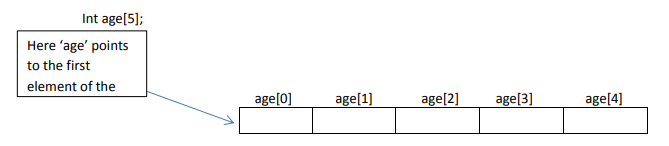
Base Address= 390

10. How are arrays related to pointers?  
어레이는 포인터와 어떻게 관련되어 있는가?

Arrays and pointers are closely related in C. In fact an array declared as

int A[10];

can be accessed using its pointer representation. The name of the array A is a constant pointer to the first element of the array. So A can be considered a const int\*. Since A is a constant pointer, A = NULL would be an illegal statement. Arrays and pointers are synonymous in terms of how they use to access memory. But, the important difference between them is that, a pointer variable can take different addresses as value whereas, in case of array it is fixed.

Consider the following array:  


In C , name of the array always points to the first element of an array. Here, address of first element of an array is &age[0]. Also, age represents the address of the pointer where it is pointing. Hence, &age[0] is equivalent to age. Note, value inside the address &age[0] and address age are equal. Value in address &age[0] is age[0] and value in address age is \*age. Hence, age[0] is equivalent to \*age.

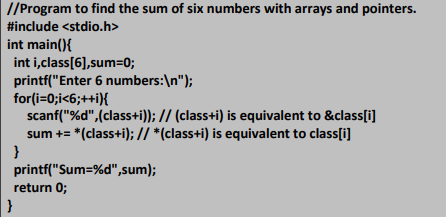
C arrays can be of any type. We define array of ints, chars, doubles etc. We can also define an array of pointers as follows. Here is the code to define an array of n char pointers or an array of strings. char\* A[n];

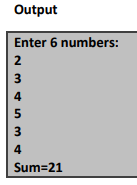
each cell in the array A[i] is a char\* and so it can point to a character. Now if you would like to assign a string to each A[i] you can do something like this.

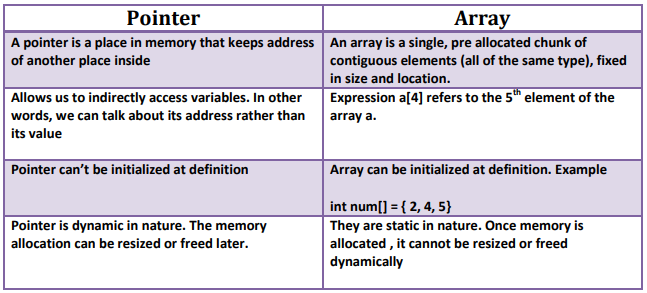
A[i] = malloc(length\_of\_string + 1);

Again this only allocates memory for a string and you still need to copy the characters into this string. So if you are building a dynamic dictionary (n words) you need to allocate memory for n char\*’s and then allocate just the right amount of memory for each string.

In C, you can declare an array and can use pointer to alter the data of an array. This program declares the array of six element and the elements of that array are accessed using pointer, and returns the sum





