Lecture 2 - Array & Structure

CPE112 - Programming with Data Structures

22 January 2025

Dr. Piyanit Ua-areemitr
Dr. Taweechai Nuntawisuttiwong

Department of Computer Engineering
KMUTT

Outlines

- All about arrays: declaration, accessing elements, storing values, operations, pointers, and 2D arrays
- String
- Structure & union

Array

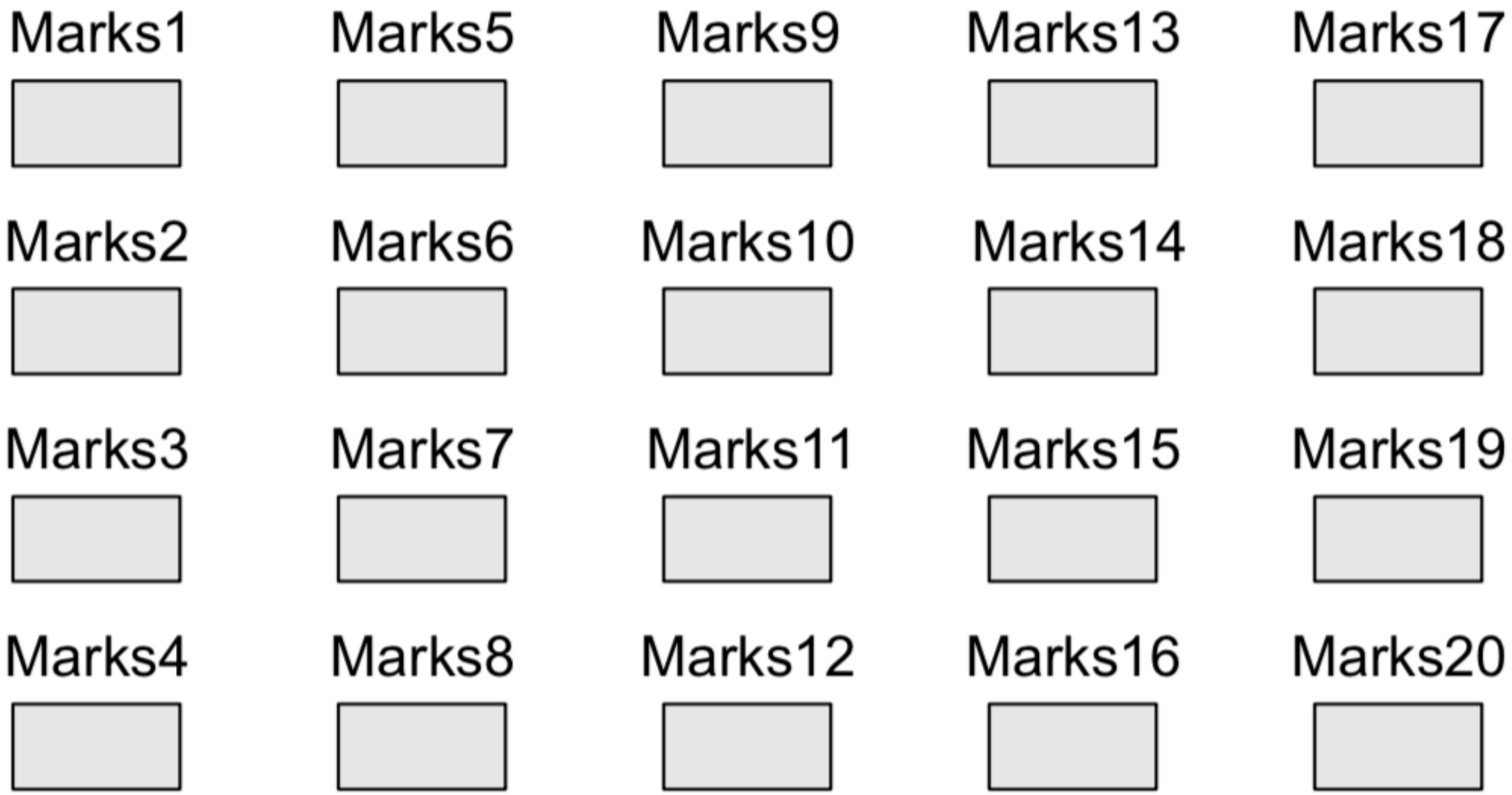


Figure 3.1 Twenty variables for 20 students

Array

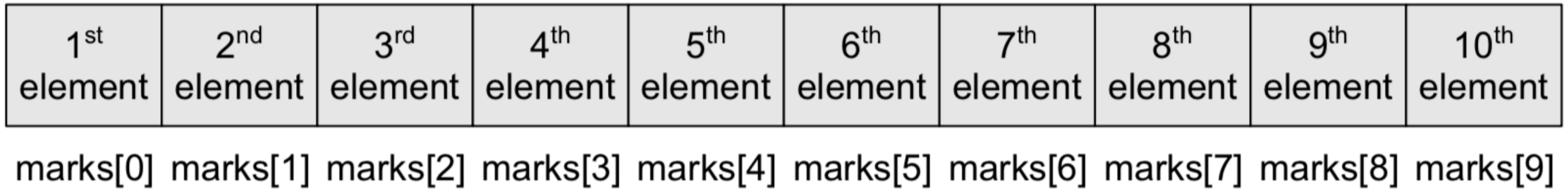
- An array is a collection of similar data elements.
- These data elements have the same data type.
- The elements of the array are stored in consecutive memory locations and are referenced by an index (also known as the subscript).
- The subscript is an ordinal number which is used to identify an element of the array.

Array

Declaration

```
type name[size];
```

```
int marks[10];
```

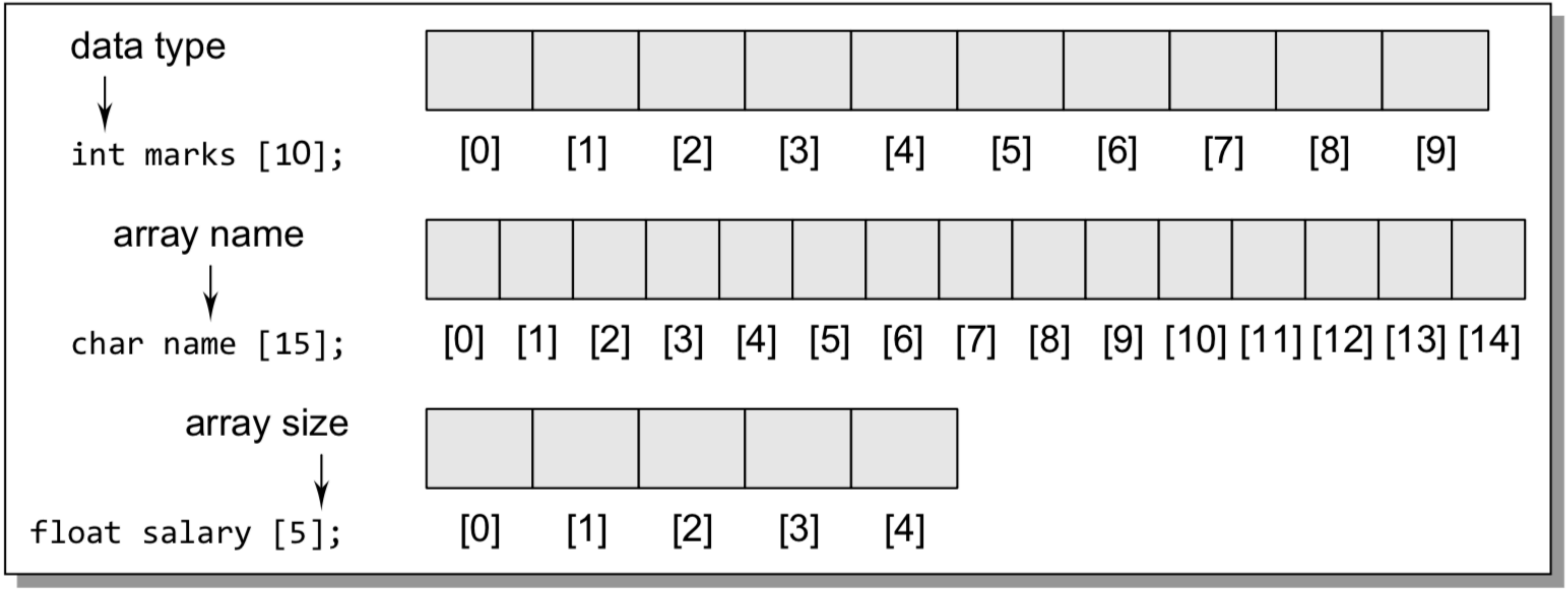


1st element	2nd element	3rd element	4th element	5th element	6th element	7th element	8th element	9th element	10th element
marks[0]	marks[1]	marks[2]	marks[3]	marks[4]	marks[5]	marks[6]	marks[7]	marks[8]	marks[9]

