# CHAPTER ONE

# BACKGROUND

**OBJECTIVES:**

- Identify data structure
- Identify algorithm
- Identify program
- Describe the importance of data structure
- Identify complexity of algorithm

## 1.1 Data Structure

In "Data Structure" there are two words, one is data and another is structure. Data mean raw facts or information those can be processed to get one or more results or products.

A structure is such a unit where elementary items are organized in different ways to perform one or more operations. As for example, some elementary items like hand, leg, eye, ear, nose, bone and some other organs constitute a human body. So, the human body is a structure. Similarly, some elementary items like pieces of wood, iron, raxine etc may constitute a chair, which is also a structure. A structure may be treated as a frame or proforma where we organize some elementary items in different ways. Like structures in our environment, the data structure is also constituted with some elementary data items.

The *data structure* is a structure or compound unit where we organize elementary data items in different ways to perform operations on them and there exists structural relationship among the items. It indicates a data structure is a means of structural relationships of elementary data items for storing into and retrieving from computer's memory. Usually, elementary data items are the *elements* of a data structure. However, **a data structure may be an *element* of another data structure.** In other words, a data structure may contain another data structure.



Scanned with CamScanner

Examples of Data Structures: Array, Linked List, Stack, Queue, Tree, Graph, Hash Table etc.

Types of elementary data item: Character, Integer, Floating point numbers etc.

Expressions of elementary data in C/C++ programming language are shown below:

| Elementary data item | Expression in C/C++ |
|---|---|
| Character | char |
| Integer | int |
| Floating point number | float |

## 1.2 Operations on Data Structure

We can also perform some operations on data structure such as insertion (addition), deletion (access), searching (locate), sorting, merging etc.

Insertion (addition) means to place (add) an element into a particular data structure. As a result, the number of elements in the data structure will be increased. Deletion (access) indicates removal of an element from a data structure. In some cases, we can delete or remove an element from the data structure. In such cases, the number of elements will be decreased. However, in some cases actual removal (deletion) does not occur, we can only access (read) the element from the data structure and use it in any operation. If the element is not required anymore we can use the space of the element (which was accessed) to store another element. By searching, we understand to locate or find out a specific element of the data structure. Sorting is an arrangement of the elements in any order. The order may be ascending (smaller to greater) or descending (greater to smaller). Merging indicates to collect the elements of two or more data structure into another data structure. To perform any operation we have to design algorithm or/and program.

## 1.3 Algorithm

It is a set or sequence of instructions arranged in steps that can be followed to solve a problem or perform a task. Each and every algorithm can b

Scanned with CamScanner

divided into *three sections*. The first section is *input* section, where we show which data items are to be used for processing. Here we should think about the use of proper data structure to store data items. The second section is very important one, which is *operational* or *processing* section. Here we have to perform all necessary operations, such as computation, taking a decision, calling other procedure (algorithm) etc. The third section is *output*, where we display the result(s) found from the second section. When we design an algorithm we have to remember presentation, clarity, and efficiency of it. An algorithm should look good, easy to understand and code using any programming language should give correct result(s) for all valid inputs as quickly as possible. It should make efficient use of computing resources. The efficiency of an algorithm can be measured by analyzing its complexity. To write an algorithm we do not strictly follow the grammar of any particular programming language. However, its language may be near to a programming language.

## 1.4 Program

A program is a set or sequence of instructions arranged in some statements of any programming language that can be followed to solve a problem or perform a particular task. For a particular problem, at first, we may write an algorithm then the algorithm may be converted into a program. In a program usually, we use a large amount of data. Most of the cases these data are not discrete elementary items, rather there exists a structural relationship among elementary data items. So, the program uses the data structure(s). Like an algorithm, a program can be divided into three sections such as input section, processing section, and output section.

## 1.5 Importance of Data Structure

Computer science and computer engineering deal with two jargons which are *software* and *hardware*. Without software, hardware (electrical, mechanical, electronic parts of a computer that we see and touch) is useless. So, the study of software is very important in computer science, and software consists of programs, which use different types of data. In a

Scanned with CamScanner

program, we not only use elementary data items but also use different types of organized data. In other words, we use the data structure(s) in a program. As we know we write programs to solve problems. It means to solve the problem(s) we have to use data structures. The different data structures give us different types of facilities. If we need to store data in such a way that we have to retrieve data directly irrespective of their storage location, we can get this facility using one type of data structure such as *array* gives us such facility. In some cases, instead of direct access, we may need efficient use of memory and this can be performed using a *linked list*. In our daily life, we handle a list of data such as a list of students, a list of customers, a list of employees etc. However, each of these entities (student, customer, employee etc.) may have different attributes. As for example, a student has roll number, name, marks attributes and these are different in types. How to organize them so that we can handle different types of data as a unit? We can get this facility from *record* or *structure*. Thus the importance of data structures is many folds in storing, accessing, arranging data.

Since all other data structures are implemented using the basic data structures like array, record (structure) and linked list so very good background on these basic data structures is very much essential. There is no single data structure that can be used to solve all the problems. So by achieving the knowledge of data structure, we can use different types of data in programs those are used to solve various problems related to our life. Without knowledge of data structures, we are not able to solve problems where we must use them. In a programming language, there are provisions to use different types of data structures so that data can be organized in different ways and solve the problem properly. In fact, optimization of usage of memory can be done by using proper data structures. In other words, without knowledge of data structures we will not be able to write program properly, hence we will not able to solve the problem.

Sometimes practitioners think that the computing resources like processors and its associated parts are powerful now and there are enormous spaces in memory so the need for data structure is limited. However, let us think

Scanned with CamScanner

about a large amount of data created by a big organization or group of companies and efficient use of computing resources and Big Data. Even scientists are thinking and doing research on new data structure which can process a large amount of data efficiently.

Therefore, for the students and teachers of computer science and engineering, the knowledge of data structure is very much essential.

## 1.6 Complexity of Algorithm

The efficiency of an algorithm or a program depends on its complexity. The complexity of an algorithm can be measured by its running (execution) time and space in memory required for the data used in it. There are two types of complexities: One is **time complexity** and another is **space complexity**.

**Time complexity:** This complexity is related to the execution time of an algorithm or a program. It depends on the number of operations taken by the program or algorithm. The operations are arithmetic and/or relational (greater, less, equal). However, the comparison based algorithm described the number of element comparisons. So, the complexity of an algorithm is computed with respect to the total number of element (item) comparisons needed for the algorithm.

**Space complexity:** This complexity is related to space (memory) needs in the main memory for the data used to implement the algorithm for solving any problem. If there are $n$ data items used in an algorithm, the space complexity of the algorithm will be proportional to $n$.

The complexity of an algorithm (either time complexity or space complexity) is represented using asymptotic notations. One of the asymptotic notations is O (big-oh) notation. In general we write $T(n) = O(g(n))$ if there are positive constants C and a number n. Such that $T(n) \leq cg(n)$ for all n, $n \geq n_0$. In words, the value of T(n) always lies on or below $cg(n)$ for $n \geq n_0$. In this book, we have represented the complexity using O notation. Big-oh (O) notation is also called upper bound of the complexity.

Scanned with CamScanner

If we get the total number of element comparisons is $\frac{1}{2} n^2 - \frac{1}{2} n$, then we can write it as $O(n^2)$. Since $(\frac{1}{2} n^2 - \frac{1}{2} n) < n^2$. Similarly $10n^2 + 4n + 2 = O(n^2)$. Since $10n^2 + 4n + 2 \leq 11n^2$. Some other examples are as follows:

$$5n^2 + 3n + 4 = O(n^2)$$
$$6.2^n + n^2 + 3 = O(2^n)$$
$$3n + 5 = O(n)$$
$$5n\log_2 n + 4 = O(n\log_2 n)$$
$$2n^3 + n^2 + 5n = O(n^3)$$

| Complexity in notation | Complexity in word |
|---|---|
| logn | Logarithmic |
| n | Linear |
| nlogn | |
| $n^2$ | Quadratic |
| $n^3$ | Cubic |
| $2^n$ | Exponential |

$\log n < n < n\log n < n^2 < n^3 < 2^n$

*logn* means $log_2 n$ and it denotes the lowest complexity and $2^n$ denotes the highest complexity.

Table-1: Shows the advantages and disadvantages of the various data structures:

| Data Structure | Advantages | Disadvantages |
|---|---|---|
| Array | Quick insertion, very fast access if index is known | Slow search, slow deletion, fixed size |
| Linked list | Quick insertion, quick deletion | Slow search |
| Stack | Provides last-in, first-out (LIFO) access | Slow access to other items except for top item |
| Queue | Provides first-in, first-out(FIFO) access | Slow access to other items except for front or rear item. |

Scanned with CamScanner

| Data Structure | Advantages | Disadvantages |
|---|---|---|
| Binary tree | Quick search, insertion, deletion (if tree remains balanced) | The deletion process is complex for non-leaf nodes. |
| Hashtable | Very fast access if key is known, Fast insertion | Slow deletion, access slow if key is not known, inefficient memory usage. |
| Heap | Fast insertion, deletion | Slow access to other items other than root. |
| Graph | models real-world situations | Some algorithms are slow and complex |

**Summary:**

**Data structure** is a structure or unit where we organize data items in different ways and there exists a structural relationship of them (data items).