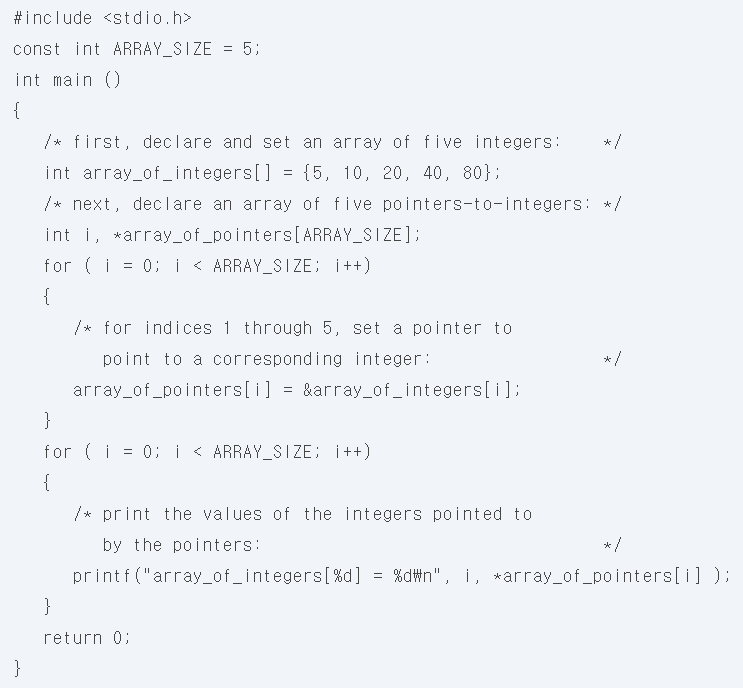


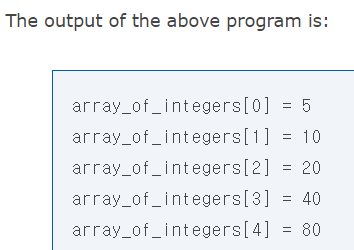
11. Briefly explain the concept of array of pointers.  
포인터의 배열 개념을 간략하게 설명하라.

In , an array of pointers is an indexed set of , where the variables are (referencing a location in ).

Pointers are an important tool in for creating, using, and destroying all types of data structures. An array of pointers is useful for the same reason that all arrays are useful: it allows you to numerically index a large set of variables.

Below is an array of pointers in that points each pointer in one array to an in another array. The value of each integer is printed by dereferencing the pointers. In other words, this code prints the value in memory of where the pointers point.

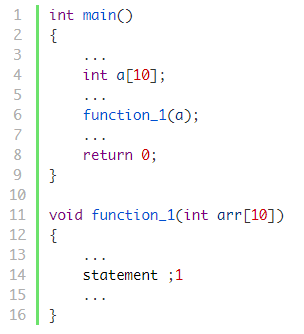




12. How can one-dimensional arrays be used for inter-function communication?  
기능간 통신에 1차원 어레이를 어떻게 사용할 수 있는가?

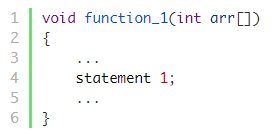
Passing the whole Array to a Function

Just like normal variables you can pass an array variable to a function. But before you do so, make sure the formal arguments is declared as an array variable of same data type. For example:

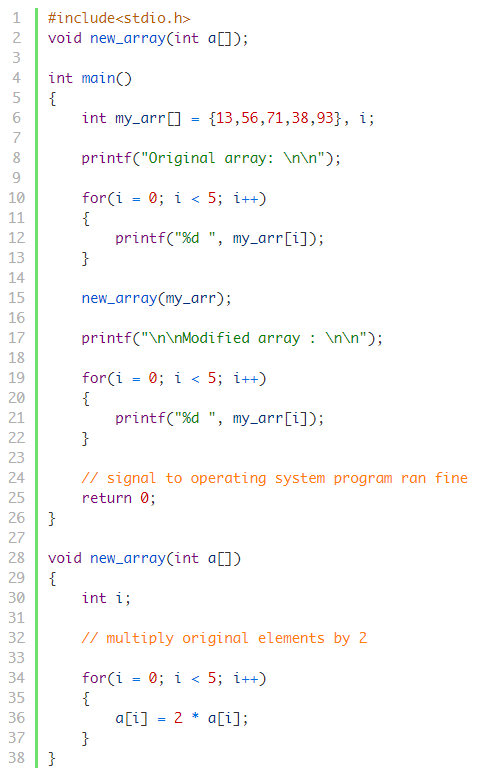


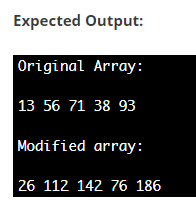
Here we are passing an array of 10 integers to function\_1(), that’s why the formal argument of function\_1() is also declared as an array of 10 integers.

It is optional to specify the size of the array in the formal arguments. This means you can also declare formal argument of function\_1() as follows:



While learning about formal and actual arguments, we have learned that changes made in the formal arguments do not affect the actual arguments. This is not the case with arrays. When an array is passed as an actual argument, the function gets access to the original array, so any changes made inside the function will affect the original array.