Figure 3.2 Memory representation of an array of 10 elements

Array

Declaration



```
int marks [10];
char name [15];
float salary [5];
```

Figure 3.3 Declaring arrays of different data types and sizes

Array

Declaration

A

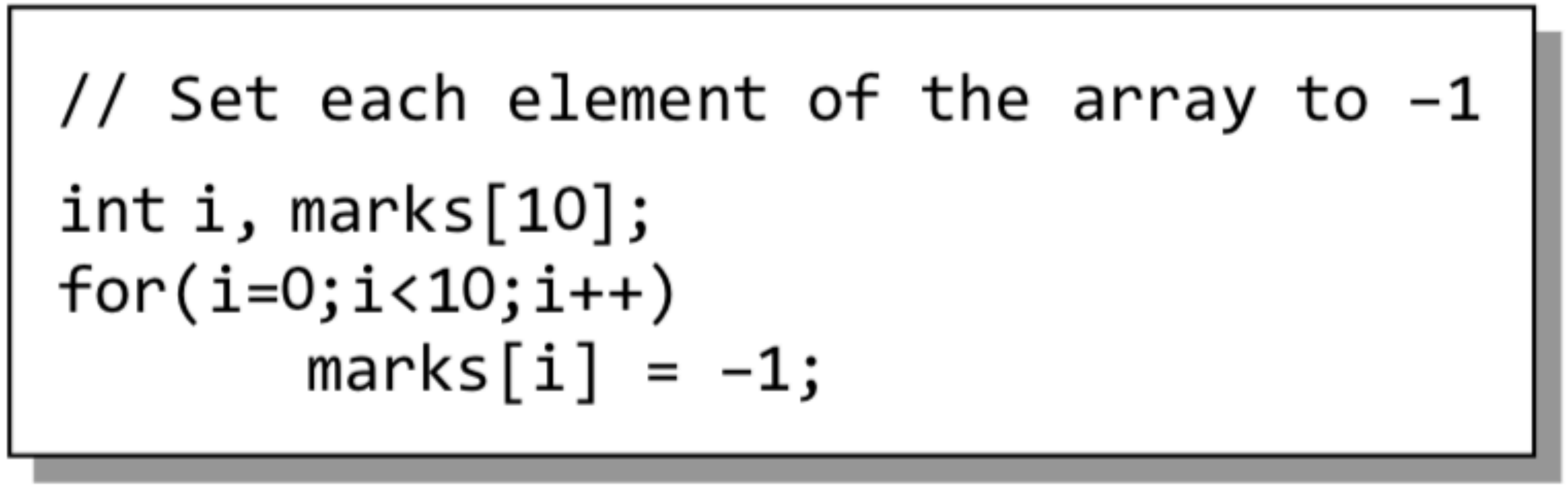
```
int marks[10];
```

B

```
int ARRAYSIZE = 10;
int* marks = calloc(ARRAYSIZE,sizeof(int));
```

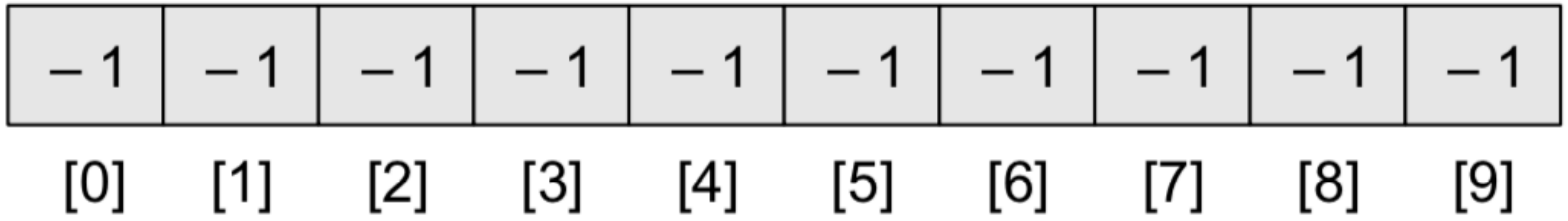
Array

Accessing the elements



```
// Set each element of the array to –1
int i, marks[10];
for(i=0;i<10;i++)
        marks[i] = –1;
```

Figure 3.4 Code to initialize each element of the array to –1

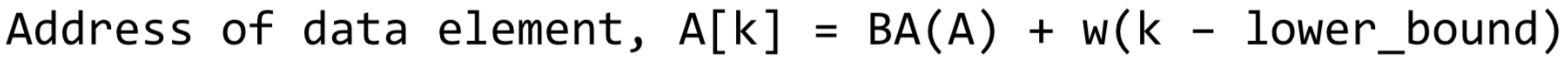


– 1	– 1	– 1	– 1	– 1	– 1	– 1	– 1	– 1	– 1
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

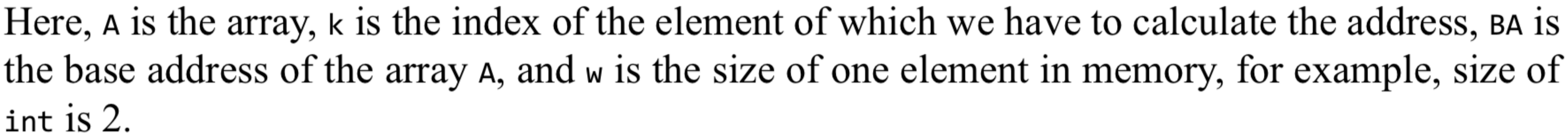
Figure 3.5 Array `marks` after executing the code given in Fig. 3.4