**Algorithm** is a set or sequence of instructions arranged in steps that can be followed to solve a problem or perform a particular task.

**Program** is a set or sequence of instructions of a particular programming language that can be followed to perform a particular task.

Data structure is the most important building block of a program. It facilitates organized storage and easy retrieval of data.

There are two types of complexity, the time complexity, and the space complexity. **Time complexity** is related to the number of element comparisons and element movement. **Space complexity** is used to determine the memory space usage and requirements for a particular problem.

Scanned with CamScanner

**Questions:**

1. What do you mean by data structure?
2. What are the objectives of learning data structure?
3. What are elementary data types?
4. What do you mean by space complexity?
5. Define data structure. Give examples.
6. State two complementary goals of the study of the data structure.
7. Data structure is a structure that may contain another data struc Explain the statement with example(s).
8. Define elementary data type and data structure.
9. What do you mean by a data structure? Explain the basic oper that are normally performed on a particular data structure.
10. Differentiate between atomic data type and structured data type.
11. What is the difference between an algorithm and a program?

Scanned with CamScanner

# CHAPTER TWO

# ARRAY

**OBJECTIVES:**

- Identify array
- Show data storing and accessing methods using array
- Write algorithms using array
- Identify two-dimensional array
- Show data storing and accessing processes using two-dimensional array
- Describe the process of representation of two-dimensional array in computer's memory
- Write algorithms using two-dimensional array
- Identify dynamic array
- Write algorithms using dynamic array
- Give some problems related to array

## ARRAY

### 2.1 Definition

An array is a finite set of the same type of data items. In other words, it is a collection of homogeneous data items (elements). The elements of an array are stored in successive memory locations. The elements of an array are stored using its name and size (number of elements in the array). Any element of an array is referred by *array name* and the *index number* (subscript). There may have many dimensional arrays. But usually, two types of arrays are widely used; such as one dimensional (linear) array and two dimensional array.

### 2.2 One Dimensional Array

#### 2.2.1 Definition

A one dimensional (linear) array can be represented by only one dimension such as row or column and holds a finite number of the same type of data items associated with a name. Graphical view of a one dimensional array is as follows:

Scanned with CamScanner

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Array B → | 15 | 28 | 12 | 17 | 13 | 20 | 19 | 23 | 29 | 39 |

Figure-2.1: Graphical representation of one dimensional array.

Here 0, 1, 2, 3, … … …, 9 are index numbers, and 15, 28, 12, … , 39 are data items or elements of the array and B is a name of the array. Symbolically an element of the array is expressed as $B_i$ or B[*i*], which denotes element in the $i^{th}$ index of the array, B. We will use 0 as starting index.

Thus B[4] and B[9] denote the elements in the $4^{th}$ index and in the $9^{th}$ index of the array, B.
The name of the array usually is a name constituted by one or more characters. Thus array name may be *A*, *S*, *Stock*, *Array1* etc. The elements of an array may be numbers (integers or floating point numbers) or characters.

A one dimensional array at a glance as below:
1) A list of same type of data items.
2) It has a name.
3) It has a size (number of data items).
4) Its elements are associated with sequential numbers called indices such as 0, 1, 2, 3, . . .and so on.
5) Any element or item is represented using a name of the array and its index number.

**Expression of one dimensional array in C/C++:**

For integer array: int a[10];

For character array: char b[30];

For floating point array: float B[10];

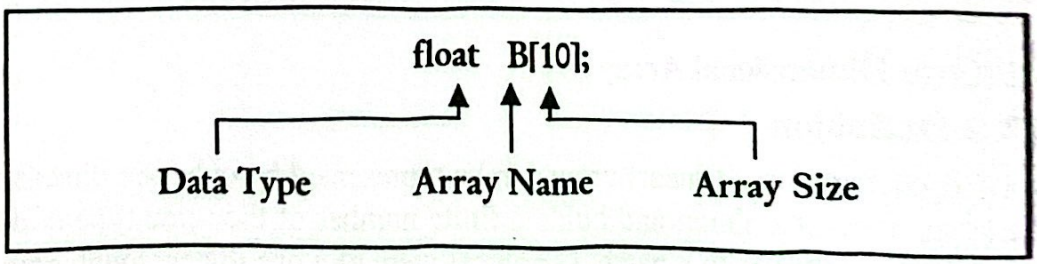



Figure-2.2: Declaration of Array in C/C++.

Scanned with CamScanner

### 2.2.2 Code in C/C+ for storing data in an array

```
int x[10];
for (i = 0; i < 10;  i++)
        scanf ("%d", &x[i]);
```

The equivalent code in C++ is as follows:

```
int x[10];
for (i = 0; i < 10; i++)
cin >> x[i];
```

We can store integer type of data to the array, *x* using the above segment of codes.

The following code in C is for reading data from an array and the data will be displayed on the monitor's screen:

```
for (i =0; i < 10; ++i)
printf ("%d", x[i]);
```

The equivalent code in C++ is as follows:

```
for (i = 0; i< 10; i++)
cout << x[i];
```

In the above examples of code, type of the array, x is an integer. If we want to access (read) any element from the array to a variable p, the type of the variable, p should be an integer. The example of code is as below.

```
int p;
p = x[0];
```

Suppose there is an array c of 10 characters and we want to assign a value to its index 4 (counting from 0), the code will be as follows.

```
char c [10];
c[4]= 'v';
```

If we want to assign a value from a variable the code will be as below.

```
char c[10];
```

Scanned with CamScanner

```
char ch;
ch = '(';
c[4] = ch;
```

It can be seen that types of the array and the variable used to assigned values to the elements of the array are same.

### 2.2.3 Store an element into an array

B[3] = 19; here a value, 19 is assigned to the 4th element (index 3) of the array, B. That is, 19 is stored to the index 3 of the array, B. If there was any previous value, it would be overwritten.

We can assign a variable to any element of the array also. In this case, the type of the array and the type of the variable should be **same**. Such as, *B[2] = x*; here the variable *x* is assigned to the 3rd item of the array, B. If there is any value of *x* that would be copied to the item number 3 of the array. The previous value of item number 3 of the array will be overwritten and there will be no change in the value of *x*.

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 7 | 25 | 28 | 54 | 32 |

Array, B before execution of the statement (B[3]=19).

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 7 | 25 | 28 | **19** | 32 |

Array, B after execution of the statement (B[3]=19).

Figure-2.3: Data storing using assignment

### 2.2.4 Read (retrieve) a value (element) from an array

*C = B[3];* this statement indicates that the item number 4 (index 3) of the array B is assigned to a variable, *c*. The value from the index 3 of the array will be copied to the variable c. Any previous value of c will be overwritten. There will be no change in the value of item number 4 of the array, B. Let us consider the data of the array in Fig-2.3. After execution of the statement C = B[3]; the value of the variable C will be 19.

From the above description, we can say that the assignment statement works *from direction right to left*. The right side contains the value transferring item and the left side is the value receiving item. When it needs to store more than one value to several items of an array, we have to use a

Scanned with CamScanner

loop of the programming language. Such as we can use **for** or **while** loop. Similarly, if we want to access several items from an array we have to use a loop. Since in C/C++ *index of an array start from 0* (zero) instead of 1, so for 10 items in an array, the indices will be 0, 1, 2, 3, . . ., 9.

| 0 | 1 | 2 | 3 | ..... | 8 | 9 |
|---|---|---|---|---|---|---|
| d | F | g | h | ..... | m | p |

Figure-2.4: Array with character

The type of the array indicates the type of its elements. Such as if the type of the array B is character, the elements B[0], B[1], ..., B[9] will also be characters.

**Problem 2.1:**

Given a list of elements, write an algorithm to store the list of elements (numbers) in an array and find out the largest element of the list.

**Algorithm 2.1:** Algorithm to search the largest element of a list

1. Input: $x[0 \ldots n-1]$;
2. for ($i = 0$; $i < n$; $i$ ++) //n is the size of the array

   cin >> x [i];
3. $large = x[0]$;
4. for ($i = 1$; $i < n$; $i$ + +)

   if ($x[i] > large$), $large = x[i]$; //if any element larger than the previous data in *large*
5. Output: *large* (the largest number);

**Comments:** $x$ is the array to store data and $n$ is the size of the list. The above code is as like as C/C++, but not exactly written in C/C++. We shall follow this style for other algorithms.

Scanned with CamScanner

Now we will see the simulation results of Algorithm 2.1. Suppose that there is an array, x with the following data items.

| | | | | |
|---|---|---|---|---|
| 15 | 12 | 9 | 17 | 12 |

We see the simulation results from step 3 since steps 3 and 4 are the steps of processing section.

1) *large* =15
2) Within *for* loop:
   a) (12 > 15) **no** (no change in the value of *large* = i.e. *large* =15)
   b) (9 > 15) **no** (as above)
   c) (17 > 15) **yes** *large* =17 (change in the value of *large* =15)

   d) (12 > 17) **no** (no change in the value of *large*, which was 17).

From the above simulations, we have got *large* =17 which is the largest item of the array, x.

**Problem 2.2:**

Given a linear array with data, find out a particular (specific) element, *x* from the array. We do not know the index (cell) number where the element was stored.

