Data Structures and Algorithms I SCS1201 - CS

Dr. Dinuni Fernando

Senior Lecturer

Learning Objectives of the course

LO1: Understand the different data structures and their practical application.

LO2: Understand Basic stack operations, practical usage and their implementation.

LO3 : Understand Basic Queue operations, practical usage and their implementation.

LO4 : Understand, different types of Linked Lists (singly , doubly, circular and with and without header nodes) and their implementation.

LO5: Understand and implement the selected applications of general trees, Binary trees, Binary Search trees and AVL trees.

LO6: Understand and implement the basic sorting algorithms such as exchange sort(bubble sort), Insertion sort and Selection Sort.

LO7 : Understand and implement Merge, Shell, Heap, Quick, Straight Radix, and Radix Exchange sort .

LO8 : Understand the running times of different algorithms .

Learning Objectives of the lesson

- Understand the What is data structure and how its being used.
- Differentiate types of data structures
- Understand and differentiate different types of data structures namely Arrays, lists,….

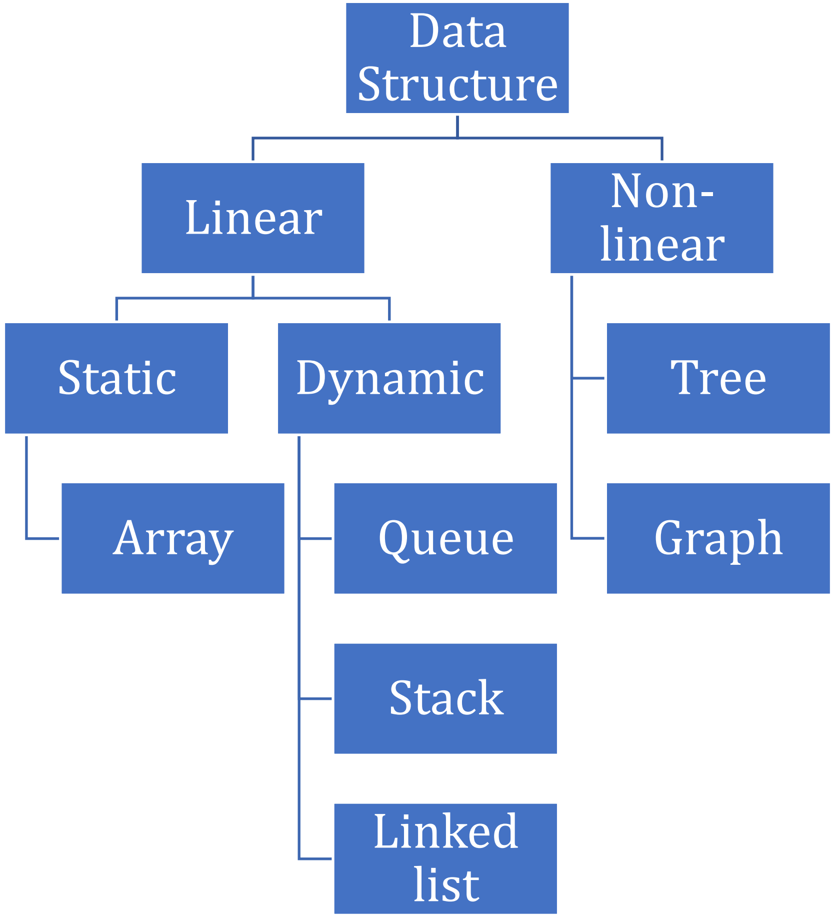
What is a Data Structure ?

What is a data structure ?

- What - Is a way of organizing and storing data in a computer program.
  - Why - To access and use efficiently.
  - Goal – to reduce space and time complexity of different tasks.
  - How – helps to manage large amounts of data, enabling efficient searching, sorting and deletion of data.

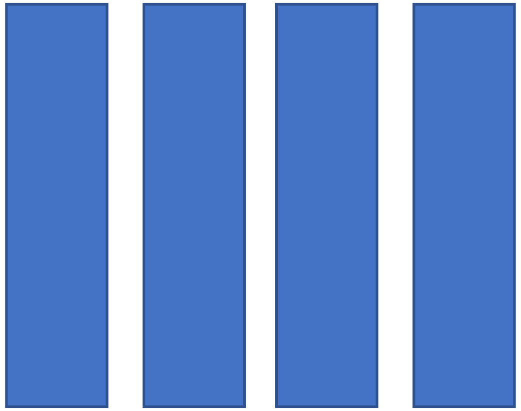
  - Objective – Choice of a good data structure perform variety of operations effectively.
    - Efficient data structure uses minimum space and execution time to process the structure.