Array

Calculating the address of array elements



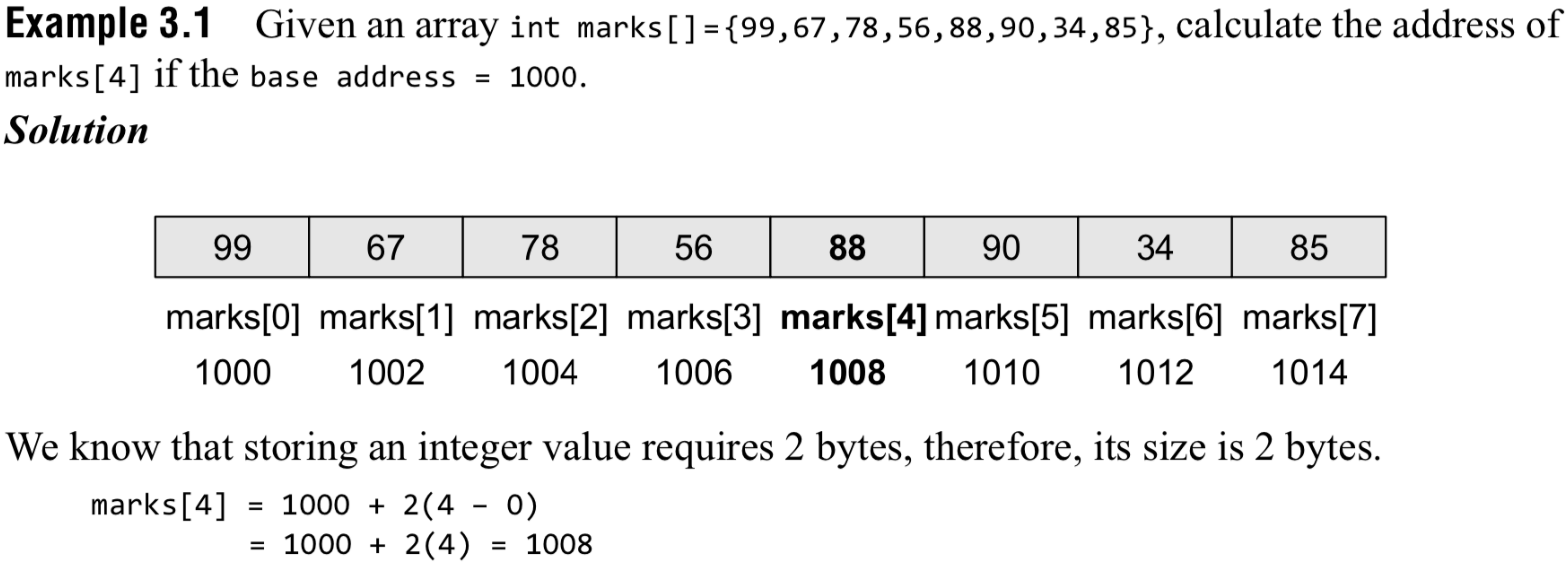
```
Address of data element, A[k] = BA(A) + w(k – lower_bound)
```



Here, `A` is the array, `k` is the index of the element of which we have to calculate the address, `BA` is the base address of the array `A`, and `w` is the size of one element in memory, for example, size of `int` is 2.

Array

Calculating the address of array elements



Example 3.1 Given an array `int marks[]={99,67,78,56,88,90,34,85}`, calculate the address of `marks[4]` if the `base address = 1000`.

Solution

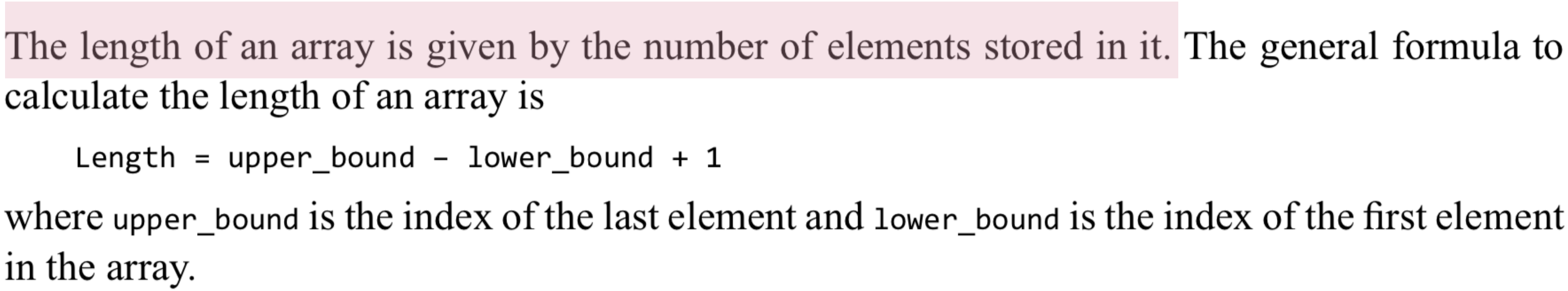
99	67	78	56	**88**	90	34	85
marks[0]	marks[1]	marks[2]	marks[3]	**marks[4]**	marks[5]	marks[6]	marks[7]
1000	1002	1004	1006	**1008**	1010	1012	1014

We know that storing an integer value requires 2 bytes, therefore, its size is 2 bytes.

```
    marks[4] = 1000 + 2(4 – 0)
             = 1000 + 2(4) = 1008
```

Array

Calculating the length of array elements



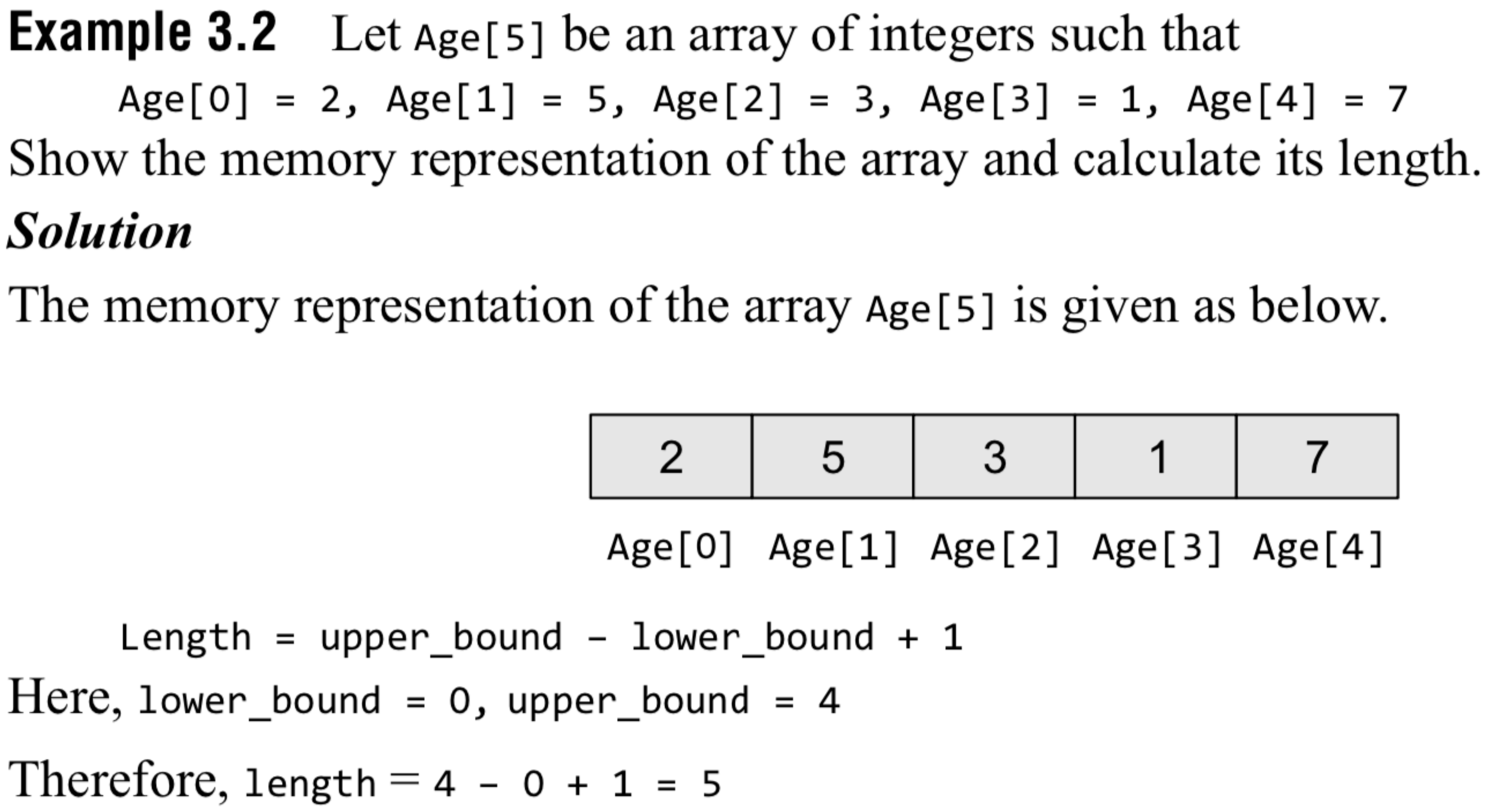
The length of an array is given by the number of elements stored in it. The general formula to calculate the length of an array is

```
Length = upper_bound – lower_bound + 1
```

where `upper_bound` is the index of the last element and `lower_bound` is the index of the first element in the array.