**Algorithm 2.2:** Algorithm to search a particular element from a list

1. Input: A set of data in an array a and variable *x i.e.*, the target element
   *a*[0 . . . *n-1*], *x*; //n is the size of the array
2. flag = 0;
3. for (*i* = 0; *i* < *n*; *i* ++)
   {
   if (*a*[*i*] = = *x*)
   location = i, flag = 1, break;
   }
4. Output: if (flag = = 1), print "Found" and *i* (location).
   else print "Not Found".

Scanned with CamScanner

Simulation results of the algorithm 2.2 are as follows:

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| A | 7 | 15 | 12 | 9 | 24 | 32 |

We will see from step 2 and let $x = 24$

1) Initially *flag* = 0

2) Within for loop:
a) (7 = x) no (no change in the value of flag *i.e.*, *flag* = 0)
b) (15 = x) no (flag = 0)
c) (12 = x) no (flag = 0)
d) (9 = x) no (flag = 0)
e) (24 = x) yes, *flag* = 1 and break (exit the *for* loop).
3) Since *flag* = 1, output will be Found and 4 (*index* = 4)

**Problem 2.3:**
Given a list of integers stored in a linear array. Find out the summations of odd and even numbers separately.

**Description of Solution:** Given a list of integers, we have to find out the odd numbers and then we add those numbers. Similarly, we find out the even numbers in the list and by adding those numbers we shall get the summation of even numbers.

To store the results, we require two variables such as *sum_even* and *sum_odd*. Initially, values of these variables will be zero (0) and when we find an even number we add it to the *sum_even* and we find an odd number, we add it to the *sum_odd*. If a number is divisible by 2 it is even, otherwise odd. We have to start with the first number of the list. If it is even, it is added to the sum_even and if it is odd, it is added to the sum_odd. Similarly, we

Similarly, we shall access the whole list one by one and add them to either sum_even (if a number is even) or sum_odd (if a number is odd).

Scanned with CamScanner

**Algorithm 2.3:** Algorithm to find the summations of even and odd numbers

1. Input: An array and variables (to store the results of summations)

   *A*[0...*n-1*], *sum_odd* = 0, *sum_even* = 0; //n is the size of the array

2. for (*i* = 0; *i* < *n*; *i* + +)

   {

   if (*A*[*i*]%2 = = 0), *sum_even* = *sum_even* + *A*[*i*];

   else *sum_odd* = *sum_odd* + *A*[*i*];

   }

3. Output: *sum_even* (the summation of the even numbers) and *sum_odd* (summation of the odd numbers)

**Comments:** Here *A* is an array that holds a list of integers and *n* is the size of the list. *sum_even* and *sum_odd*, are two variables to store the summation of even numbers and the summation of odd numbers respectively.

**Problem 2.4:**

Given a list of integers stored in a linear array, find out the summations of numbers in odd and even indices separately.

**Solution:** This problem is similar to the problem 2.3. Here the difference is that we have to check whether the index (not the number) is odd or even.

**Algorithm 2.4:** Algorithm to find the summations of the numbers in even and odd indices.

1. Input: An array and variables (to store the results of summations)

   *A*[0...*n-1*], *sum_odd* = 0, *sum_even* = 0; //n is the size of the array

2. for (*i* = 0; *i* < *n*; *i* ++)

   {

   if (*i*%2 = = 0), *sum_even* = *sum_even* + *A*[*i*];

   else *sum_odd* = *sum_odd* + *A*[*i*];

   }

Scanned with CamScanner

3. Output: *sum-even* (Summation of the numbers in even indices), *sum_odd* (summation of the numbers in odd indices)

**Comments:** Here *A* is an array that holds a list of integers and *n* is the size of the list. *sum_even, sum_odd* are two variables to store the summation of numbers in odd indices and the summation of numbers in even indices respectively.

**Problem 2.5:**

Given *n-1* integers in a linear array, stored in 0 to *n-2* positions (indices). **The (*n*-1)th index is empty.** Insert a data (integer) in *k*th position (index), the relative order of the data before the position and after the position will remain as it is, where $0 < k < n-2$. **Condition:** You cannot delete (overwrite any item). Any item and arrangement of data before and after *k*th index should be same as previous.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 12 | 15 | 25 | 28 | 35 | 45 | 52 | 60 | Empty position |

Figure-2.5: Array with data before insertion

Let $k=5$ and $x=40$ should be inserted at 5th index of the array. The output should be as follows:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 12 | 15 | 25 | 28 | 35 | **40** | 45 | 52 | 60 |

Figure-2.6: Array with data after insertion of 40 at 5th index.

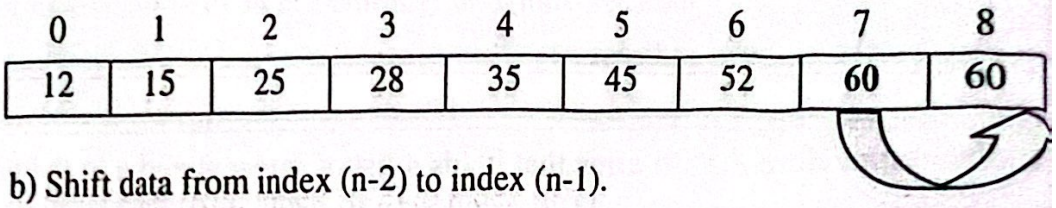
We see now the solution process of this problem in pictorial view.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 12 | 15 | 25 | 28 | 35 | 45 | 52 | 60 | |

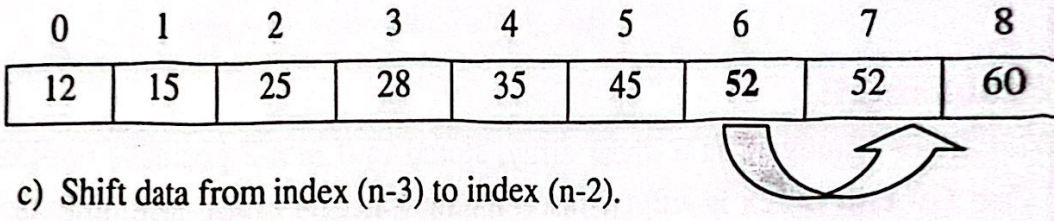
a) Array with data before insertion



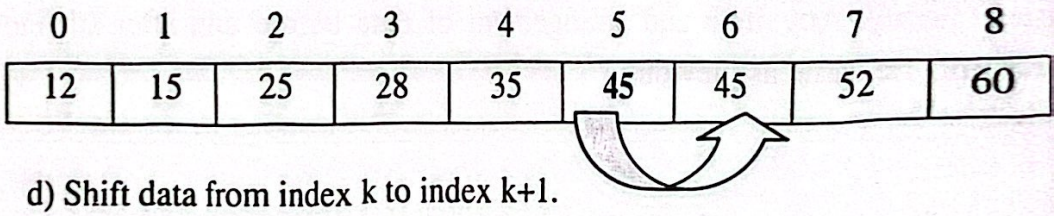
Scanned with CamScanner



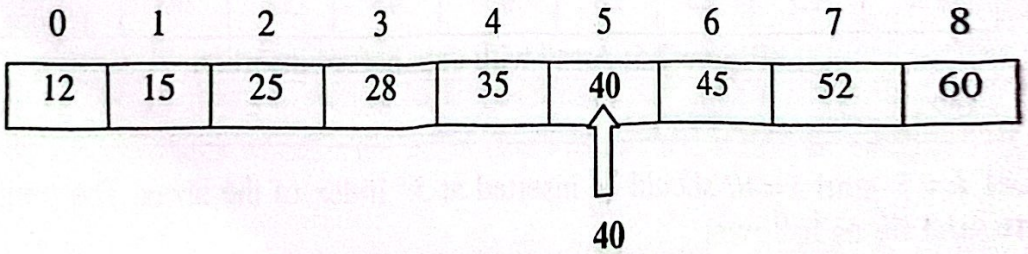

b) Shift data from index (n-2) to index (n-1).




c) Shift data from index (n-3) to index (n-2).




d) Shift data from index k to index k+1.




e) Insert data to the desired position

Figure-2.7: A pictorial view of data insertion to a particular position.

**Algorithm 2.5**: Algorithm to insert an item into a particular position (index) of an array.

1. Input: An array A[0.....n-1] with data in indices 0 to n-2, and the data item, x should be inserted into the index, k.

Scanned with CamScanner

2. for (i = n-2 ;i >= k; i--)
   A[i+1] = A[i]; //A[i+1] ← A[i]
3. A[k] = x; //A[k] ← x

4. Output: Array A with data in all indices.

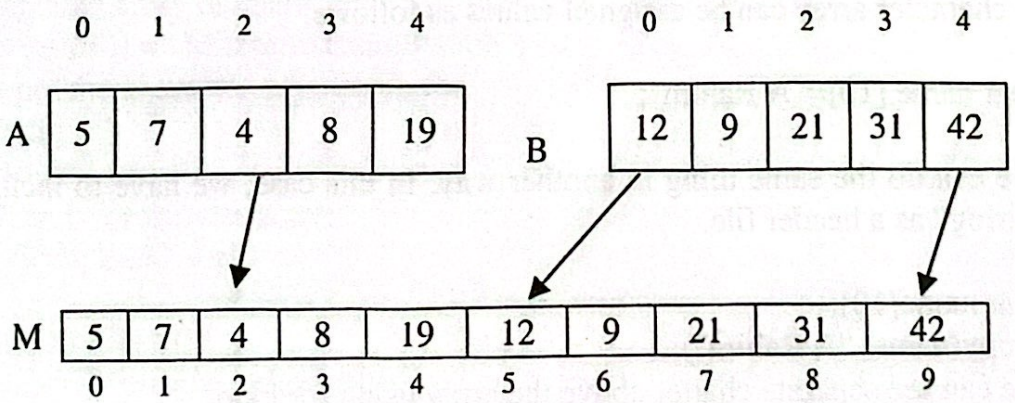
**Problem 2.6:**
Given a list of integers stored in a linear array, delete a data from a given position of the array.