Classification of Data Structure

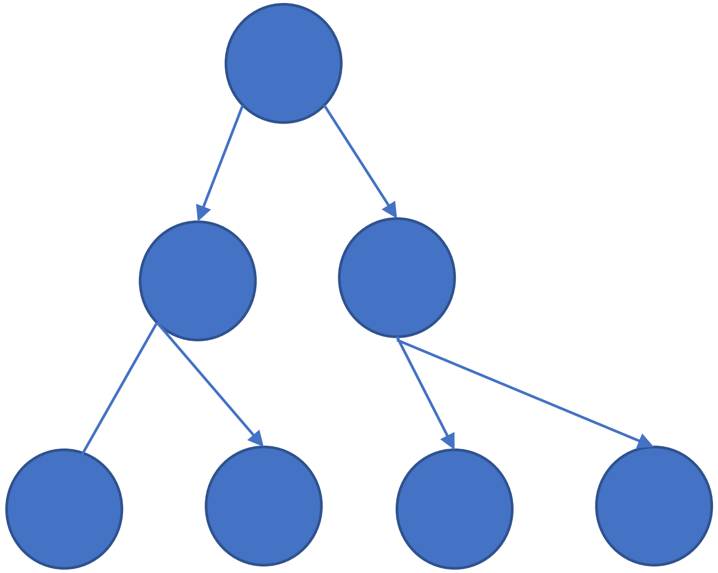


Linear / Non-Linear Data Structure

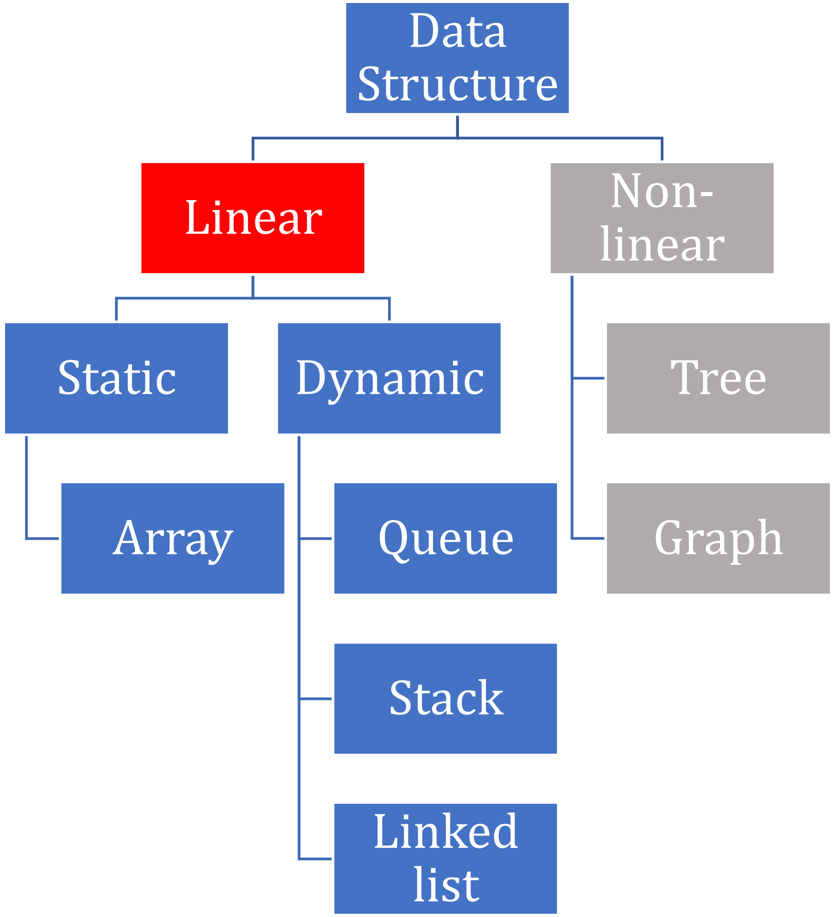
- Linear Data structure



- Non-linear Data structure

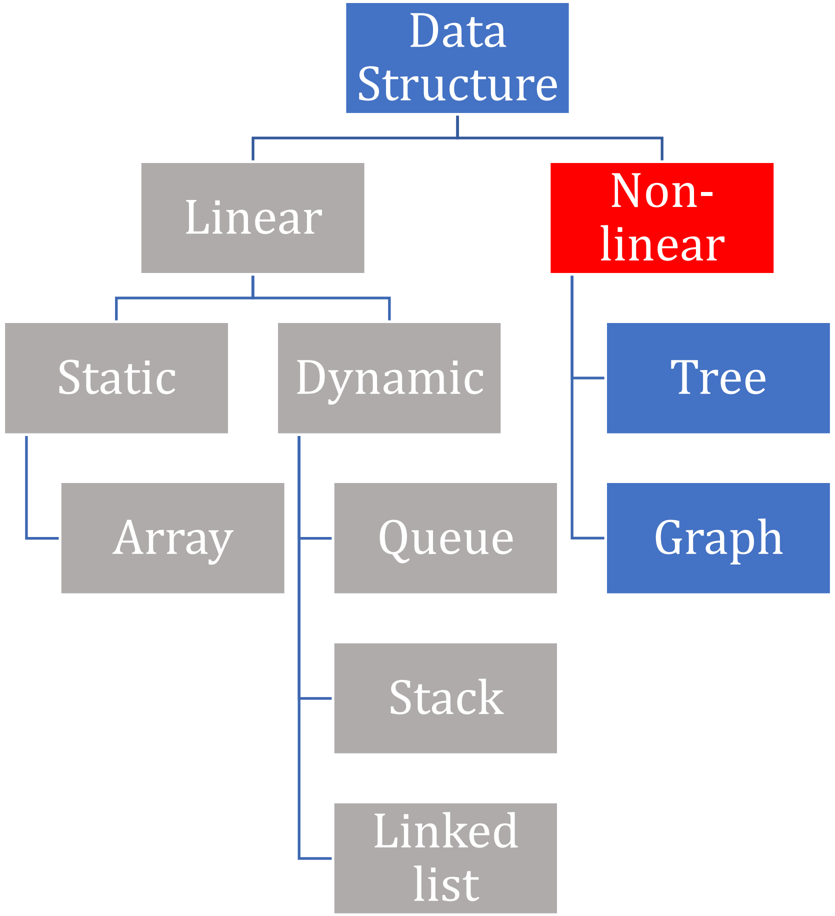


Linear Data Structure

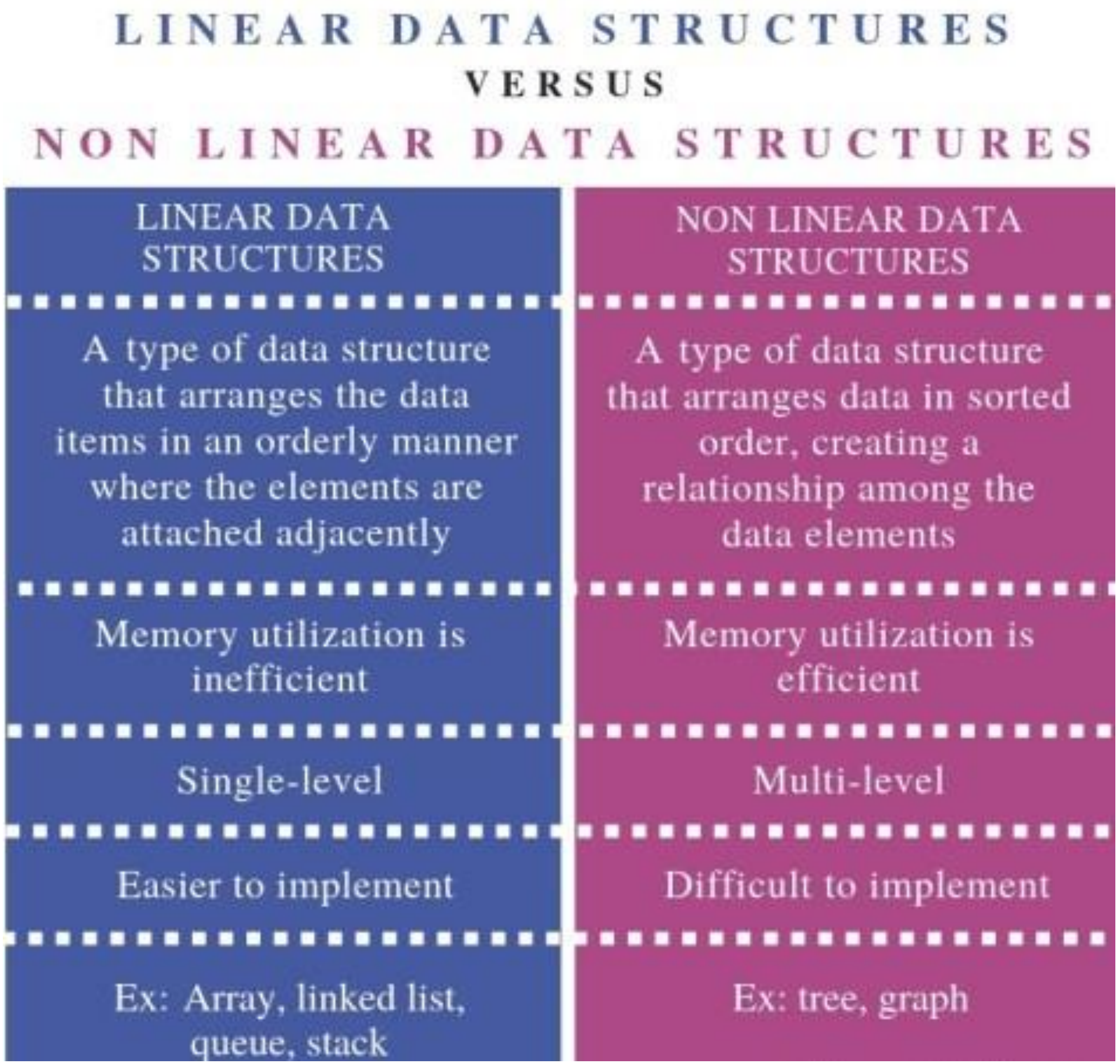


- Data elements are arranged in a sequential order.
- Each element has a unique predecessor and successor except first and last elements.
- One/linear dimensional.
- Able to traversed sequentially from the first to the last element.
- Eg: Lists, stack, queue

Non-Linear Data Structure



- Data elements are **not** arranged in a sequential order.
- Each element may have one or more predecessor and successors.
- Elements are arranged in one-many, many-one and many-many dimensions.
- Eg: tree, graph, table