Array

Calculating the length of array elements



Example 3.2 Let `Age[5]` be an array of integers such that

```
Age[0] = 2, Age[1] = 5, Age[2] = 3, Age[3] = 1, Age[4] = 7
```

Show the memory representation of the array and calculate its length.

Solution

The memory representation of the array `Age[5]` is given as below.

2	5	3	1	7
Age[0]	Age[1]	Age[2]	Age[3]	Age[4]

```
Length = upper_bound – lower_bound + 1
```

Here, `lower_bound = 0, upper_bound = 4`

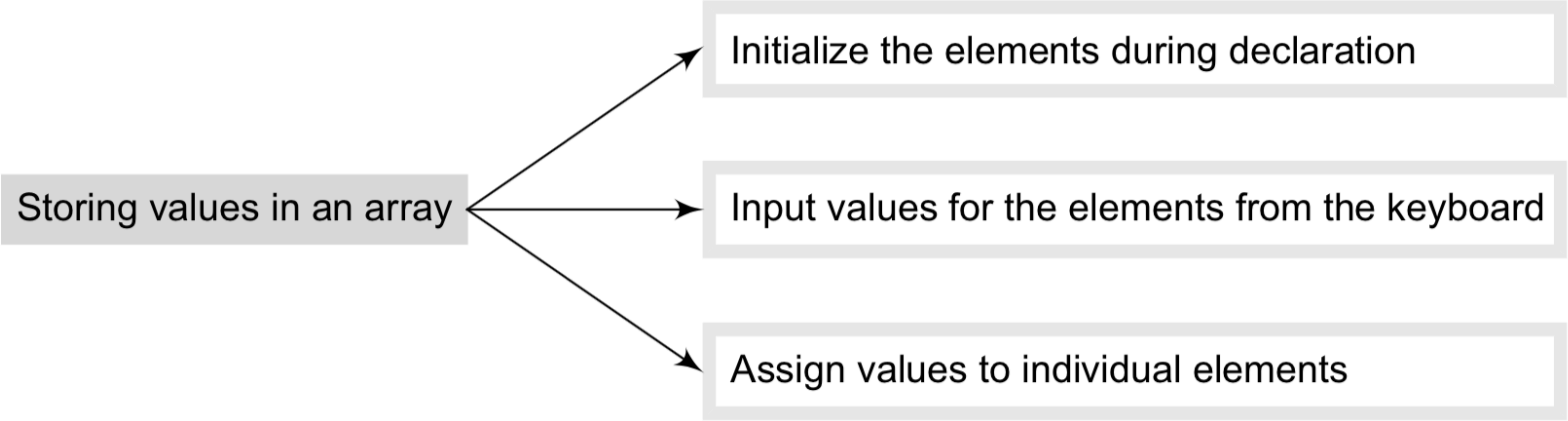
Therefore, `length` = `4 – 0 + 1 = 5`

Array

Storing values in arrays

- When we declare an array, we are just allocating space for its elements; no values are stored in the array.

- There are 3 ways to store values in an array.



Storing values in an array
Initialize the elements during declaration
Input values for the elements from the keyboard
Assign values to individual elements

Array

Initializing array during declaration

- The elements of an array can be initialized at the time of declaration, just as any other variable.
- When an array is initialized, we need to provide a value for every element in the array.

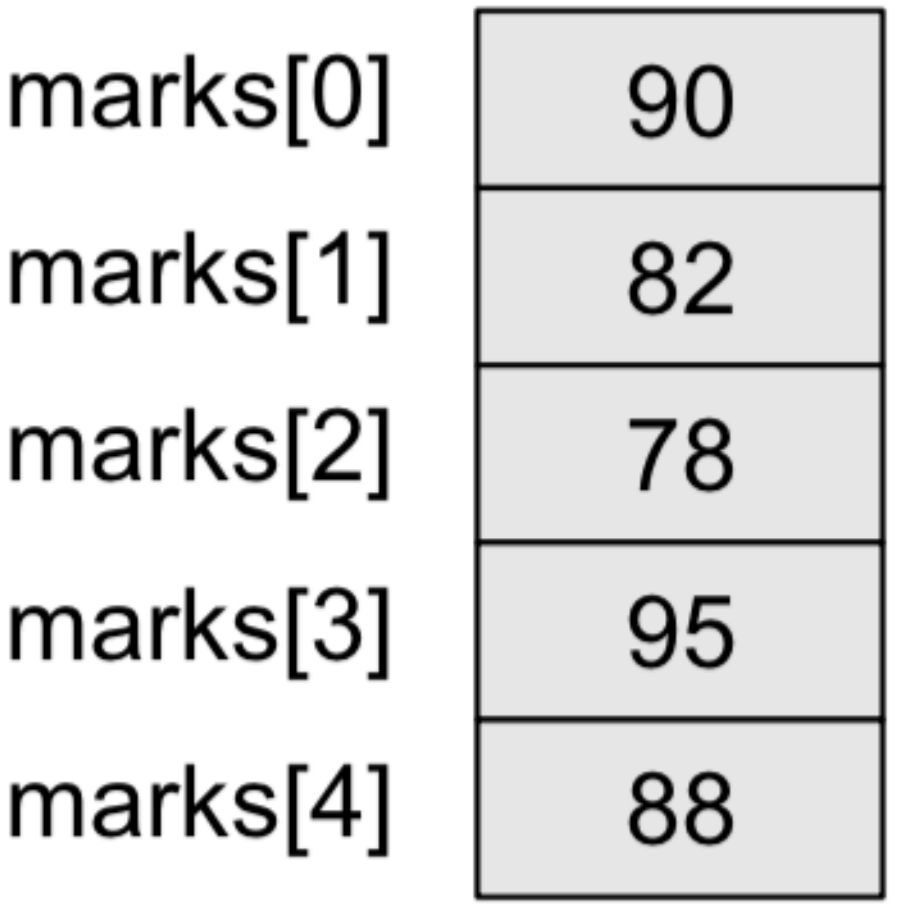
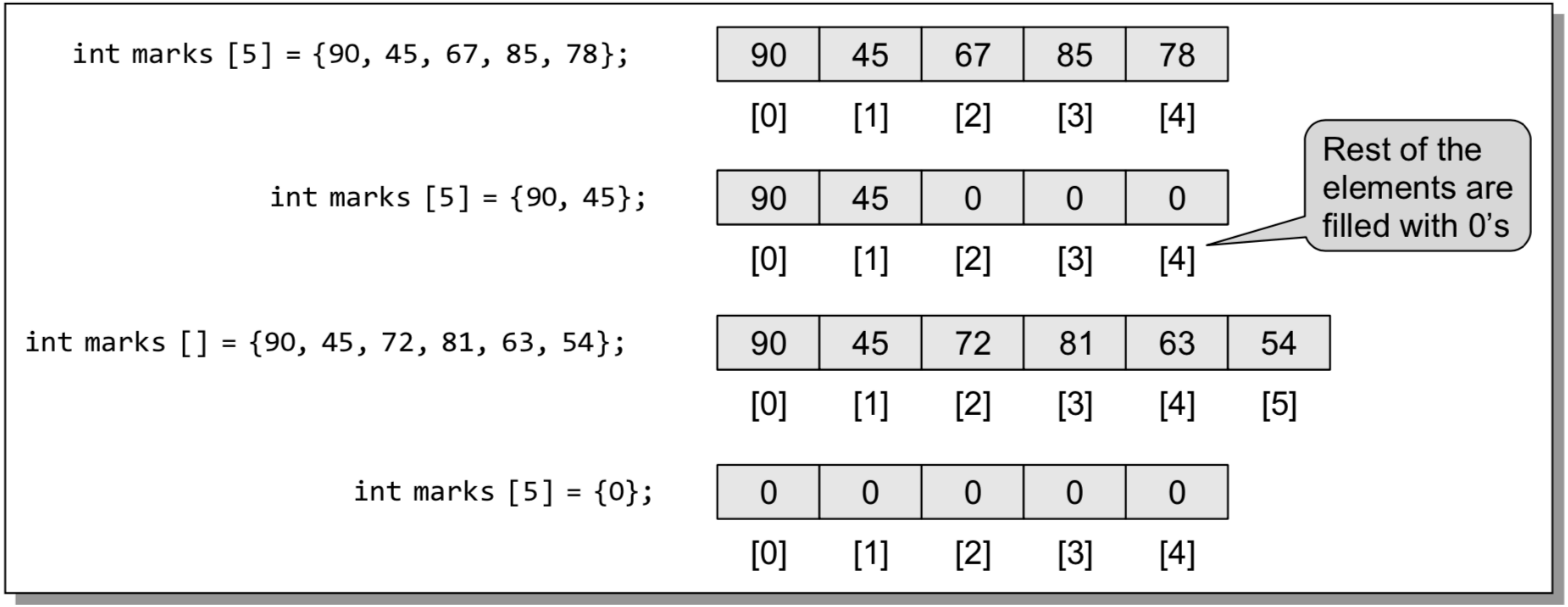