**Algorithm 2.6:** Algorithm to delete an element from an array.

1. Input: An array *A*[0...*n-1*], //n is the size of the array
2. for (*i* = *k*; *i*< *n*; *i* + +)
   *A* [*i*] = *A* [*i* + 1]; //A[i] ← A[i+1]
3. Output: The updated array, *A*.

**Comments:** Here the data of the positions (index) *k* to *n-2* are overwritten by the data of the positions *k + 1* to *n-1* of the array, *A*. Now the position (index) *n-1* of the array is empty.

**Problem 2.7:**
Given two linear arrays of integers, merge the two arrays into a single array.




**Algorithm 2.7:** Algorithm to merge two arrays.

1. Input: Two arrays *A* [0...*m-1*] and *B*[0...*n-1*] //m is the size of A and //n is the size of B
2. Take an array *M* [0... *m* + *n-1*] //m + n is the size of M

Scanned with CamScanner

3. for ($i = 0$; $i < m$; $i$ ++)

$M[i] = A[i]$; //M[i] ← A[i]

4. for ($i = 0$; $i < n$; $i$ ++)

$M[i+m] = B[i]$; //M[i + m] ← B[i]

5. Output: The merged array, *M*.

**Comments:** Here the output array is *M*. The data of the array, *A* are copie to the positions *0* to *m-1* of *M* and the data of the array, *B* are copied to th positions *m* to *m + n-1* of *M*. Here position means index.

Now we see how we can perform some simple operation using array c characters. We can assign values to an array of characters as follows:

```
char test[5]={'a','b','c','d','e'};
We can see output using for loop
for(i = 0;i < 5; ++i)
cout<<test[i];
```

The above statement is equivalent to input some charters to the array. If w want to see the output we can use the code as below.

```
cout<<test;
```

A character array can be assigned values as follows.

```
char name [10]="A Rahim";
```

We can do the same thing in another way. In this case, we have to includ "string" as a header file.

```
char name[10];
strcpy(name, "A Rahim");
We can see separate charter above the array using for loop.
for(i=0; i<10; ++i)
cout<<name[i];
```

Another way to enter data into an array of characters and display them o the screen is as below. Here **gets** is a function that takes character dat

Scanned with CamScanner

including space. To use **gets** function we have to include string.h as a header file.

```
char name[30];
fflush(stdin);
gets(name);
cout<<name;
```

**Problem 2.8**: Find a particular character from an array of characters

**Algorithm 2.8:** Searching a character

1. Input an array of character

```
char test[30] =" Mid term examination";
```

2. Search for the character:

```
for (k=0; k<30; k++)
{
        if(test[k] == 'x')
        {
                cout<< "Found";
                break;
        }
}
```

We can write the above algorithm in another way.

1. Input an array of character

```
char test[30] =" Mid term examination";
```

2. input the character you search for:

```
char ch;
cin>>ch
```

2. Search for the character:

```
for (k=0; k<30; ++k)
{
if(test[k] == ch)
{ cout<<"Found in index"<<k;
break;}
}
```

3. if (k>29) cout<<"Not Found";

**Problem 2.9:** There two very large numbers say each number is 40 digits long. How can we add these two numbers?

Scanned with CamScanner

**Description of Solution:** We can solve this problem using arrays. Tak digits of numbers in two arrays and we have to put results in the third array Each digit of the numbers is an element of the array. Two elements of tw array are added and we check whether there is any carry. If there is a car we will add it to the sum of the next two elements. This method is as like a paper-pencil method of addition.

**Algorithm 2.9:** Algorithm to add two very large numbers.

1. Input the digits of the first numbers in array, a[0 ...n-1];
2. Input the digits of the second number in array, b[0...n-1].
3. Take an empty array c[0.....n]. //size of the array c will be n+1.
4. k = 0;
5. for (i = n-1; i >= 0; i--)

```
{
  c[i+1] = (a[i] + b[i] + k) % 10;
  k = (a[i] + b[i] + k) / 10;
        }
```

6. c[0] = k;
7. Output the array, c[0....n] (Display the data from c[0] to c[n]);

**Problem 2.10:** There two very large numbers say each number is 40 dig long. How can we multiply these two numbers?

**Algorithm 2.10:** Algorithm to multiply two very large numbers.

1. Input the digits of the first numbers in array, a[0 ...n-1];
2. Input the digits of the second number in array, b[0...m-1].
3. Take an empty array c[0...m+n-1].
4. Initialize all the items of c to zeros.
5. for (j = m-1; j >= 0; j--)

```
   {
i)   k = 0;
ii).  for (i = n-1; i >= 0; i--)
   {
   t = a[i]*b[j] + c[i+j+1] + k;
  c[i+j+1] = t % 10;
   k = t / 10;
```

Scanned with CamScanner

```
    }
iii). c[j] = k;
```

6. Output the array, c (Display the data from c[0] to c[m+n-1]).

## 2.3 Pointer and dynamic array

To understand dynamic array, knowledge of pointer is necessary. So, in this section at first, we describe pointer shortly and then dynamic array.

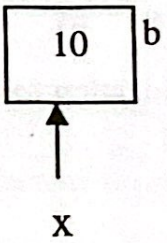
**Pointer:** A pointer is like a variable, but it does not hold actual data. It holds the *address* of the data. It is a variable that holds the *address* of another variable. In other words, a pointer points any other variable. We can declare pointer variable as follows:

```
int *x;
char *ch;
```

To declare pointer variable we use an *asterisk* (*). Here *x* is a pointer variable that points *integer* type data. We can say alternatively *x* is a pointer variable that holds the address of an integer type data. Similarly, ch is a pointer variable that points *character* type data. Let us see examples.

**Example 2.1**: Use of pointer

```
1. int *x;
2. int b;
3. b = 10;
4. x = &b;
5. cout << x;
6. cout << *x;
```




In the above example, *x* is a pointer and b is a variable for integer type data. Since *x* can contain the address, so we assign the address of b to *x* (line 4 in example 1). Line 5 displays address of b and line 6 shows 10 (the data that was assigned to b in line 3).

**Example 2.2**: Use of pointer using **new**

```
1. int *p;
2. p = new int;
3. *p = 12;
4. cout << p;
5. cout << *p;
```

Scanned with CamScanner

Line 4 shows the address and line 5 displays 12 (the data that was assign to p). At line 2 we have allocated space in memory for an integer. Here *ne* is the keyword in c++ for allocating *space* in memory. Without allocati space we cannot assign any value directly using a pointer.

**Example 2.3**: Incorrect use of pointer

```
1. int *x;
2. int p =5;
3. *x = p;
4. cout <<*x;
```

The above code will not give any error during compilation. But it will r execute (run) and shows a run time error when we will try to execute Since we do not assign an address to the pointer *x*, an error occurs he Either we have to assign the address of a variable to the pointer or we ha to allocate space using *new* keyword and assign data.

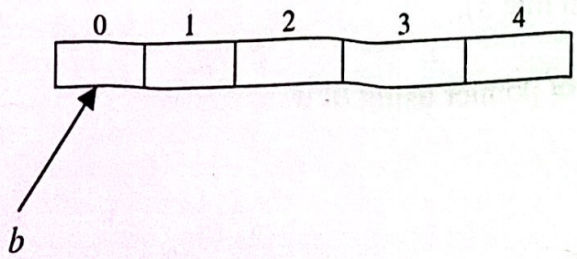
## 2.4 Dynamic array (array using pointer)

A dynamic array can be defined using a pointer and keyword *new*. T declare an array we have to define the size of the array. Without definir the size we cannot declare an array. A dynamic array can be defined a follows:

**Example 2.4:** space allocation for dynamic array.

```
1. int *b;
2. b= new int[5];
```

Here b is a pointer to an integer type data. Using pointer we can alloca space in memory for a number of data items. At line 2 of the above cod we have allocated space for five integers. The graphical view for allocatin space for five integers is as follows:




To store data to a dynamic array the code will be as follows:

Scanned with CamScanner

**Example 2.5**: Store data in a dynamic array

```
int *b, i;
b = new int[5];
for (i = 0; i < 5; i++)
cin >> b[i];
```

### 2.4.1 Increasing size of the array using dynamic array

After defining the size of an array we cannot increase or decrease without redefining it. However, we can increase the size of the array using a dynamic array. Let us consider that using the code of **related to problem 2-5** and after entering data we will get the array with data as follows.