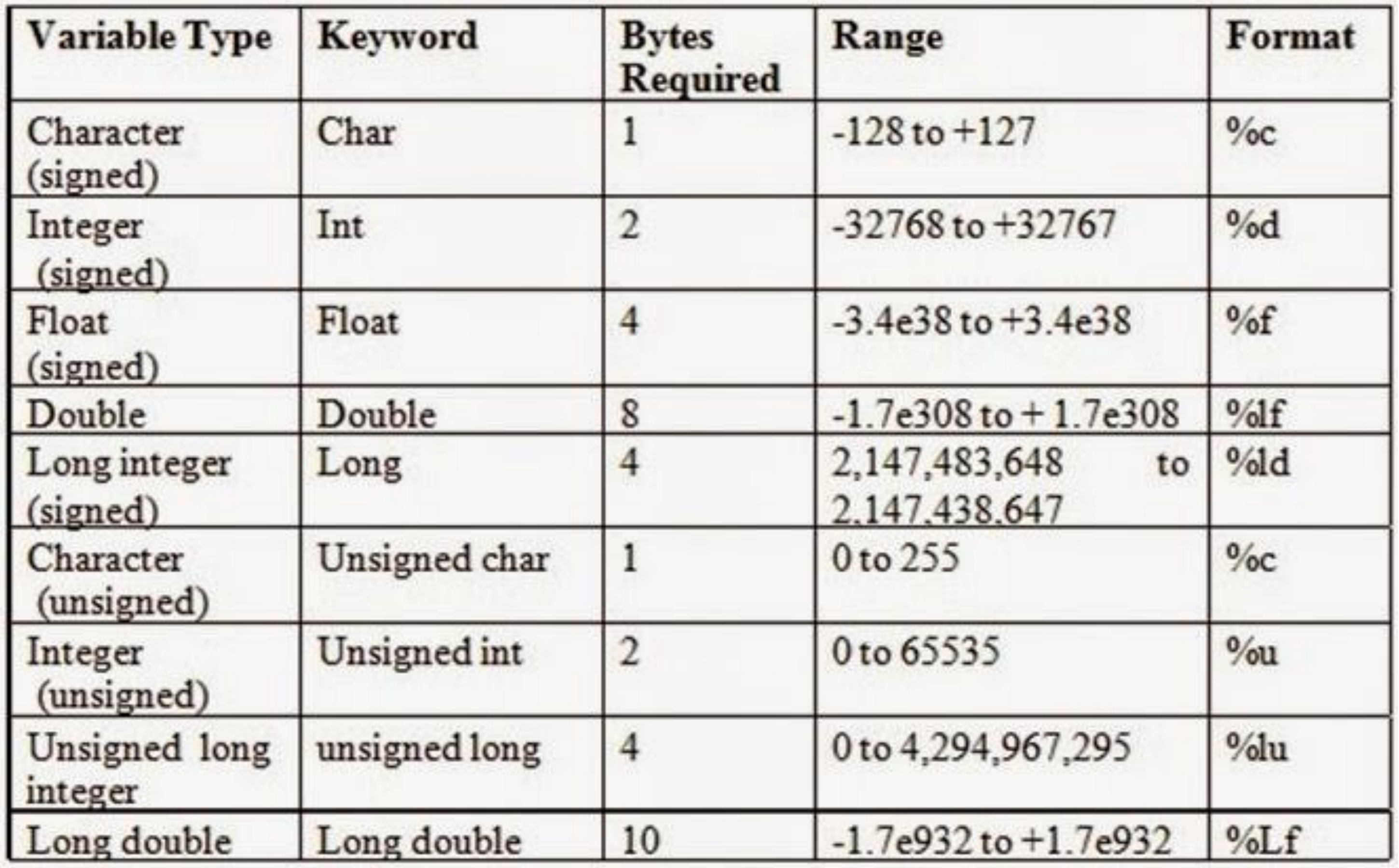
LINEAR DATA STRUCTURES VERSUS NON LINEAR DATA STRUCTURES

LINEAR DATA STRUCTURES	NON LINEAR DATA STRUCTURES
A type of data structure that arranges the data items in an orderly manner where the elements are attached adjacently	A type of data structure that arranges data in sorted order, creating a relationship among the data elements
Memory utilization is inefficient	Memory utilization is efficient
Single-level	Multi-level
Easier to implement	Difficult to implement
Ex: Array, linked list, queue, stack	Ex: tree, graph

Source : Pediaa.com

Background

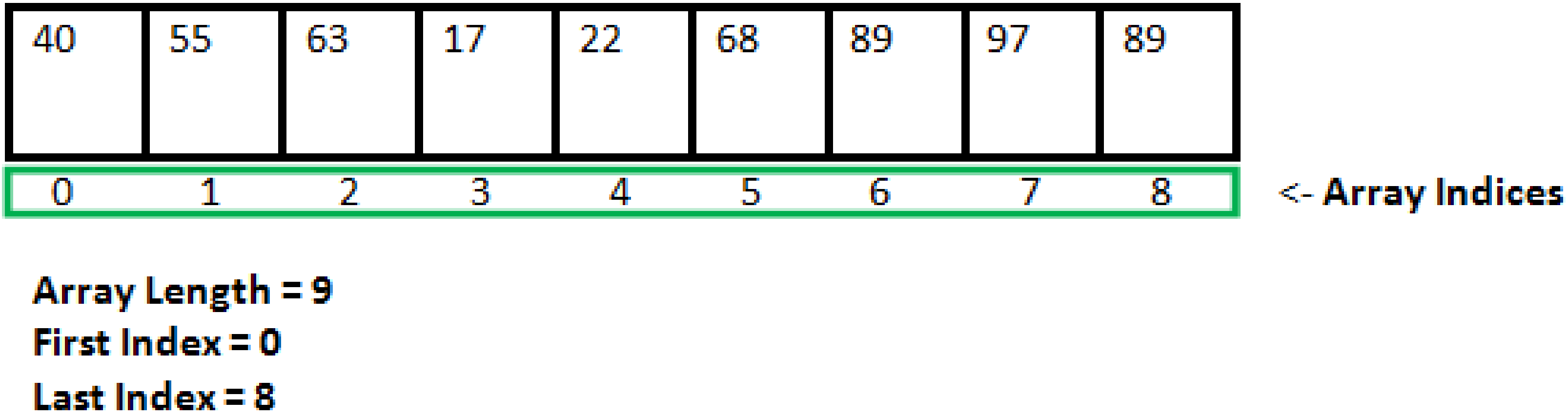
- Variable - placeholder to store basic unit of data.
- Data types



Variable Type	Keyword	Bytes Required	Range	Format
Character (signed)	Char	1	-128 to +127	%c
Integer (signed)	Int	2	-32768 to +32767	%d
Float (signed)	Float	4	-3.4e38 to +3.4e38	%f
Double	Double	8	-1.7e308 to + 1.7e308	%lf
Long integer (signed)	Long	4	2,147,483,648 to 2,147,438,647	%ld
Character (unsigned)	Unsigned char	1	0 to 255	%c
Integer (unsigned)	Unsigned int	2	0 to 65535	%u
Unsigned long integer	unsigned long	4	0 to 4,294,967,295	%lu
Long double	Long double	10	-1.7e932 to +1.7e932	%Lf

Arrays

- Fixed size collection of similar data items
- Stored in contiguous memory locations in RAM
- Can used to store
    - Primitive data types eg: int, char, float …etc
    - User defined data type – pointers, structures,…etc



40	55	63	17	22	68	89	97	89
0	1	2	3	4	5	6	7	8

<- Array Indices

Array Length = 9
First Index = 0
Last Index = 8

Primitive variable Vs. collection of variables

- Two methodologies can be used:

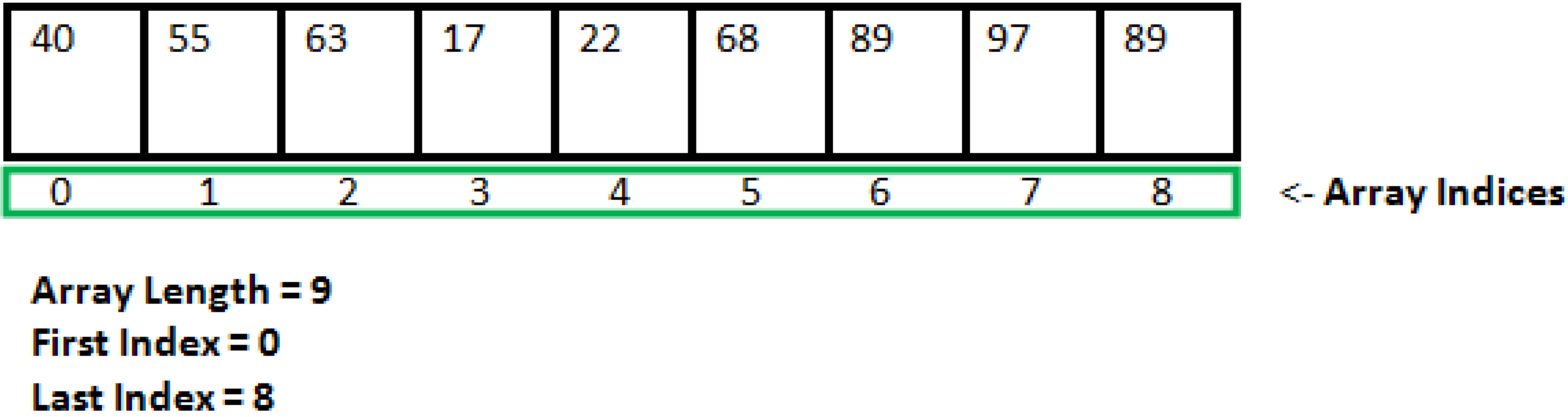
1: Create different variables each having a different name.

E.g. int num1, num2, num3 etc.

2: Create a collection of variables referred by a common name.

E.g. int num[3]

Array representation



40	55	63	17	22	68	89	97	89	
0	1	2	3	4	5	6	7	8	<- Array Indices

Array Length = 9

First Index = 0

Last Index = 8

Element : each item stored in array

Index : Each location of an element in an array has a numerical index, which is used to identify the element.