Figure 3.7 Initialization of array `marks[5]`

```
int marks[5]={90, 82, 78, 95, 88};
```

Array

Initializing array during declaration



```
int marks [5] = {90, 45, 67, 85, 78};
```

90	45	67	85	78
[0]	[1]	[2]	[3]	[4]

```
int marks [5] = {90, 45};
```

90	45	0	0	0
[0]	[1]	[2]	[3]	[4]

Rest of the elements are filled with 0's

```
int marks [] = {90, 45, 72, 81, 63, 54};
```

90	45	72	81	63	54
[0]	[1]	[2]	[3]	[4]	[5]

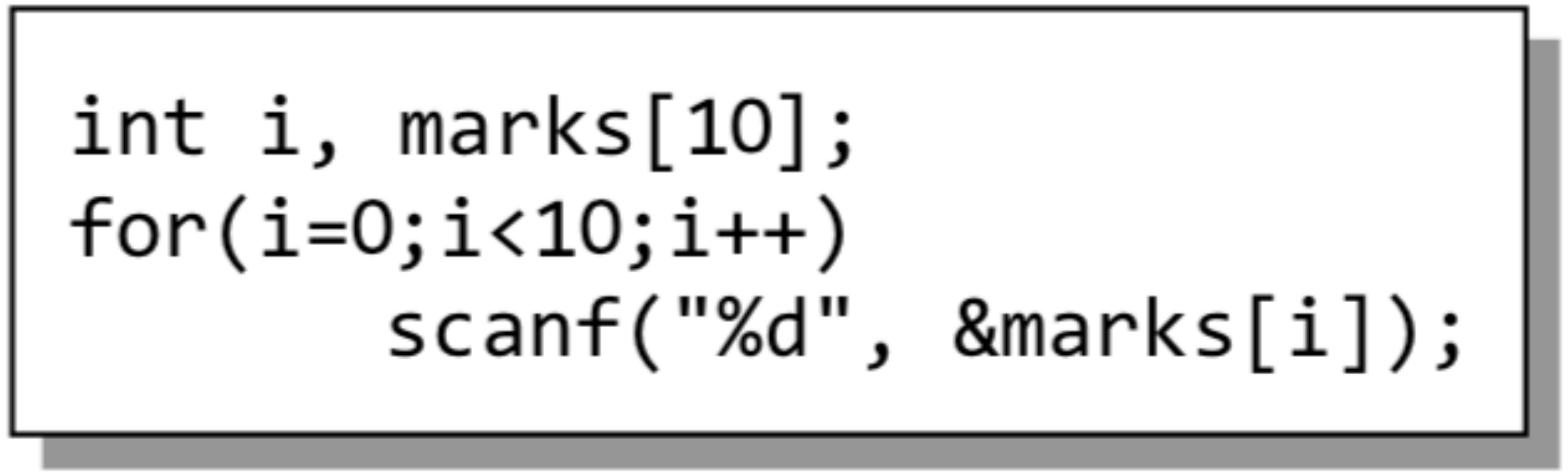
```
int marks [5] = {0};
```

0	0	0	0	0
[0]	[1]	[2]	[3]	[4]

Figure 3.8 Initialization of array elements

Array

Inputting values from keyboard

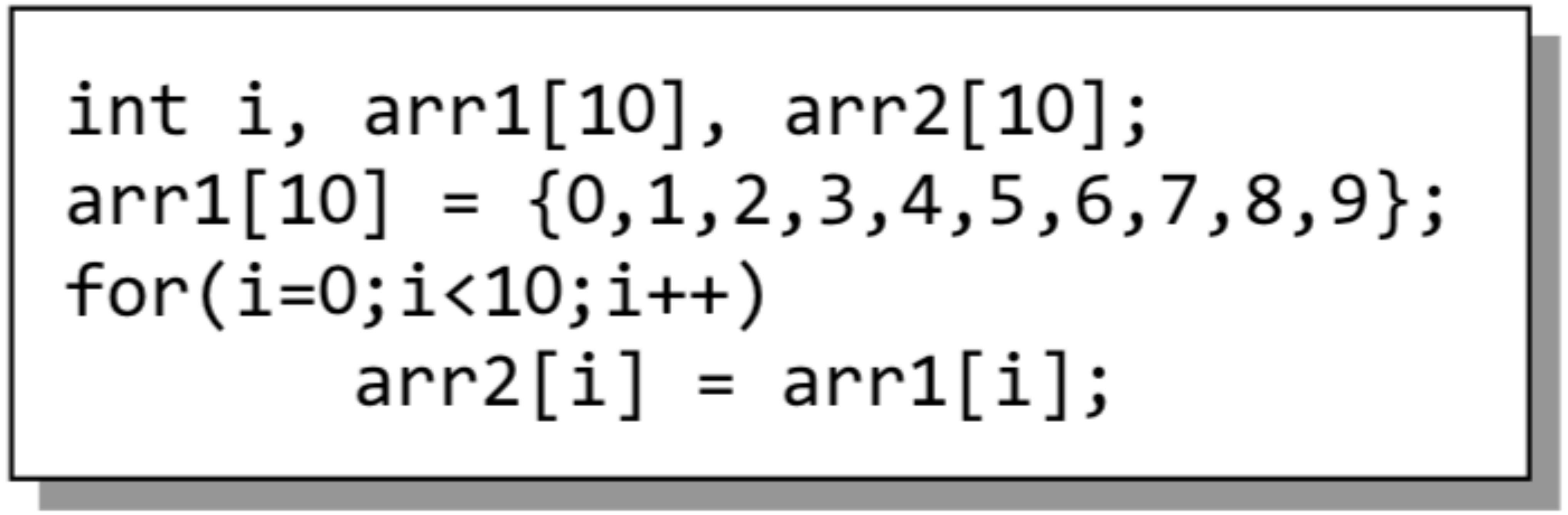


```
int i, marks[10];
for(i=0;i<10;i++)
        scanf("%d", &marks[i]);
```