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 15 | 21 | 25 | 19 | 45 |

Figure-2.8: Before increasing size

Now we want to increase the size of the array by one. In the above figure, there are data in the indices 0 to 4. We can assign data to index 5 as below.

```
a[5] = 99;
```

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 15 | 21 | 25 | 19 | 45 | 99 |

Figure-2.9: After increasing size

Now the size of the array is six and it happens dynamically due to the assignment of data in index 5. We can display the data of the array using the following code.

```
for (i=0; i<6; i++)
cout << b[i];
```

Now let us see a problem similar to the **problem 2-5.**

**Problem 2-11**: Given an array with data in all the positions (indices). Now insert a data at a particular index without deleting any data of the array. **Condition:** If we have to insert data in the kth index, the arrangement of data before and after *k*th index must be same as it was. The Figure-2.10 and Figure-2.11 depict the scenario of input array and output array respectively.

Scanned with CamScanner

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 10 | 20 | 30 | 50 | 60 | 70 |

Figure-2.10: Before insertion of data

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 10 | 20 | 30 | **40** | 50 | 60 | 70 |

Figure-2.11: After insertion of data, 40 in index 3.

**Description of Solution:** We shift the data one by one as follows and the problem. Let us see the pictorial view of solving process of the pro

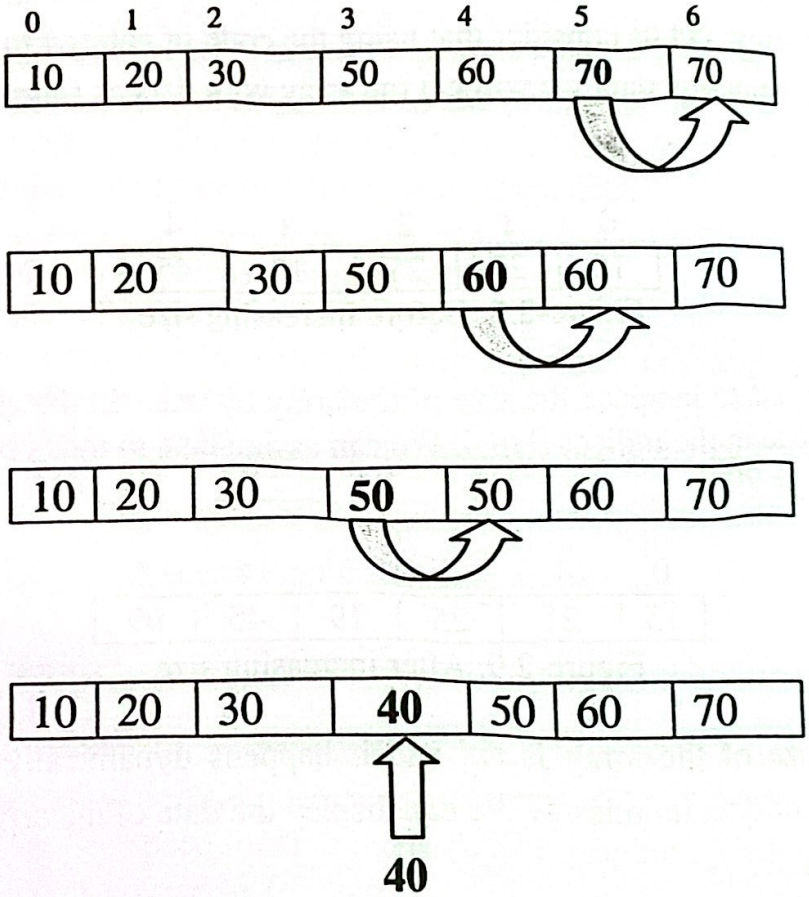



Figure-2.12: Pictorial view to insert a data in a particular positic dynamic array.

We can solve the above problem using a dynamic array of the written below.

**Algorithm 2-11:** Algorithm for inserting data in a particular posi

1. Input:
   a) a dynamic array, d with data in all indices

Scanned with CamScanner

b) index to which the data will be inserted, *k*.
c) data (variable) to be inserted, *x* in *k*th position.

2. Shift the data from *(n-1)*th index to nth index, *(n-2)*th index to *(n-1)*th index and do this until you reach at *k*th index:

```
for (i = n-1; i>=k; i--)
  d[i+i] = d[i];
```

3. d[k] = *x*;
4. output: updated array, d;

Let us see how we can decrease the size of an array using a dynamic array. Just before this, we saw that we can increase the size of the array dynamically or using a dynamic array. However, we cannot decrease the size of the array using only one array. In the case of reduction of the size of the array, we need an assisting array. Let us consider that we have an array of 10 elements. Now we have to decrease the size of the array to 7. Finally, there will be 7 elements in the array. We can do it as follows:

```
int *a, *b;
a= new int [10];
for (i=0; i<10; i++)
cin>> a[i];
for (i=0; i<7; i++)
b[i]=a[i];
delete [ ] a;
a = b;
```

Now the data of array can be easily displayed and there are 7 elements of the array, but not 10.

## 2.5 Set operations using array

In this section, let us see some operations on set using an array. The operations are union, intersection and difference.

Scanned with CamScanner

### 2.5.1 Union of two sets

Let A and B are two sets. We can represent the union of sets A and B as follows:

A U B = {x: x ∈ A or x ∈ B} where x denotes an element of any of these two sets.

If A = {1, 3, 7, 8} and B = {2, 7, 8}
So, A U B = {1, 3, 7, 8, 2}

Another example,

X = {1, 3, 7, 9} and Y = {3, 4, 6}

So, X U Y = {1, 3, 7, 9, 4, 6}

We can represent a set using an array. When we make a union of two sets, the resulting set will be another set, but the size of this set is not fixed or cannot be predefined. So, here we can use a dynamic array for set union operation. We can do a union of two sets using the following algorithm.

**Algorithm 2-12:** Algorithm for union of two sets

1. Input: an array, A[m] with its elements (data) and an array, B[n] with its elements (data)
2. Take an array, C[m+n] with size of A and B.
3. for(i = 0; i < m; i++) // we consider that m >= n.
   C[i]=A[i]; //copy elements of A to C
   p=m;
4. i) for (i=0; i<n; i++) //with the size of smaller array
   {
   flag=0;
   for(j=0;j<m; j++) //with the size of larger array
   {
   if(C[j] = = B[i])

Scanned with CamScanner

```
    {
    flag=1; break;  //for equal items flag =1;
    }
  } //end of j loop
  if(flag==0)
    {
    C[p]=B[i];  //copy item which was not in C before.
     p++  //next index in array, C.
     }
     } //end of i loop
```

5. Output array C with data.

The students can try to implement the above algorithm using a dynami array.

Intersection of two sets: Intersection of sets A and B can be represented a $A \cap B$. We can write

$A \cap B = \{x : x \in A \text{ and } x \in B\}$.

Let $A = \{2, 4, 6, 8, 10\}$ and $B = \{8, 10, 12\}$

$A \cap B = \{8, 10\}$

We can perform intersection operation of two sets using the followin algorithm.

**Algorithm 2-13:** Algorithm for intersection of two sets

1. Input an array A with data and array B with data
2. Take a dynamic array, C and a variable p = -1;
3. for (i=0; i<n; i++)  //with the size of smaller array

```
    {
    for(j=0;j<m; j++)  //with the size of larger array
     {
     if(C[j] = = B[i])
      {
      ++p;
      C[p] = B[i];  break;
     }
```

Scanned with CamScanner

4. Output: array C with data.

Set Difference: Difference of two sets can be represented as follows:

A – B = {x : x ∈ A, x not ∈ B} and
B-A = {x : x ∈ B, x not ∈ A} where A and B two sets.
Let A = {1, 2, 3, 4} and B = {6, 2, 4, 8}
A- B = {1, 3} and B –A = {6, 8}
Algorithm for sets difference operation is left as an assignment for the students.

## 2.6 Two Dimensional Array

### 2.6.1 Definition

*Two dimensional array* is an array that has two dimensions, such as *row* and *column*. The total number of elements in a two dimensional array can be calculated by multiplying the number of rows and the number of columns. If there are *m* rows and *n* columns, then the total number of elements is $m \times n$, and $m \times n$ is the *size* of the array. Of course, the data elements of the array will be same type. In mathematics, the two dimensional array is called a *matrix* and in business, it is called a *table*. It short it is called 2-D array.
The two-dimensional array at a glance:
1) A list of the same type of data times.
2) It has a name.
3) It has a size represented by the number of rows and the number of columns.
4) Rows are numbering as 1, 2, 3,...., m or 0, 1, 2,...., m-1
5) Columns are numbering as 1, 2, 3,..., n or 0, 1, 2,...., n-1
6) Each item is associated with a specific row number and a specific column number.

We will use row numbers as 0, 1, 2,...., m-1 and column numbers as 0, 1, 2,...., n-1.
A two dimensional array can be expressed as follows:

Scanned with CamScanner

$A_{ij}$ or A[$i$, $j$] for $0 \le i \le$ m-1 and $0 \le j \le n-1$ (where $m$ and $n$ are the numbers of rows and columns respectively, see Figure-2.13).

$$A\,[\underbrace{0 \ldots \ \ldots m\text{-}1}_{\text{m rows}},\ \underbrace{0 \ldots \ \ldots n\text{-}1}_{\text{n columns}}]$$

Figure-2.13: Symbolic representation of two dimensional array.