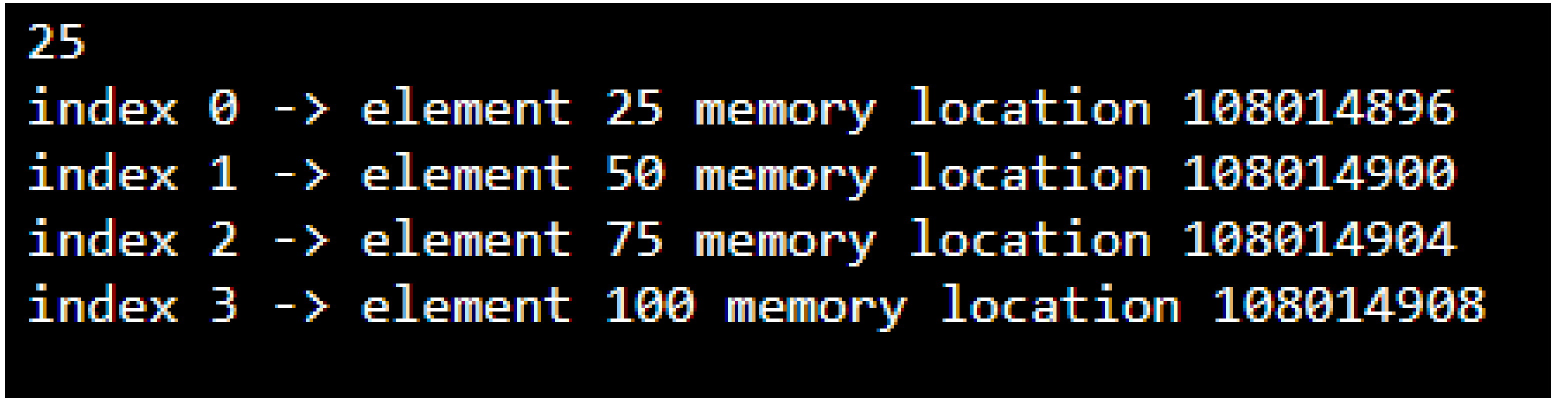
Example 1

```
#include <stdio.h>

int main() {
 int myNumbers[] = {25, 50, 75, 100};
 printf("%d\n" , myNumbers[0]);

 printf("index 0 -> element %d memory location %d\n",myNumbers[0], &myNumbers[0]);
 printf("index 1 -> element %d memory location %d\n",myNumbers[1], &myNumbers[1]);
 printf("index 2 -> element %d memory location %d\n", myNumbers[2],&myNumbers[2]);
 printf("index 3 -> element %d memory location %d\n",myNumbers[3], &myNumbers[3]);
  return 0; }
```

What is the output ?

Example 1

```c
#include <stdio.h>

int main() {
 int myNumbers[] = {25, 50, 75, 100};
 printf("%d\n" , myNumbers[0]);

 printf("index 0 -> element %d memory location %d\n",myNumbers[0], &myNumbers[0]);
 printf("index 1 -> element %d memory location %d\n",myNumbers[1], &myNumbers[1]);
 printf("index 2 -> element %d memory location %d\n", myNumbers[2],&myNumbers[2]);
 printf("index 3 -> element %d memory location %d\n",myNumbers[3], &myNumbers[3]);
  return 0; }
```

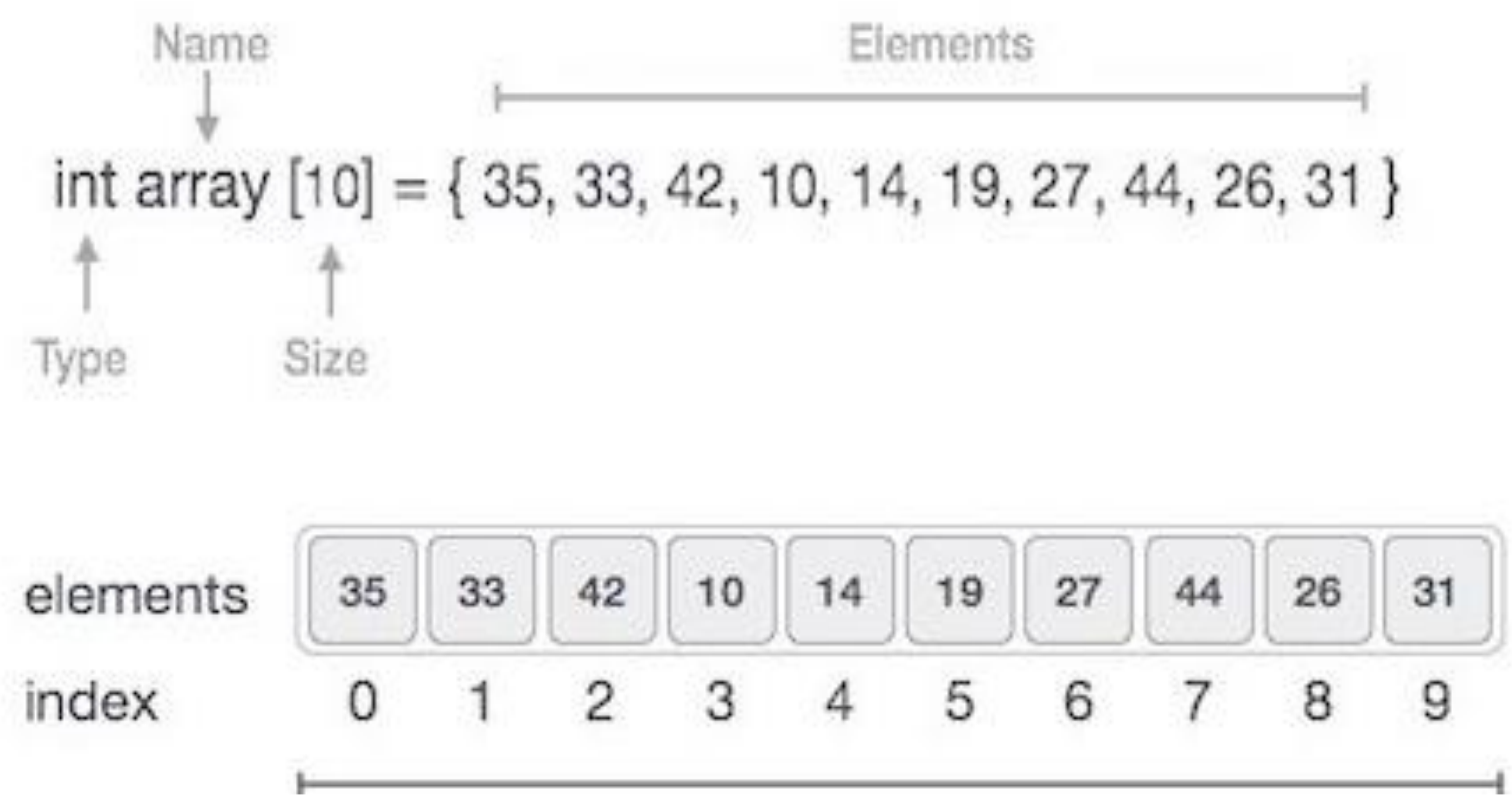


```
25
index 0 -> element 25 memory location 108014896
index 1 -> element 50 memory location 108014900
index 2 -> element 75 memory location 108014904
index 3 -> element 100 memory location 108014908
```

Array Representation

Declaration

Illustration



```
int array [10] = { 35, 33, 42, 10, 14, 19, 27, 44, 26, 31 }
```

elements	35	33	42	10	14	19	27	44	26	31
index	0	1	2	3	4	5	6	7	8	9

Array characteristics

- Finite : contains only a finite (limited) number of elements
- Ordered : all elements are stored in one continuous chunk of memory
- Homogeneous : all elements of array are of same data type

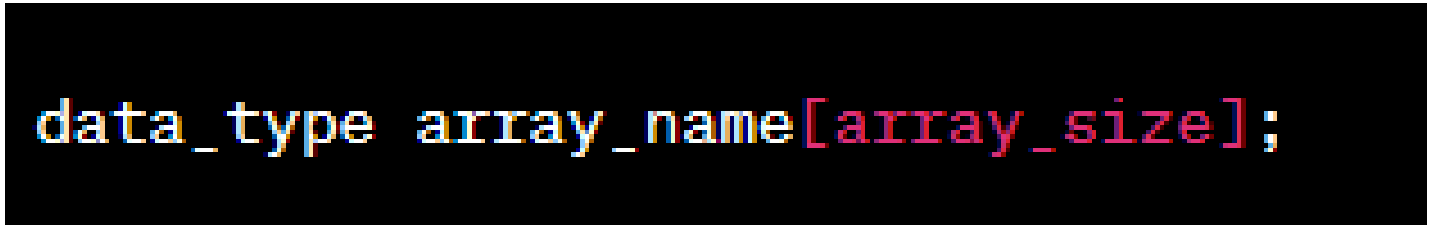
Basic Array Operations

Following are the basic operations supported by an array.

- **Traverse** – print/visit all the array elements one by one.
- **Insertion** – Adds an element at the given index.
- **Deletion** – Deletes an element at the given index.
- **Search –** Searches an element using the given index or by the value.
- **Update –** Updates an element at the given index.