13. Consider a two-dimensional array arr[10][10] which has base address = 1000 and the number of bytes per element of the array = 2. Now, compute the address of the element arr[8][5] assuming that the elements are stored in column major order.  
기본 주소 = 1000이고 어레이 = 2의 요소당 바이트 수를 갖는 2차원 배열 부정[10]을 고려한다. 이제 요소가 주요 순서로 저장되어 있다고 가정하여 요소 부정[8][5]의 주소를 계산한다.

Total no. of Rows R=10

Total no. of Columns C=10

Lowest Row lr=0

Lowest Column lc=0

Size of element W=2 bytes

Arr[I][J] i.e., Arr[8][5]=1000

Arragement Order:Row wise

Base Address B=?

* Arr[I][J]=B+W(C(I-lr)+(J-lc))

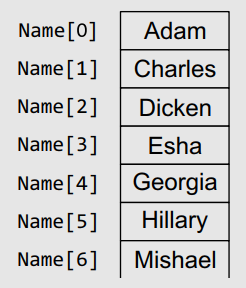
Arr[8][5]=B+2(10(8-0)+(5-0))

1000=B+170

B=830

Base Address=830

14. Consider the array given below:  
아래에 제시된 배열을 고려하십시오.



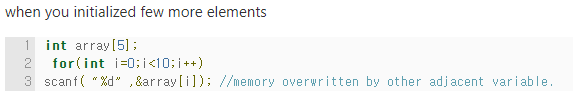
(a) How many elements would be moved if the name Andrew has to be added in it?  
앤드류라는 이름을 그 안에 추가해야 한다면 얼마나 많은 요소들이 옮겨질 것인가?  
(i) 7 (ii) 4 (iii) 5 **(iv) 6**

(b) How many elements would be moved if the name Esha has to be deleted from it?  
에샤라는 이름을 삭제해야 한다면 얼마나 많은 요소가 이동될 것인가?  
(i) 3 **(ii) 4** (iii) 5 (iv) 6

15. What happens when an array is initialized with  
어레이를 초기화할 때 수행되는 작업  
(a) fewer initializers as compared to its size?  
크기에 비해 이니셜라이저가 적으십니까?



(b) more initializers as compared to its size?  
크기에 비해 이니시에이터가 더 많으십니까?



**Programming exercises**

1. Consider an array MARKS[20][5] which stores the marks obtained by 20 students in 5 subjects. Now write a program to

(a) find the average marks obtained in each subject.

(b) find the average marks obtained by every student.

(c) find the number of students who have scored below 50 in their average.

(d) display the scores obtained by every student.

2. Write a program that reads an array of 100 integers. Display all the pairs of elements whose sum is 50.

3. Write a program to interchange the second element with the second last element.

4. Write a program that calculates the sum of squares of the elements.

5. Write a program to compute the sum and mean of the elements of a two-dimensional array.

6. Write a program to read and display a square(using functions).

7. Write a program that computes the sum of the elements that are stored on the main diagonal of a matrix using pointers.

8. Write a program to add two 3 ¥ 3 matrix using pointers.

9. Write a program that computes the product of the elements that are stored on the diagonal above the main diagonal.

10. Write a program to count the total number of non-zero elements in a two-dimensional array.

11. Write a program to input the elements of a two-dimensional array. Then from this array, make two arrays—one that stores all odd elements of the two-dimensional array and the other that stores all even elements of the array.

12. Write a program to read two floating point number arrays. Merge the two arrays and display the resultant array in reverse order.

13. Write a program using pointers to interchange the second biggest and the second smallest number in the array.

14. Write a menu driven program to read and display a p ¥ q ¥ r matrix. Also, find the sum, transpose, and product of the two p ¥ q ¥ r matrices.