Array B

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 | 2 | 1 | 2 |
| 1 | 56 | 5 | 6 | 7 | 9 | 7 | 8 | 6 |
| 2 | .. | .. | .. | .. | .. | .. | .. | .. |
| 3 | .. | .. | .. | .. | .. | 7 | .. | .. |
| 4 | .. | .. | .. | .. | .. | .. | .. | .. |
| 5 | 2 | 5 | 3 | 7 | 3 | 4 | 4 | 6 |

Size = 6 × 8

Cell B[3][5]

Figure-2.14: Graphical representation of a two dimensional array

An element represented as B[1][2] indicates that B is the name of the array, the row number is 2 and column number is 3. If we notice **Figure-2.14,** the item B[1][2] is 6. Thus any item can be found in cross-point of the row number and column number.

Two dimensional array can be expressed in C/C$^{++}$ as follows:

int A[$m$][$n$]; // Here $m$ is the number of rows and $n$ is the number of columns [see Figure-2.15]

```
float  B[5][7];
```

Array Name  No. of rows  No. of columns

Figure-2.15: Declaration of a two dimensional array in C/C++

Scanned with CamScanner

The size of this array is 6 × 8. So, the total number of elements or items is 6×8=48.

### 2.6.2 Store into and retrieve values from 2-D array

As we know in C/C++ indexing start at 0(zero), so we have to store data considering index number zero. Data elements can be stored in a two dimensional array using loop or directly as shown below:

```
int B[7][3];
for (int i = 0; i < 7;  i++)
{
  for (int j = 0; j < 3; j++)
      cin >> B[i][j]);
        }
```

//storing data taken from keyboard

Another way to store data:

```
int B[7][3] =  {
        { 1,  2,  3},
        { 9, 10, 11},
        ... .... ....,
        ... .... ....,
        ... .... ....,
        {22, 25, 40}
        };
```

//Direct insertion of elements

### 2.6.3 Assigning a value to an item

B[3][4] = 25; Here the item represents in cross-section of index 3 (row 4) and index 4 (column 5) is assigned a value 25.

Scanned with CamScanner

### 2.6.4 Reading (Retrieving) an item

y = B[4][2]; The item represented by row number 5 (index 4) and column number 3 (index 2) is assigned to a variable y. If the value of B[4][2] is 85, this value is assigned to the variable y, consequently, the value of y will be 85.

### 2.6.5 Two dimensional array representation in memory

The elements of a two dimensional array are stored in computer's memory ***row by row*** or ***column by column***. If the array is stored as *row by row*, it is called ***row-major order*** and if the array is stored as a *column by column*, it is called ***column-major order***. Suppose that there is a two-dimensional array of size 5 × 6. That means, there are 5 rows and 6 columns in the array.

In *row-major order*, elements of a two dimensional array are ordered as belwo.

$$\underbrace{A_{11}, A_{12}, A_{13}, A_{14}, A_{15}, A_{16}}_{\text{row 1}}, \underbrace{A_{21}, A_{22}, A_{23}, A_{24}, A_{25}, A_{26}}_{\text{row 2}}, A_{31}, \ldots, A_{46}\ldots, \underbrace{A_{51}, A_{52}, \ldots\ldots, A_{56}}_{\text{row 5}}$$

and in *column-major order*, elements are ordered as

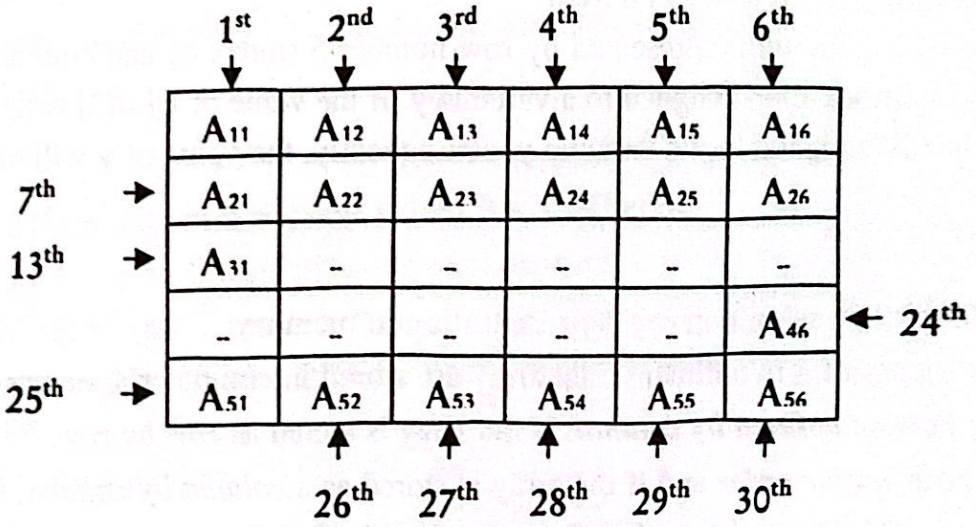
$$\underbrace{A_{11}, A_{21}, A_{31}, A_{41}, A_{51}}_{\text{column 1}}, \underbrace{A_{12}, A_{22}, A_{32}, A_{42}, A_{52}}_{\text{column 2}}, A_{13}, \ldots\ldots\ldots\ldots, A_{55}, \underbrace{A_{16}, A_{26}, \ldots\ldots\ldots, A_{56}}_{\text{column 6}}$$

Figure-2.16: shows these arrangements in details.

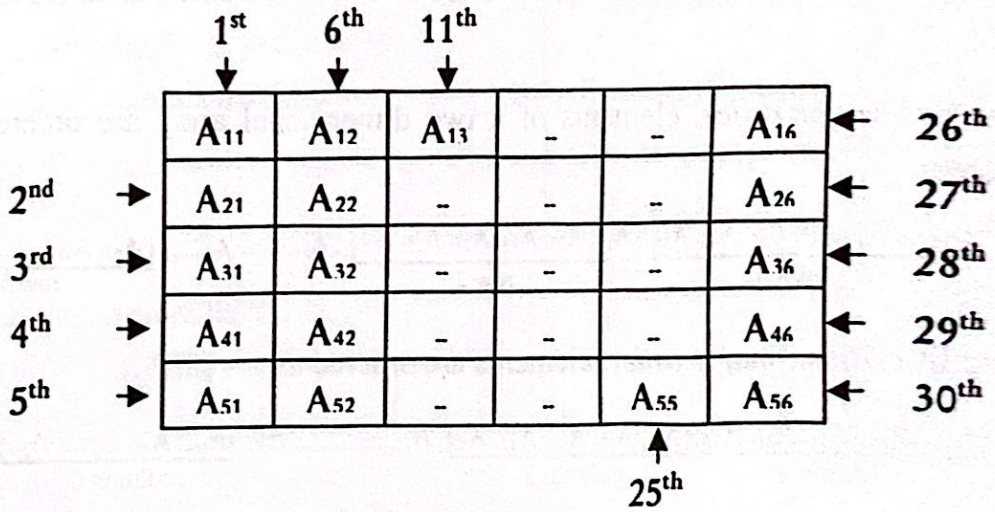
In row-major order all the rows make a long row (one dimensional array) in the memory. Similarly, in column-major order all the columns make a row (one dimensional array) in the memory. In figure 2.17 1st, 2nd, 3rd etc denote cell numbers of the elements in the array.



Scanned with CamScanner




(a) Row-major order




(b) Column-major order

Figure-2.17: Pictorial view of a two dimensional array representation in memory

### 2.6.6 Location of an element of a two-dimensional array

***Row-major Order*:**

Let Loc (A[*i*, *j*]) be a location in the memory of the element A[*i*][*j*] or $A_{ij}$, then in row-major order

$$\text{Loc}(A[i, j]) = \text{Base}(A) + (n(i - 1) + (j - 1)) * w;$$

Here Base (A) is starting or base address of the array A, *n* is the number of columns and w is the width of each cell, *i.e*, number bytes per cell.

Scanned with CamScanner

***Column-major Order*:**

In column-major order,

$$\text{Loc}(A[i, j]) = \text{Base}(A) + (m(j - 1) + (i - 1)) * w;$$

Here Base (A) is starting or base address of the array A, *m* is the number of rows and w is the cell width.

**Example:**

Base address, Base (A) = 100, Size of the array = 5 × 6. If the type of array is integer then find out Loc (A[4, 3]).

**Solution:**

If the array is stored in row-major order:

$$\text{Loc}(A[4, 3]) = \text{Base}(A) + (n(i - 1) + (j - 1))* 4$$
$$= 100 + (6 \times 3 + 2)* 2$$
$$= 100 + 40$$
$$= 140$$

(4 bytes for each integer cell in C/C++)

If the array is stored in memory in column-major order:

$$\text{Loc}(A[4, 3]) = \text{Base}(A) + m(j - 1) + (i - 1)* 4$$
$$= 100 + (5 \times 2 + 3)* 2$$
$$= 100 + 26$$
$$=126$$

**Problem 2.12:**

Given a two-dimensional array, find out the summation of the boundary elements of the array. Here no element should be added twice.

**Description of Solution:** First, we have to identify the boundary elements In a two dimensional array, elements of the first and the last columns an the first and last rows are the boundary elements as shown in the Figure 2.18.