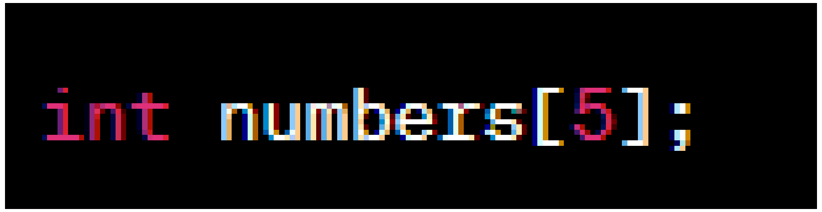
Arrays in C

- Declaring an Array : specify the data type of the elements it will hold and the size of the array.



```
data_type array_name[array_size];
```

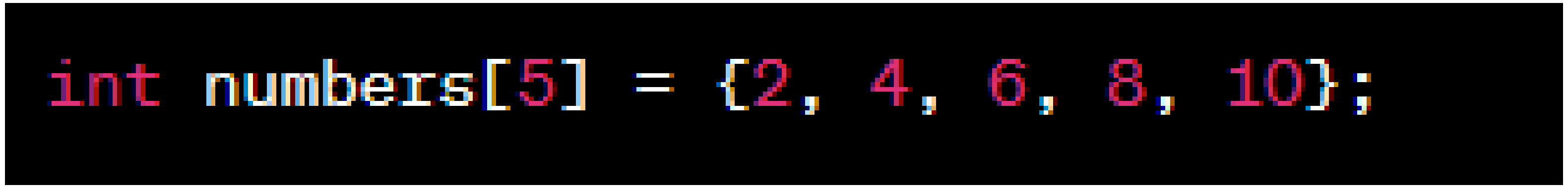
- Eg: Declare an array of integers with a size of 5



```
int numbers[5];
```

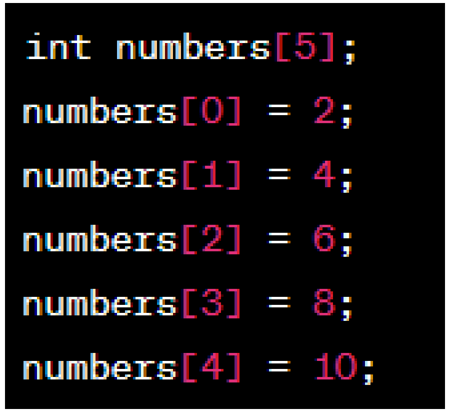
Arrays in C (Cont’d)

- Initializing an array
  - You can initialize an array at the time of declaration by providing a list of values enclosed in curly braces.
  - The number of values provided should be equal to or less than the size of the array.



```c
int numbers[5] = {2, 4, 6, 8, 10};
```

- Alternatively, you can initialize individual elements of the array separately.

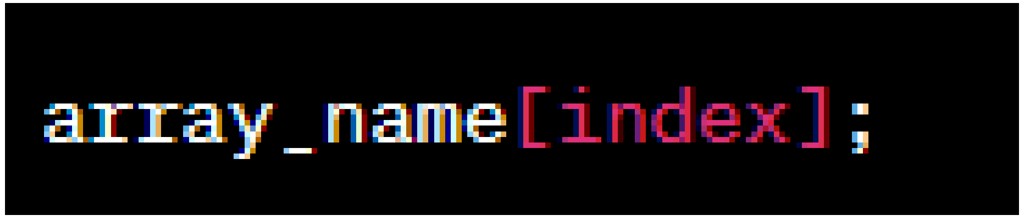


```c
int numbers[5];
numbers[0] = 2;
numbers[1] = 4;
numbers[2] = 6;
numbers[3] = 8;
numbers[4] = 10;
```

Arrays in C (Cont’d)

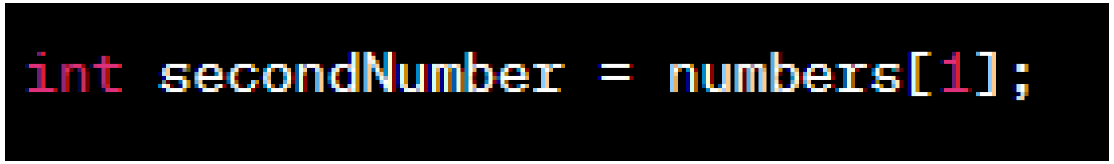
- Accessing Array Elements:

Array elements are accessed using their index, which starts from 0 for the first element and goes up to the size of the array minus 1 for the last element.



```
array_name[index];
```

To access the second element of the numbers array

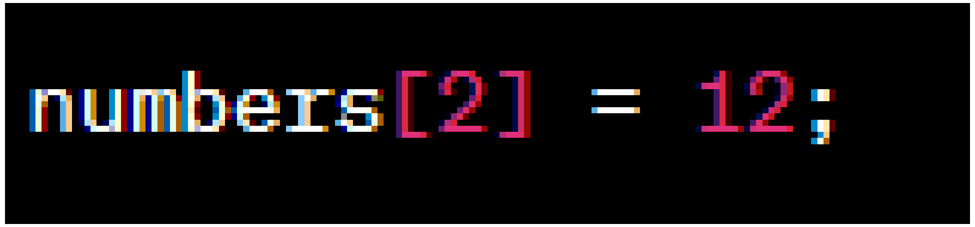


```
int secondNumber = numbers[1];
```

Arrays in C (Cont'd)

- Modifying Array Elements:

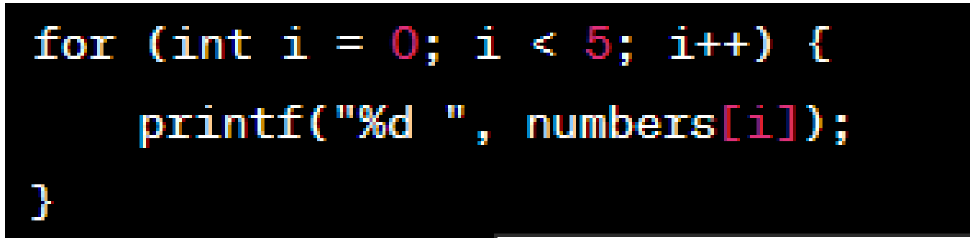
You can modify array elements using the same syntax as accessing array elements.



```
numbers[2] = 12;
```

- Looping through an array

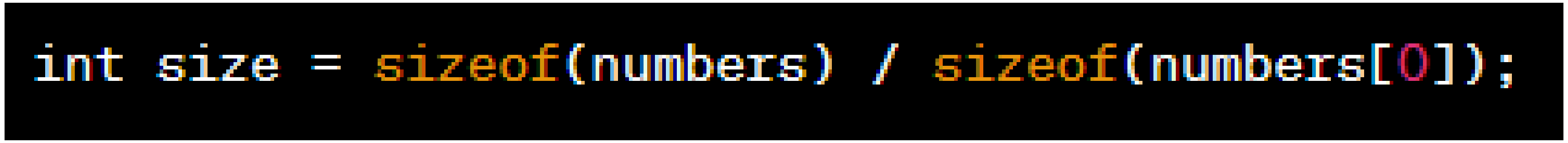
can use loops, such as the for loop, to iterate through an array.



```
for (int i = 0; i < 5; i++) {
    printf("%d ", numbers[i]);
}
```

Arrays in C (Cont’d)

- Array size - To determine the size of an array, you can use the sizeof operator.



```
int size = sizeof(numbers) / sizeof(numbers[0]);
```

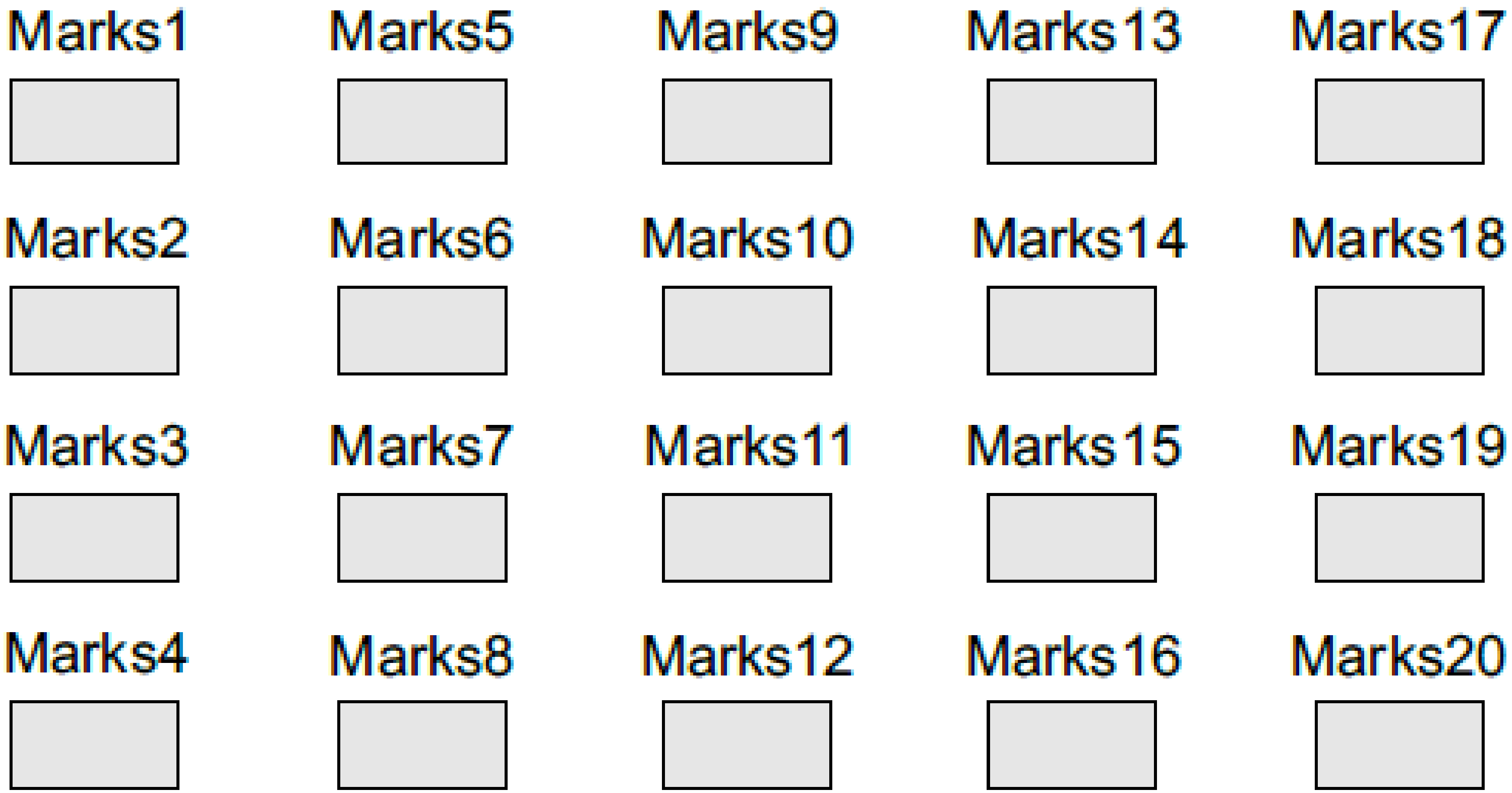
- The total size of the array is divided by the size of a single element to get the number of elements.