Figure 3.9 Code for inputting each element of the array

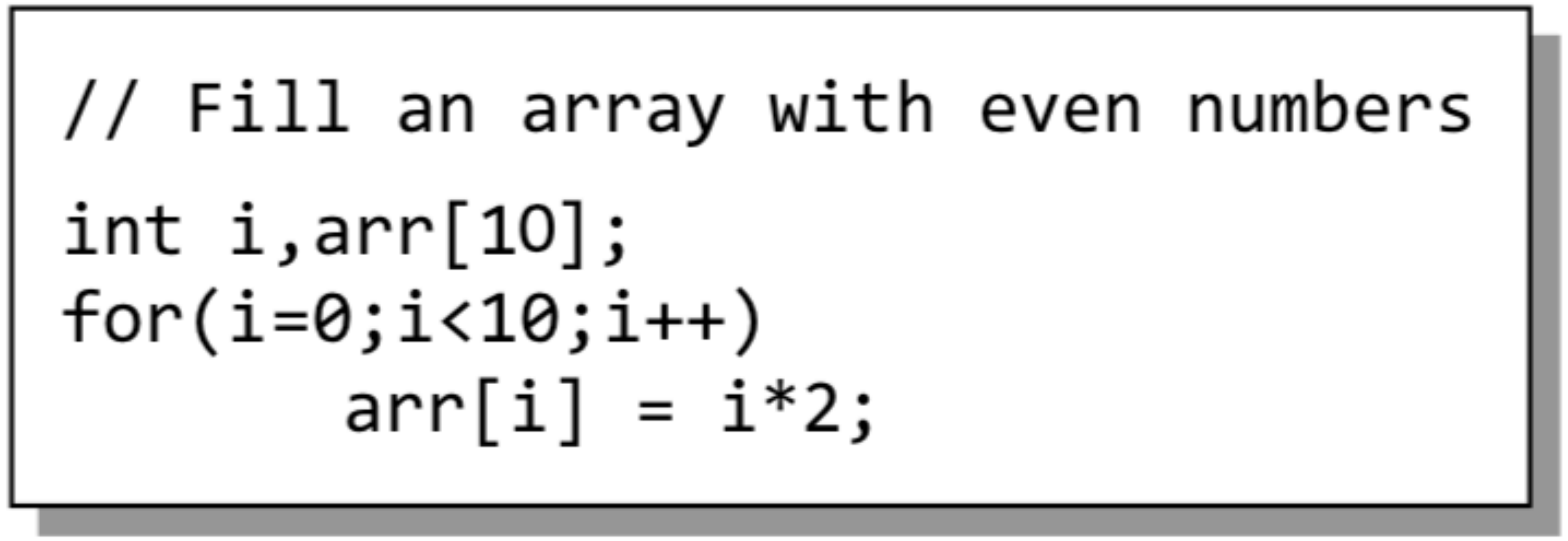
Array

Assigning values to individual elements



```
int i, arr1[1O], arr2[1O];
arr1[1O] = {O,1,2,3,4,5,6,7,8,9};
for(i=O;i<1O;i++)
       arr2[i] = arr1[i];
```

Figure 3.10 Code to copy an array at the individual element level



```
// Fill an array with even numbers

int i,arr[1O];
for(i=0;i<10;i++)
      arr[i] = i*2;
```

Figure 3.11 Code for filling an array with even numbers

Array

Operations on arrays

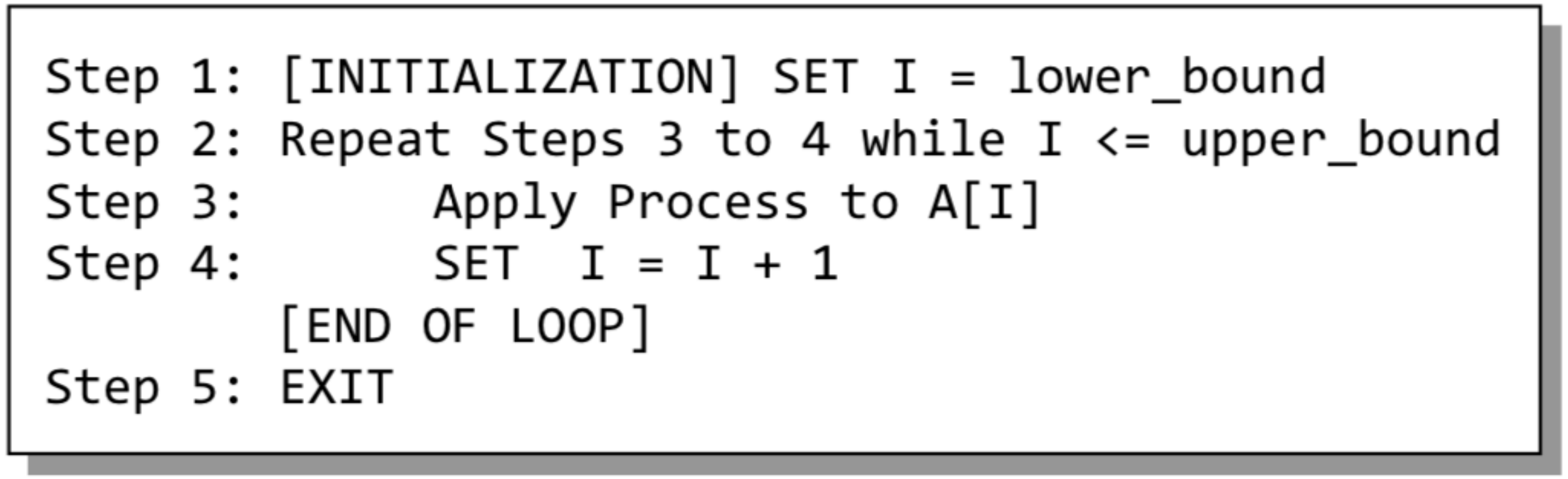
- Traversing an array
- Inserting an element in an array
- Searching an element in an array
- Deleting an element from an array
- Merging two arrays
- Sorting an array in ascending or descending order.

Array

Traversing an array

- Traversing the data elements of an array, *A*, can include printing every element, counting the total number of elements, or performing any process on these elements.

- Since, array is a linear data structure (because all its elements form a sequence), traversing its elements is very simple and straightforward.



```
Step 1: [INITIALIZATION] SET I = lower_bound
Step 2: Repeat Steps 3 to 4 while I <= upper_bound
Step 3:         Apply Process to A[I]
Step 4:         SET  I = I + 1
        [END OF LOOP]
Step 5: EXIT
```