15. Write a program that reads a matrix and displays the sum of its diagonal elements.

16. Write a program that reads a matrix and displays the sum of the elements above the main diagonal. ( Hint: Calculate the sum of elements A ij where i<j )

17. Write a program that reads a matrix and displays the sum of the elements below the main diagonal. ( Hint: Calculate the sum of elements A ij where i>j )

18. Write a program that reads a square matrix of size n ¥ n . Write a function int isUpperTriangular(int a[][], int n) that returns 1 if the matrixis upper triangular.  
( Hint: Array A is upper triangular if A ij = 0 andi>j )

19. Write a program that reads a square matrix of sizen ¥ n . Write a function int isLowerTriangular(int a[][], int n) that returns 1 if the matrixis lower triangular.( Hint: Array A is lower triangular if A ij = 0 and i<j )

20. Write a program that reads a square matrix of size n ¥ n . Write a function int isSymmetric(int a[][], int n) that returns 1 if the matrixis symmetric. ( Hint: Array A is symmetric if A ij =A ji for all values of i and j )

21. Write a program to calculate XA + YB where A and B are matrices and X = 2 and Y = 3 .

22. Write a program to illustrate the use of a pointer that points to a 2D array.

23. Write a program to enter a number and break it into n number of digits.

24. Write a program to delete all the duplicate entries from an array of n integers.

25. Write a program to read a floating point array. Update the array to insert a new number at the specified location.

**Multiple-choice Questions**

1. If an array is declared as arr[] = {1,3,5,7,9};then what is the value of sizeof(arr[3]) ?  
(a) 1 **(b) 2** (c) 3 (d) 8

2. If an array is declared as arr[] = {1,3,5,7,9};then what is the value of arr[3] ?  
(a) 1 **(b) 7** (c) 9 (d) 5

3. If an array is declared as double arr[50]; howmany bytes will be allocated to it?  
(a) 50 (b) 100 (c) 200 **(d) 400**

4. If an array is declared as int arr[50] , how manyelements can it hold?  
(a) 49 **(b) 50** (c) 51 (d) 0

5. If an array is declared as int arr[5][5] , howmany elements can it store?  
(a) 5 **(b) 25** (c) 10 (d) 0

6. Given an integer array arr[] ; the i th element canbe accessed by writing  
(a) \*(arr+i) (b) \*(i + arr) (c) arr[i] **(d) All of these**

**True or False**

1. An array is used to refer multiple memory locations having the same name. : True

2. An array name can be used as a pointer. : True

3. A loop is used to access all the elements of anarray. : True

4. An array stores all its data elements in non-consecutive memory locations. : False

5. Lower bound is the index of the last element in an array. : False

6. Merged array contains contents of the first array followed by the contents of the second array. : True

7. It is possible to pass an entire array as a function argument. : True

8. arr[i] is equivalent to writing \*(arr+i). : True

9. Array name is equivalent to the address of its last element. : False

10. mat[i][j] is equivalent to \*(\*(mat + i) + j). : True

11. An array contains elements of the same data type. : True

12. When an array is passed to a function, C passes the value for each element. : False

13. A two-dimensional array contains data of two different types. : False

14. The maximum number of dimensions that an array can have is 4. : False

15. By default, the first subscript of the array is zero. : True

**Fill in the blanks**

1. Each array element is accessed using a \_\_\_\_\_\_.  
: Index or subscript

2. The elements of an array are stored in \_\_\_\_\_\_ memory locations.  
: Consecutive

3. An n -dimensional array contains \_\_\_\_\_\_ subscripts.  
: n

4. Name of the array acts as a \_\_\_\_\_\_.  
: Pointer

5. Declaring an array means specifying the \_\_\_\_\_\_, \_\_\_\_\_\_, and \_\_\_\_\_\_.  
: Data type, name, and size

6. \_\_\_\_\_\_ is the address of the first element in the array.  
: Base address

7. Length of an array is given by the number of \_\_\_\_\_\_.  
: The number of elements stored in it

8. A multi-dimensional array, in simple terms, is an \_\_\_\_\_\_.  
: Array of arrays

9. An expression that evaluates to an \_\_\_\_\_\_\_\_\_\_ value may be used as an index.  
: Integral value

10. arr[3] = 10; initializes the \_\_\_\_\_\_\_\_\_\_ element of the array with value 10.  
: Fourth