Question 1

- Consider a situation in which we have 20 students in a class, and we have been asked to write a program that reads and prints the marks of all the 20 students.

Question 1 – Approach 1

Declaring 20 variables to store each student’s mark



Marks1	Marks5	Marks9	Marks13	Marks17
Marks2	Marks6	Marks10	Marks14	Marks18
Marks3	Marks7	Marks11	Marks15	Marks19
Marks4	Marks8	Marks12	Marks16	Marks20

Question 1 – Approach 1

- If it is just a matter of 20 variables, then it might be acceptable for the user to follow this approach.
- But would it be possible to follow this approach if we have to read and print the marks of students,
  - In the entire course (say 100 students)
  - In the entire college (say 500 students)
  - In the entire university (say 10,000 students)

Then ?

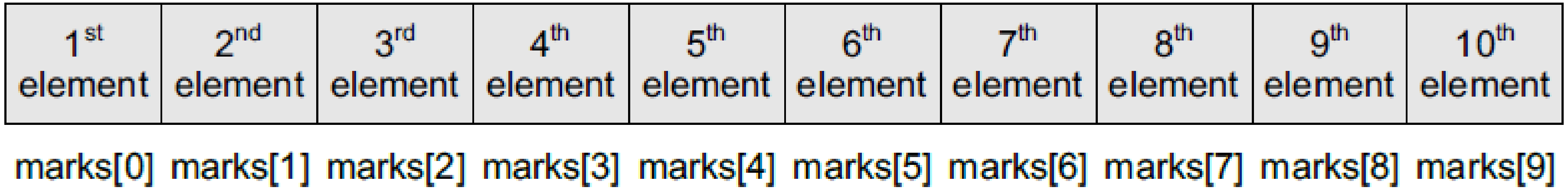
Question 1 – Approach 2

- If it is just a matter of 20 variables, then it might be acceptable for the user to follow this approach.
- But would it be possible to follow this approach if we have to read and print the marks of students,
  - in the entire course (say 100 students)
  - in the entire college (say 500 students)
  - in the entire university (say 10,000 students)
- The answer is no, definitely not! To process a large amount of data,
- we need a data structure like *array*.

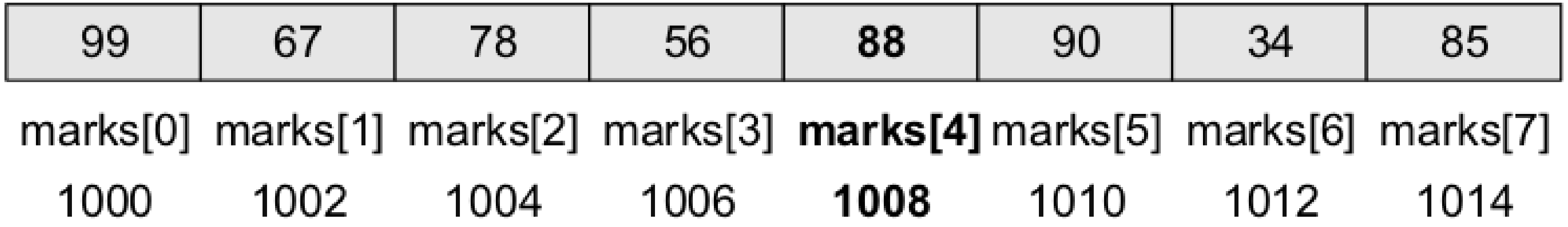
Question 1 – Approach 2 -> Why Array ?

- An array is a collection of similar data elements.
- These data elements have the same data type.
- The elements of the array are stored in consecutive memory locations and are referenced by an index.
- An array must be declared before being used.
  - ***Data type***—the kind of values it can store, for example, int, char, float, double.
  - ***Name***—to identify the array.
  - ***Size***—the maximum number of values that the array can hold.

Question 1 – Approach 2 - Memory representation



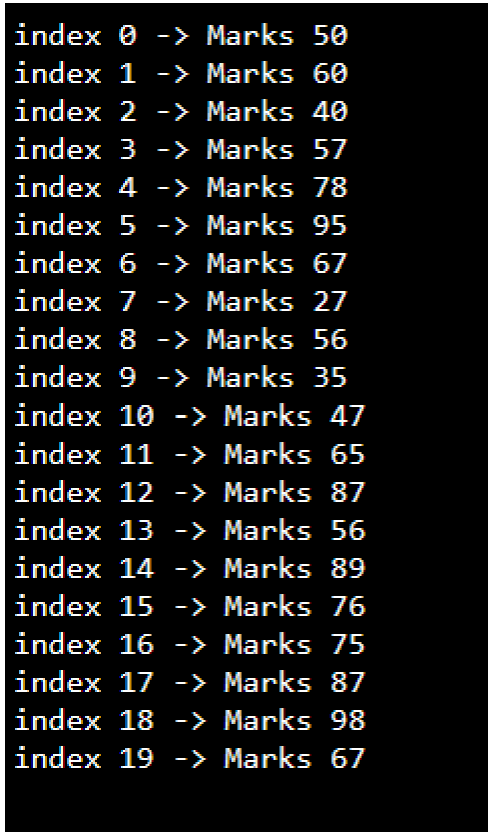
1st element	2nd element	3rd element	4th element	5th element	6th element	7th element	8th element	9th element	10th element
marks[0]	marks[1]	marks[2]	marks[3]	marks[4]	marks[5]	marks[6]	marks[7]	marks[8]	marks[9]



99	67	78	56	**88**	90	34	85
marks[0]	marks[1]	marks[2]	marks[3]	**marks[4]**	marks[5]	marks[6]	marks[7]
1000	1002	1004	1006	**1008**	1010	1012	1014

Approach 2 - Printing array

```
#include <stdio.h>

int main() {

  int student_marks[20] =
{50,60,40,57,78,95,67,27,56,35,47,65,87,56,89,76,75,87,98,67};

  //print the array

  int i;
  for(i = 0; i < 20; i++)
  {
       printf("index %d -> Marks %d \n",i, student_marks[i]);
  }

  return 0;
}
```



```
index 0 -> Marks 50
index 1 -> Marks 60
index 2 -> Marks 40
index 3 -> Marks 57
index 4 -> Marks 78
index 5 -> Marks 95
index 6 -> Marks 67
index 7 -> Marks 27
index 8 -> Marks 56
index 9 -> Marks 35
index 10 -> Marks 47
index 11 -> Marks 65
index 12 -> Marks 87
index 13 -> Marks 56
index 14 -> Marks 89
index 15 -> Marks 76
index 16 -> Marks 75
index 17 -> Marks 87
index 18 -> Marks 98
index 19 -> Marks 67
```

Approach 2 – Reading elements and Printing array

```
#include <stdio.h>

int main() {

int student_marks[20];
  printf("Enter 20 integers : ");
  // taking input and storing it in an integer array
  for(int i = 0; i < 20; ++i) {
     scanf("%d", &student_marks[i]);
  }
  printf("Displaying integers: ");
  // printing elements of an array
  for(int i = 0; i < 20; ++i) {
     printf("%d\n", student_marks[i]);
  }   return 0;
}
```

Operations on Arrays

- Traversing an array
- Inserting an element in an array
- Searching an element in an array
- Deleting an element from an array
- Merging two arrays
- Sorting an array in ascending or descending order