Figure 3.12 Algorithm for array traversal

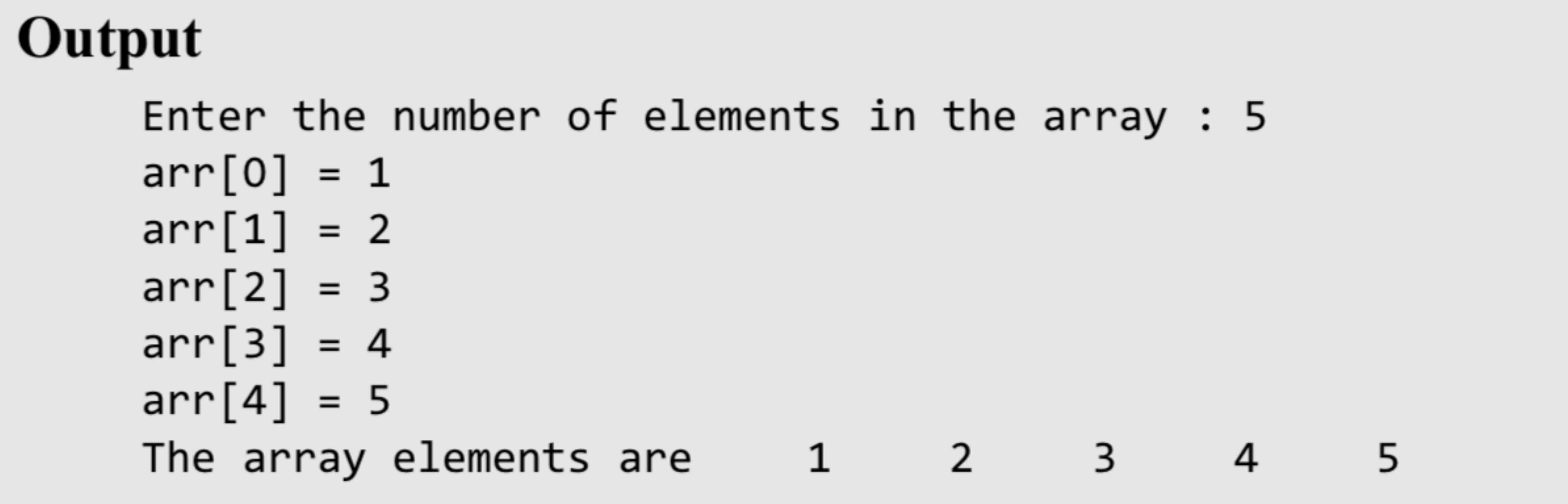
Array

Traversing an array

- Write a program to read and display n numbers using an array.

```
#include <stdio.h>
#include <conio.h>
int main() {
    int i, n, arr[20];
    clrscr();
    printf("\n Enter the number of elements in
the array : ");
    scanf("%d", &n);

    for(i=0;i<n;i++) {
       printf("\n arr[%d] = ", i);
       scanf("%d",&arr[i]);
    }
    printf("\n The array elements are ");
    for(i=0;i<n;i++)
       printf("\t %d", arr[i]);
    return 0;
}
```



Output

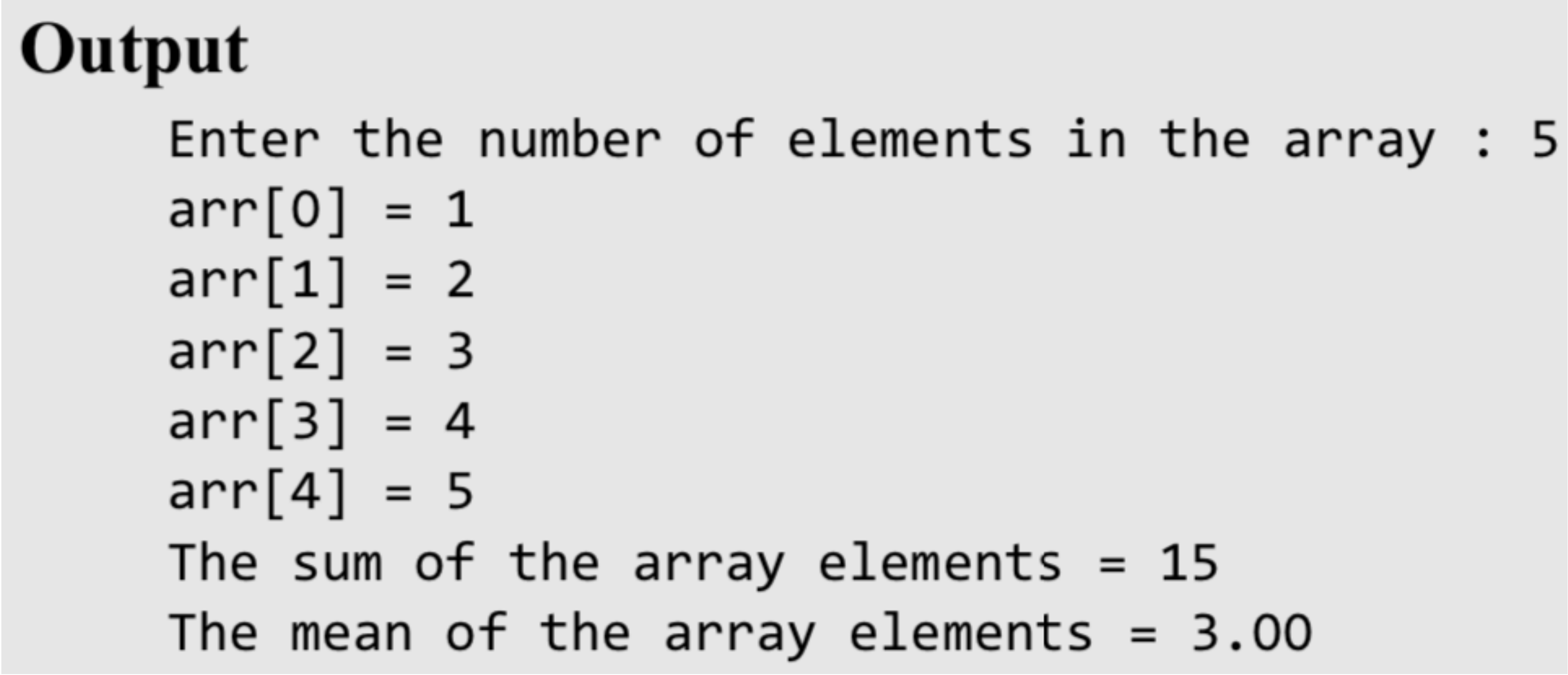
```
Enter the number of elements in the array : 5
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
The array elements are    1    2    3    4    5
```

Array

Traversing an array

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

```
#include <stdio.h>
#include <conio.h>
int main() {
    int i, n, arr[20], sum =0;
    float mean = 0.0;
    clrscr();
    printf("\n Enter the number of elements in the array : ");
    scanf("%d", &n);
    for(i=0;i<n;i++)
    {
         printf("\n arr[%d] = ", i);
         scanf("%d",&arr[i]);
    }
    for(i=0;i<n;i++)
         sum += arr[i];
    mean = (float)sum/n;
    printf("\n The sum of the array elements = %d", sum);
    printf("\n The mean of the array elements = %.2f", mean);
    return 0;
}
```



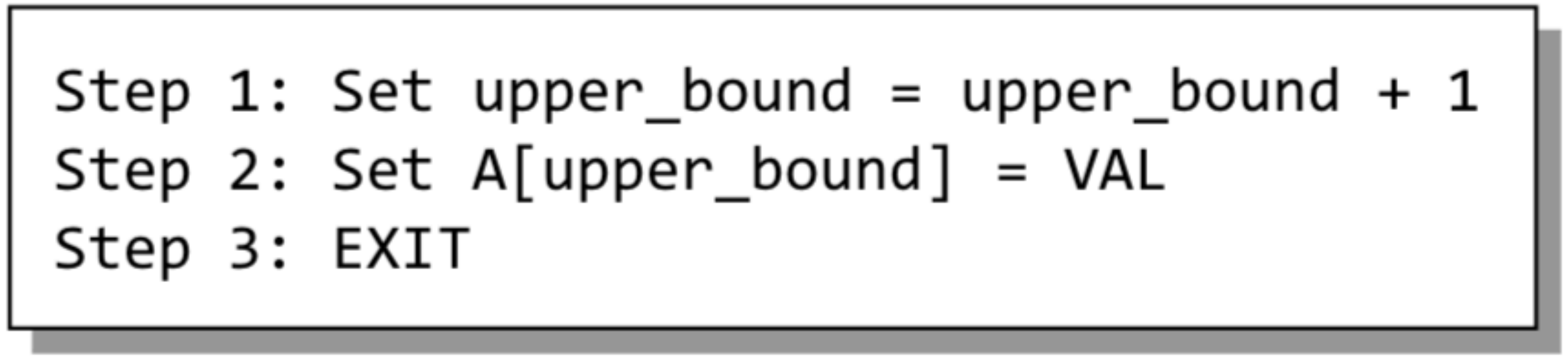
Output

```
    Enter the number of elements in the array : 5
    arr[0] = 1
    arr[1] = 2
    arr[2] = 3
    arr[3] = 4
    arr[4] = 5
    The sum of the array elements = 15
    The mean of the array elements = 3.00
```