Here the index, *i* represents the row number and *j* represents the colum number. When *i* is 0, it indicates the first row and when *i* is m-1 (where

Scanned with CamScanner

represents the number of rows in the array), if is the last row. Similarly when $j = 0$, it indicates the first column and the index of the last column $j = n - 1$. So, we get the number of boundary elements if $i = 0$, $i = m-1$, $j =$ or, $j = n-1$.

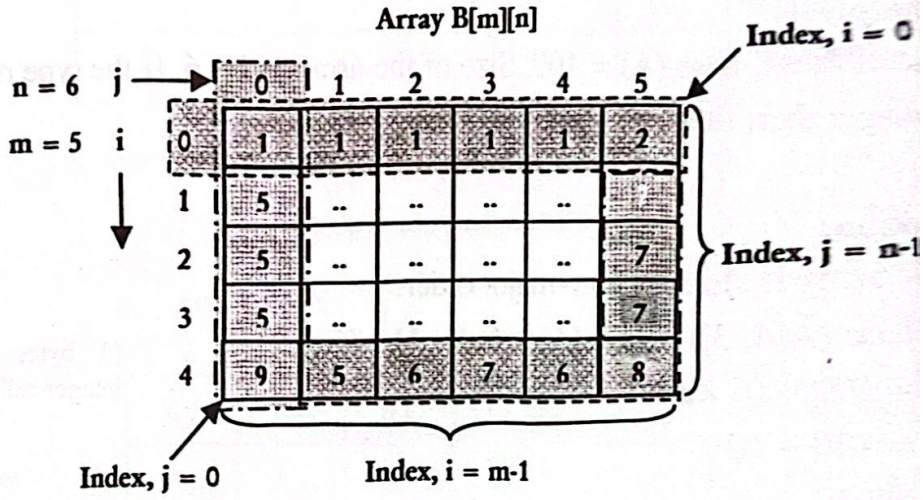



Figure-2.18: Boundary elements of a two-dimensional array

**Algorithm 2.14:** Algorithm to find the summation of boundary element

1. Input: a two-dimensional array

   A[0...*m-1*, 0...*n-1*]

   *sum* = 0;

2. Find the boundary element

```
for (i = 0; i < m; i++1)
    for (j = 0; j < n; j ++1)
        if (i = =0 || j == 0 || i = =m-1 || j = -1), sum = sum + A[i][
```

[Boundary elements are those elements whose index $i = 0$ or $j = 0$, whose index $i = m-1$ or $j = n-1$) and add it with *sum* (previous result)]

3. Output: Print *sum* as a result of the summation of boundary elements

Scanned with CamScanner

Given a two-dimensional array find out the summation of the diagonal elements of the array.

**Description of Solution:** First, we have to identify the diagonal elements. A diagonal element is one, whose either row index and column index are equal or the summation of row index and column index is equal to n - 1 where *n* is the number of rows or columns [here, the number of columns and number of rows are same].

In the Figure-2.19, shaded elements are diagonal elements. The column indices of all elements of the first diagonal (from the upper left corner to the lower right corner) are equal to their corresponding row indices. On the other hand, in second diagonal (from the upper right corner to the lower left corner) the summation of the row index and the column index is n - 1 for all elements.

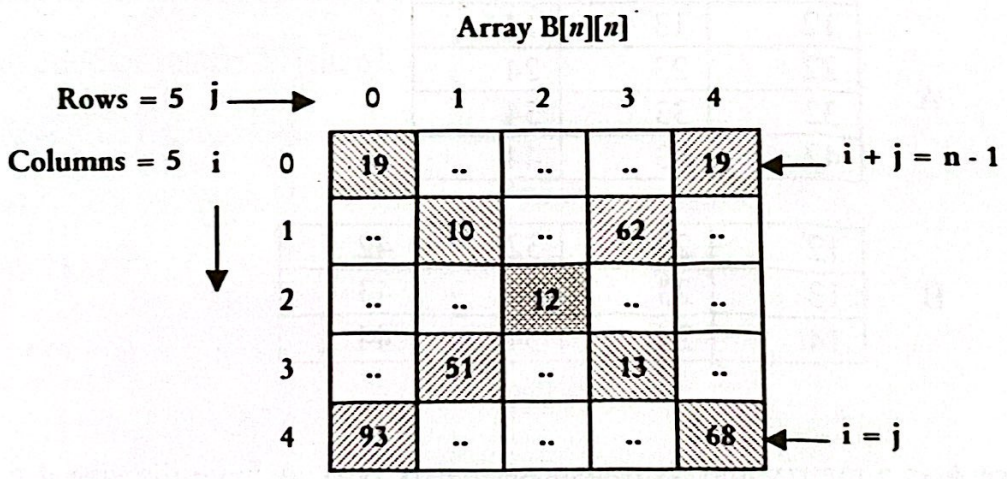



Figure-2.19: Diagonal elements of a two-dimensional array.

**Algorithm 2.15:** Algorithm to find out summation of diagonal elements

1. Input: a two dimensional array

   B[0 . . . *n-1*, 0...*n-1*]

   *sum* = 0;

Scanned with CamScanner

2. Find the diagonal element and add them with *sum*.

```
for (i = 0; i< n; i ++)
        for (j = 0; j <n; j ++)
        if (i = =j || i + j = =n -1), sum = sum + B[i][ j] ;
}
```

[Note: Diagonal elements are those elements whose indices are equal (*i.e.*, $= j$) or their summation results is n -1 (*i.e.*, $i + j == n-1$).]

3. Output: Print ***sum*** as a result of the summation of diagonal elements.

**Problem 2-13**: Find out the transpose of a matrix.

**Description of Solution**: There is a matrix (two-dimensional array), A. W can find out transpose of the matrix by converting the elements of each ro as the elements of the respective column of the matrix. If the size of t matrix is m × n, the size of the transpose matrix will be n × m. Let us s the following example.

A

| | | |
|---|---|---|
| 12 | 13 | 14 |
| 22 | 23 | 24 |
| 32 | 33 | 34 |
| 42 | 43 | 44 |

B

| | | | |
|---|---|---|---|
| 12 | 22 | 32 | 42 |
| 13 | 23 | 33 | 43 |
| 14 | 24 | 34 | 44 |

Here A is a matrix and its transpose matrix is B. A have the size 4 x 3 a the size of the matrix B is 3 x 4.

We can find transpose of a matrix using the following algorithm.

**Algorithm 2-16: Algorithm to find out the transpose of a matrix.**

1. Input a matrix A[m][n] with data

2. Take a matrix B [n][m].

3. for (i=0; i<m; i++)

```
    for(j=0; j<n; j++)
    {
```

Scanned with CamScanner

```
    B[j][i] = A[i]lj];  //row to column and column to row
    }
  }
```

4. Output matrix B with data

**Problem 2-14**: Multiply two matrices and find a product matrix.

**Algorithm 2-17: Algorithm for multiplication of two matrices.**

**Description of Solution**: For multiplying two matrices there is a condition, the number of columns of the first matrix must be equals to the number of rows of the second matrix. If there are two matrices A with size m × n and B with n × p, the size of the product matrix will be m × p.

**Algorithm 2-18**: Algorithm to find the product of two matrices

1. Input two matrices A [m, n] and B[n, p] with data
2. Take another matrix C [m, p].
3. i) for (i = 0; i < m; i++)

   ii) for (j = 0; j < p; j++)

   C[i, j] =0;

   iii) for (k = 0; k < n; k++)

   C[i, j] = C[i, j] + A[i, k]*B[k, j];
4. Output: matrix C of size m × p with data

**Problem 2.15:** There are 40 students in a class. They have written 4 class tests of a course. Find out the average of the best 3 class tests for each student.

**Description of Solution:** First, we compute the summation of marks of 4 class tests for each student. Then we find minimum marks among 4 class tests. By subtracting the minimum marks from the summation and dividing by 3, we get an average for each student.

Scanned with CamScanner

**Algorithm 2.19 : Find average of best three class tests**

1. Input: a two-dimensional array (to store the marks of class te
   of each student) and a one-dimensional array (to store t
   average of marks of each student)
   *marks*[40][4]], *avg_mrk*[40] ;

2. Add all class test marks for each student

```
for (i = 0; i < 40; i ++)     // first loop starts
    {
    sum = 0;
    min_mrk = marks[i,1];
    for (j = 0; j <4; j ++)        //second loop starts
        {
        sum = sum + marks[i,j];
```

3. Find out the minimum (worst) class test marks

```
if (min_mrk > marks[i][j]), min_mrk = marks[i][j];
        }          // end of the second loop
```

4. Compute the average marks of best three class tests.

```
avg_mrk[i]= (sum − min_mrk)/3;
    }                          // end of the first loo
```

5. Output: Average of best three class test marks, *avg_mrk*[40];

**Comments:** Here ***marks*[40, 4]** is a two-dimensional array that holds mar of four class tests for each of the 40 students and ***avg_mrk*[40]** is an array store the computed average marks of best three class tests. The ***sum*** is variable to store the summation of all (4) class test marks and ***min_mrk*** is variable to store the lowest mark of four class tests for each student.

For each time, to calculate the summation of all class test marks and identify the lowest class test ***mark*** of a particular student the variables su and ***min_mrk*** are initialized within the loops.