Homework 1.1

A. Write a program to find the mean of *n* numbers using arrays.

B. Write a program to print the position of the smallest number of *n* numbers using arrays.

C. Write a program to find the second largest of *n* numbers using an array.

D. Write a program to enter *n* number of digits. Form a number using these digits.

Inserting an element

- Let **LA** be a Linear Array (unordered) with **N** elements and **K** is a positive integer such that **K<=N**.
- Following is the algorithm where ITEM is inserted into the Kth position of array LA .

```
Setp1 : Start
Step 2: Set J=N
Step 3: Set N=N+1
Step 4 : Repeat Step 5 & 6 while J>=k
Step 5 : LA[J+1]=LA[j]
Step 6: J=J-1
Step 7 :LA[K]=ITEM
Step 8 Stop
```

Deleting an element

- Deletion refers to removing an existing element from the array and re-organizing all elements of an array.
- Consider **LA** is a linear array with **N** elements and **K** is a positive integer such that **K<=N**.
- Following is the algorithm to delete an element available at the Kth position of LA.

```
Step1 :Start
Step 2 : Set J=k
Step 3: Repeat Step 4 & 5
while J < N
Step 4: Set LA[J]=LA[J+1]
Step 5: Set J=J+1
Step 6: Set N=N-1
Step 7:  Stop
```

Searching an element

- Consider **LA** is a linear array with **N** elements and **K** is a positive integer such that **K<=N**.
- Following is the algorithm to find an element with a value of ITEM using sequential search.

```
Step 1:Start
Step 2:Set J=0
Step 3:Repeat Step 4 & 5 while J<N
Step 4: If LA[J] == ITEM THEN GO TO Step 6
Step 5:Set J=J+1
Step 6:Print J, Item
Step 7:Stop
```

Inserting an element

- **Inserting at the end**

If an element has to be inserted at the end of an existing array, then the task of insertion is quite simple.

```
Step 1 : Set upper_bound = upper_bound +1
Step 2 : Set A[upper_bound] = VAL
Step 3 : EXIT
```

Inserting an element

- **Inserting at the middle**

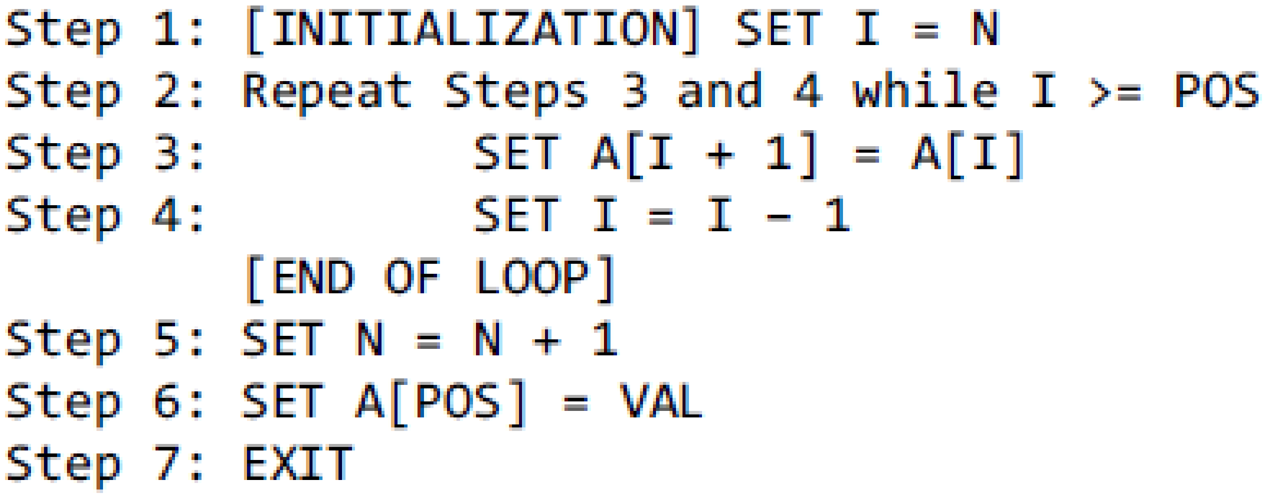
INSERT (A, N, POS, VAL). The arguments are:

(a) A, the array in which the element has to be inserted

(b) N, the number of elements in the array

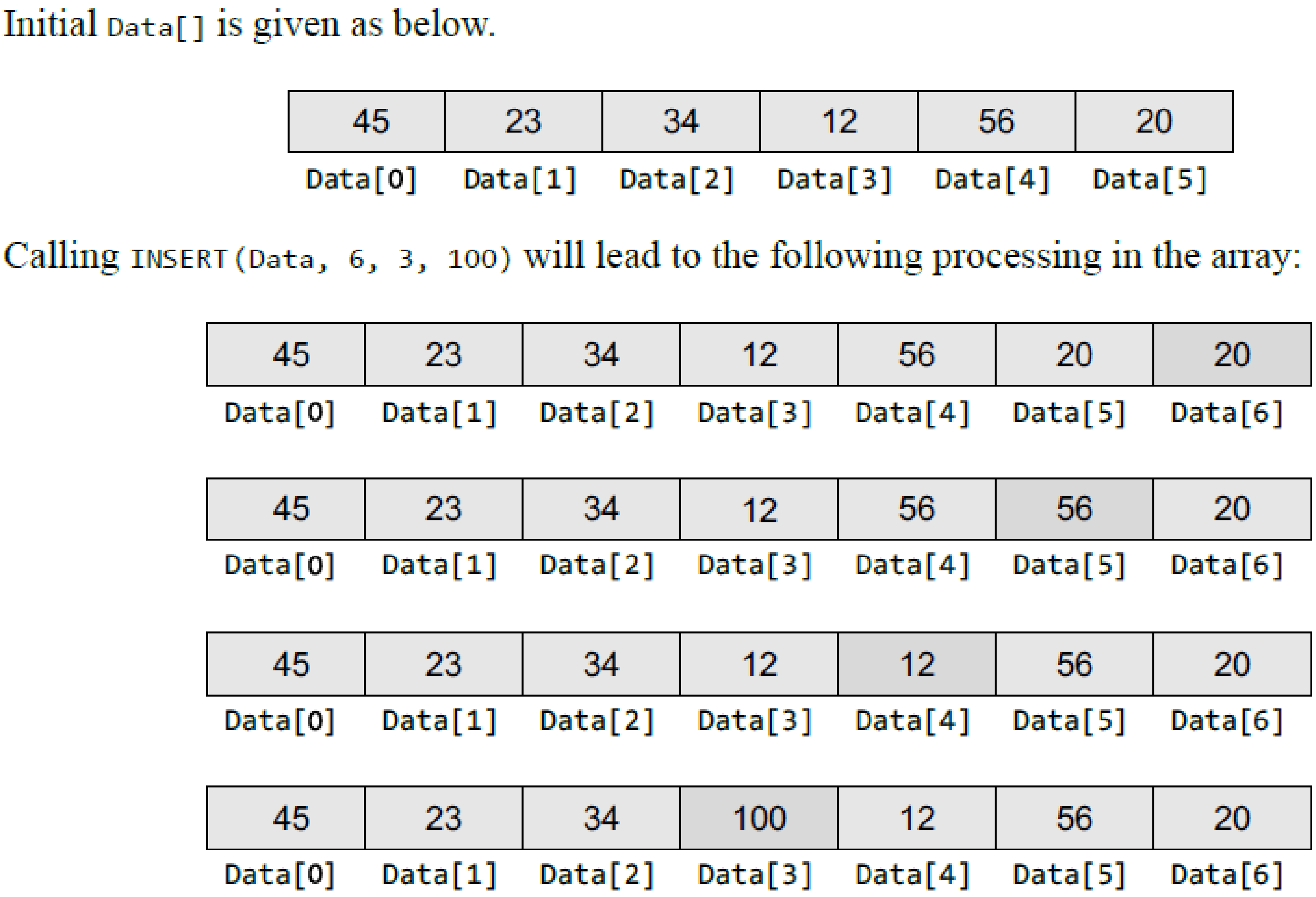
(c) POS, the position at which the element has to be inserted

(d) VAL, the value that has to be inserted



```
Step 1: [INITIALIZATION] SET I = N
Step 2: Repeat Steps 3 and 4 while I >= POS
Step 3:           SET A[I + 1] = A[I]
Step 4:           SET I = I - 1
        [END OF LOOP]
Step 5: SET N = N + 1
Step 6: SET A[POS] = VAL
Step 7: EXIT
```

Illustration – Inserting an element to the middle



Initial `Data[]` is given as below.

45	23	34	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

Calling `INSERT(Data, 6, 3, 100)` will lead to the following processing in the array:

45	23	34	12	56	20	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

45	23	34	12	56	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

45	23	34	12	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

45	23	34	100	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

Homework 1.2

A. Write a program to insert a number at a given location in an array

B. Write a program to insert a number in an array that is already sorted in ascending order.

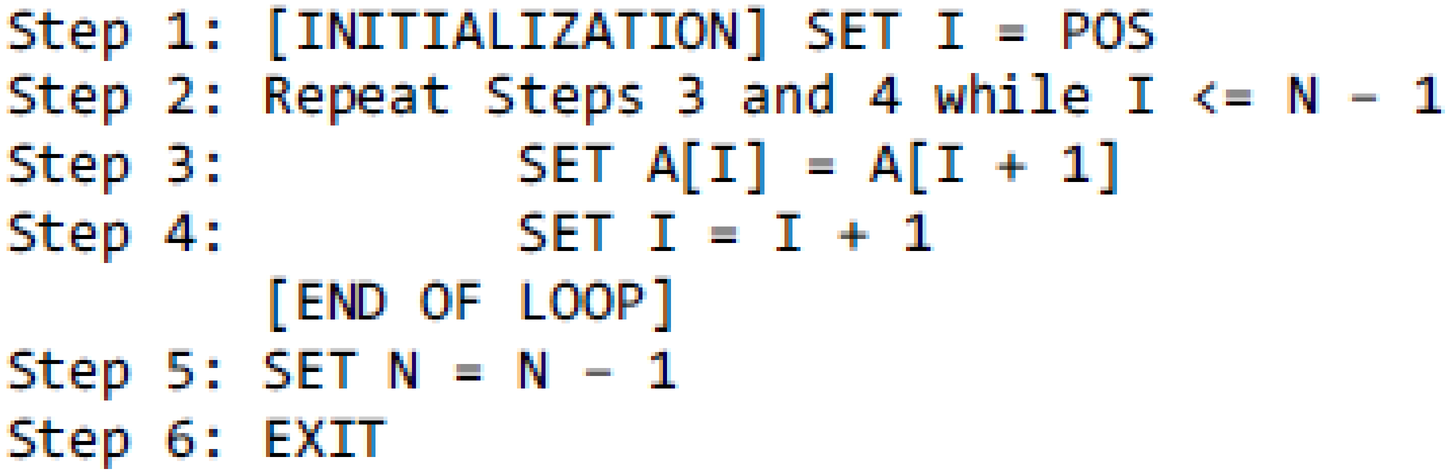
Deleting an element from middle

The algorithm DELETE will be declared as DELETE(A, N,POS). The arguments are:

(a) A, the array from which the element must be deleted

(b) N, the number of elements in the array

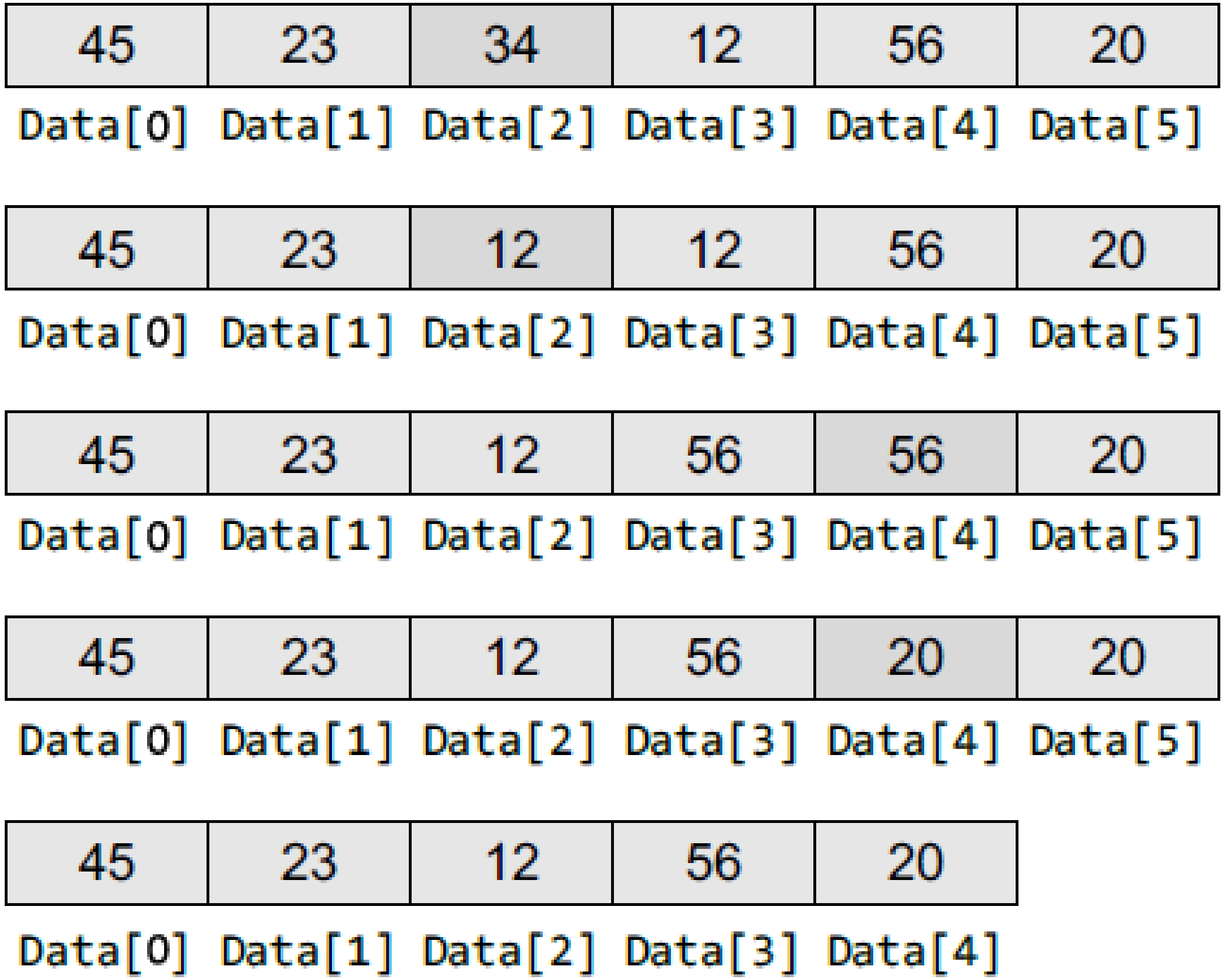
(c) POS, the position from which the element has to be deleted



```
Step 1: [INITIALIZATION] SET I = POS
Step 2: Repeat Steps 3 and 4 while I <= N - 1
Step 3:         SET A[I] = A[I + 1]
Step 4:         SET I = I + 1
        [END OF LOOP]
Step 5: SET N = N - 1
Step 6: EXIT
```

Illustration – Deleting an element from the middle

- DELETE (Data, 6, 2)



45	23	34	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

45	23	12	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

45	23	12	56	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

45	23	12	56	20	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

45	23	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]

Homework 1.3

A. Write a program to delete a number from a given location in an array.

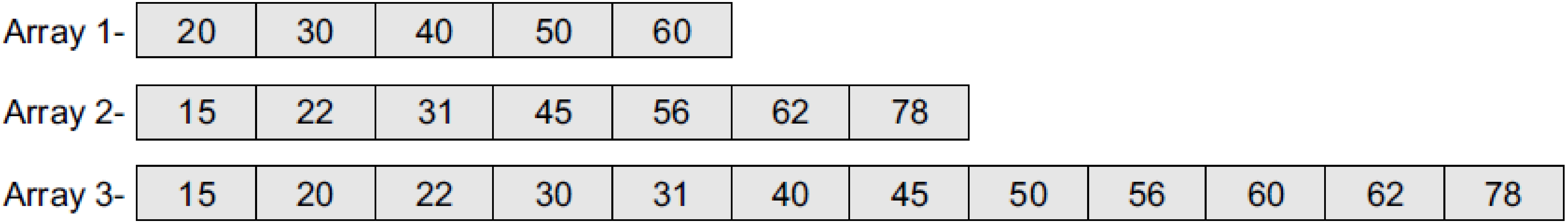
B. Write a program to delete a number from an array that is already sorted in ascending order.

Merging Two Arrays

- Merged array = array 1 + array 2
- Merged array = copying content of the first array followed by second array.

Array 1-	90	56	89	77	69							
Array 2-	45	88	76	99	12	58	81					
Array 3-	90	56	89	77	69	45	88	76	99	12	58	81

Merging and sorting two arrays



Array 1- | 20 | 30 | 40 | 50 | 60 |

Array 2- | 15 | 22 | 31 | 45 | 56 | 62 | 78 |

Array 3- | 15 | 20 | 22 | 30 | 31 | 40 | 45 | 50 | 56 | 60 | 62 | 78 |

1. Merge two arrays
2. Sort the third array

Homework 1.4

A. Write a program to merge two arrays

B. Extend the above program merge any number of arrays

Thank you