Like the linear array of pointers we can use 2-D array of pointers also. this case, we need some rows of pointers and each of these pointers has point some columns of pointers. In **Fig. 2-20,** we have shown four rows pointers and each pointer points five columns of pointers. Using this arr we can store 20 (4x5) elements. Using pointers we can use variab

Scanned with CamScanner

numbers rows and columns in the array and this type of 2-D array is called dynamic 2-D array.

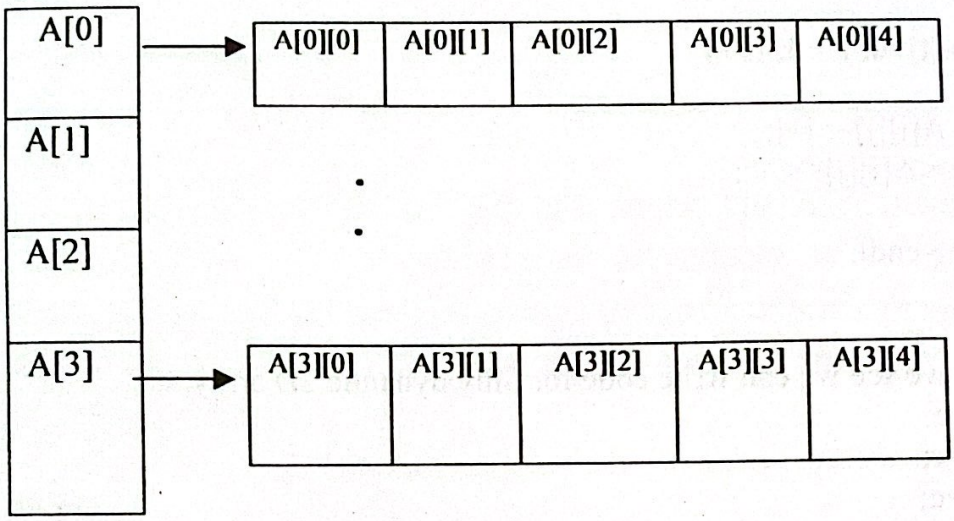



Figure-2.20: Pictorial view of a 2-D dynamic array

Let us see how to declare a 2D pointer array. For the above figure, we can write pseudo-code for declaration as follows.

```
int *A[4];
for (i=0;i<4;i++)
A[i] = new int [5];
```

The first line indicates an array of four pointers and the second and third lines are for five columns for each pointer. The above array, A is not fully dynamic, since here the number of rows should be a constant, but number columns may be variable. So, we can write similar code as

```
int *A[4];
int c, cin >> c
for(i=0;i<4;i++)
A[i]=new int [c];
```

We can use any value of c to get a variable number of columns.

To see how it will work we can use the following code.

```
int *A[4];
```

Scanned with CamScanner

```
int i,j;
for (i=0;i<4;i++)
A[i] = new int [5];
for (i=0; i < 4; i++)
{
   for (j=0; j < 5; j++)
   {
      A[i][j] = i+j;
cout<<A[i][j]<<" ";
   }
cout<<endl;
   }
```

Now we see we can write code for fully dynamic 2D array.

```
int r, c;
cin>>r;
cin>>c;
int  **b=new int*[r];   //r is the number of rows
for (i=0;i<r;i++)
b[i] = new int [c];    //each row has c columns.
```

Let us see the problem (summation of boundary elements). Using dynamic array we can test this problem for different sizes of arrays. Followings are codes for the problem.

```
          int r,c;
          cout<<"Enter number of rows:";
          cin>>r;
          cout<<"Enter number of columns:";
          cin>>c;
int **b=new int*[r]; //r rows

int i, sum=0;
for (i=0;i<r;i++)
b[i] = new int [c]; //c columns
for (i=0; i < r; i++)
{
   for (int j=0; j < c; j++)
   {
      b[i][j] = i+j;
      cout <<b[i][j] << " ";
                  if(i==0||j==0||i==r-1||j==c-1)
```

Scanned with CamScanner

```
                    sum=sum+b[i][j];
  }
  cout << endl;
}

cout<<"sum="<<sum;
```

**Summary:**

**Array** is a collection of homogeneous data items. The array, which is represented by only one dimension (row or column) is called **one dimensional** array, whereas an array represented by two dimensions such as row and column is known as **two dimensional** array. In any array same type of elements are used.

**Questions:**

1. What is a linear array? Explain with example.
2. Write an algorithm to find out a particular item from a given list of items stored in a linear array.
3. Write an algorithm to insert an item in a particular position of an array.
4. Given a list of elements stored in an array, write an algorithm to find out the largest element from the array.
5. You are given an array of sorted data in ascending order. Write an algorithm to rearrange the elements of the array in descending order without using any sorting algorithm and extra data structure.
6. What is a linear array? Write an algorithm to delete an item from a linear array.
7. Define two-dimensional array.
8. Describe how a two-dimensional array can be stored in computer memory.
9. How can address of a particular element of a two dimensional array be computed? Explain.
10. Write an algorithm to find out the summation of the diagonal elements of a two dimensional array.

Scanned with CamScanner

11. Given an array of size 35 × 40. The base address of the array is 1000. Find the address of A[18][32]. Write down the formula first, then calculate.
12. Suppose there is an array A of size 40×30. The base address of the array is 100. Calculate the address of A[23][15].
13. Given a two dimensional array with m × n size. Write an algorithm to find out summation of the boundary elements, where no element will be added twice.
14. Describe when a two-dimensional array is required to use in programming or Algorithm.
15. Write an Algorithm to print out the following triangle.

```
1
2  3
4  5  6
7  8  9  0
1  2  3  4  5
```

16. What is a string? Write a procedure to count the number of a given substring occurs in a given string.

## Problems for Practical (Lab) Class

### Array related problems

**Problem 2-1:** Write a program to store 10 integers using an array and find out the summation and average of the numbers. Display the numbers, their sum, and average (with points) separately.

**Problem 2-2:** Write a program to find out the largest and smallest of a given list of numbers in an array. Display the numbers, the largest and smallest separately.

**Problem 2-3:** Write a program to find out the summation of even number and odd numbers from a given list of numbers in an array. Display the numbers, the summation of even numbers and odd numbers separately.

Scanned with CamScanner

**Problem 2-4:** Write a program to find out the summations of the numbers stored in even indices and odd indices **of an array**. Display the numbers, the two summations separately.

**Problem 2-5:** Write a program to merge two arrays (merge two arrays and make single one). Display the data of the three arrays.

**Problem 2-6:** Given an array with data in all positions. Write a program to insert an item in a particular position of an array in such a way that no data will be lost and data will be in previous order. Hints: insert an element, $x$ in $i$th position where $i < n$ and $n$ is the size of the array. Increase the size of the array, shift all the elements starting from position i, and insert the element in free position.

**Problem 2-7:** Given two arrays of the same size with data (integers) arranged in ascending order. Create the third array that will contain the data of the first two arrays and the data will be arranged in ascending order. Condition: write the program without using any sorting procedure. Hints: compare and merge (write) in the third array without using any sorting procedure.

**Problem 2-8:** write a program to find out the summation of boundary elements of a two-dimensional array.

**Problem 2-9:** write a program to find out the summation of diagonal elements of a two-dimensional array.

**Problem 2-10:** There are 10 students in a course. They have written 4 quizzes for the course. Write a program to find out the average of best three quizzes for each student of the course.

**Problem 2-11:** Take a 2-D array of 3*4 and find out the transpose matrix as output.

**Problem 2-12:** Take a 2-D array of 4*4 (square) and find out its transpose matrix. There should be some changes in the program, which is for 2-11.

**Problem 2-13:** Implement **algorithm 2-18** for A[4, 3] and B[3, 4]. Find out the number of multiplications needed to implement the algorithm.

Scanned with CamScanner

**Problem 2-14:** There are three matrices as follows:

A [5, 4], B[4, 3] and C[3, 6]. Now write a function that will multiply two matrices and using this function find out the multiplication of three matrices. You can find the multiplication in two ways: i) (A.B).C ii) (B.C)A

Now find out the number of multiplications of both ways and show that which way gives less number of multiplications.

**Problem 2-15:** Find out the first largest and second largest from a list of numbers. Take 10 integers to execute your program.

**Problem 2-16:** There are 15 numbers in an array. Among them, there are more than one duplicate numbers. Find out the duplicate numbers from the array.

**Problem 2-17:** There is a 2-D array of size m × n. We will say a point as a min-max point if there is a number a[i, j] which is smallest in the ith row and largest in the jth column. Write a program to find out the min-max point as <i, j> and the data in the point. Take the following data set and test your program.

8 13 10
9 11 12
5 4 10

**Problem 2-18:** Take an expression such as (125 + 20 -75) in an array. Copy (put) all the numbers in an array and the +, - in the *third* array. Can you find the results of the expression?

**Problem 2-19:** Take two numbers of twenty digits each and multiply them. Display the numbers and result **(See algorithm 2-18).**

**Problem 2-20:** Take two numbers of twenty digits each and divide them. Display the quotient and remainder. After getting results multiply divisor and quotient using **algorithm 2-18**. Add the result of multiplication and remainder using **algorithm 2-18**. Now verify whether your final result is equal to dividend or not.

Scanned with CamScanner