Array

Inserting an element in an array

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



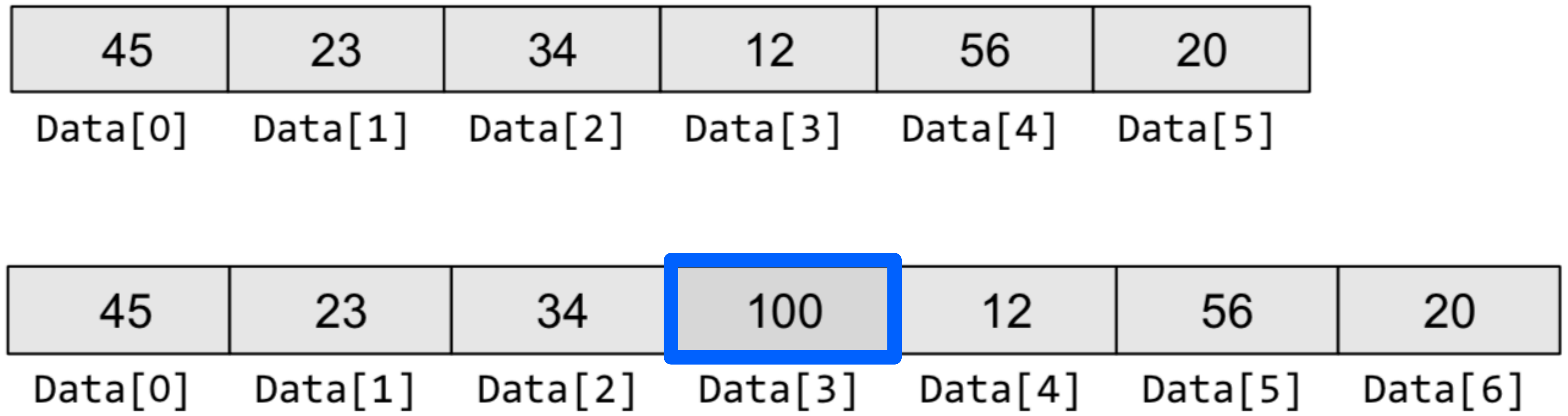
```
Step 1: Set upper_bound = upper_bound + 1
Step 2: Set A[upper_bound] = VAL
Step 3: EXIT
```

Figure 3.13 Algorithm to append a new element to an existing array

Array

Inserting an element in an array

- If an element has to be inserted at the middle of an existing array, then …

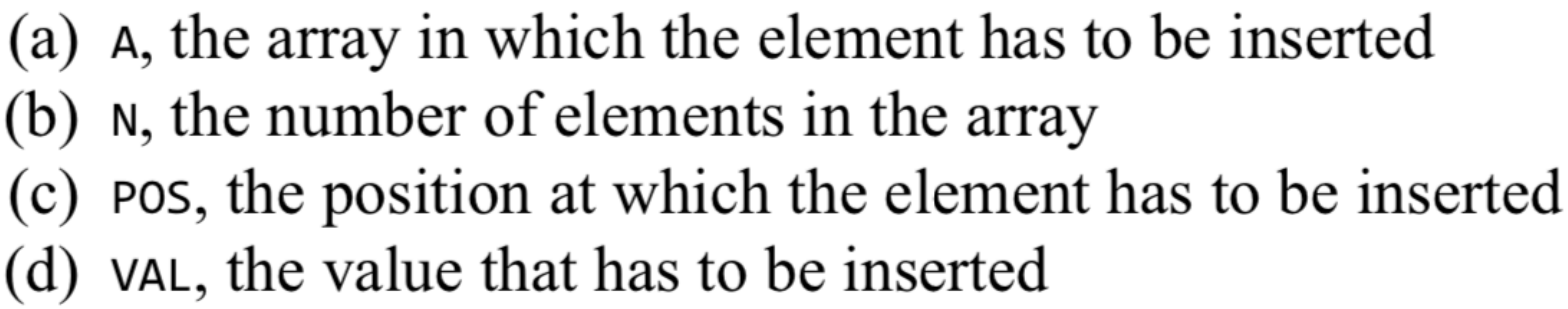


45	23	34	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

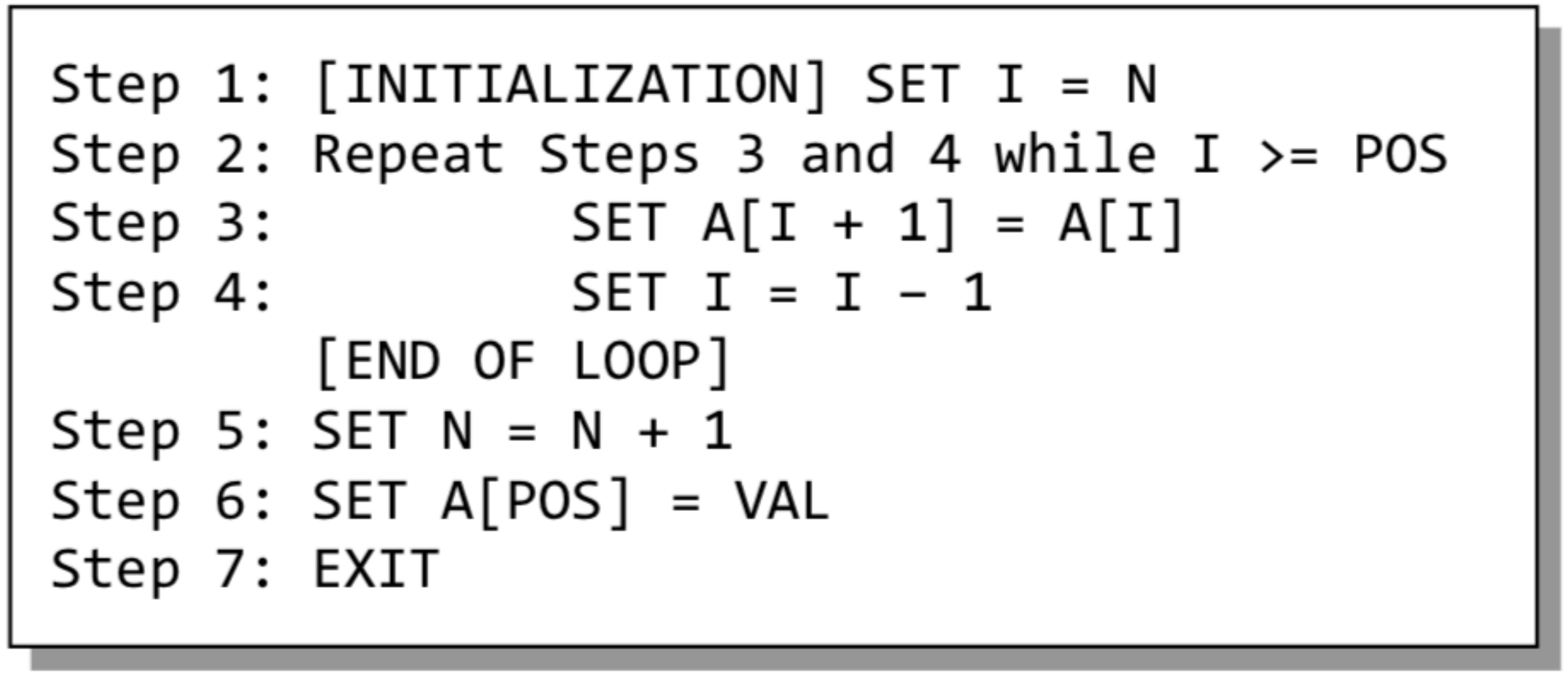
45	23	34	100	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

Array

Inserting an element in an array



(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

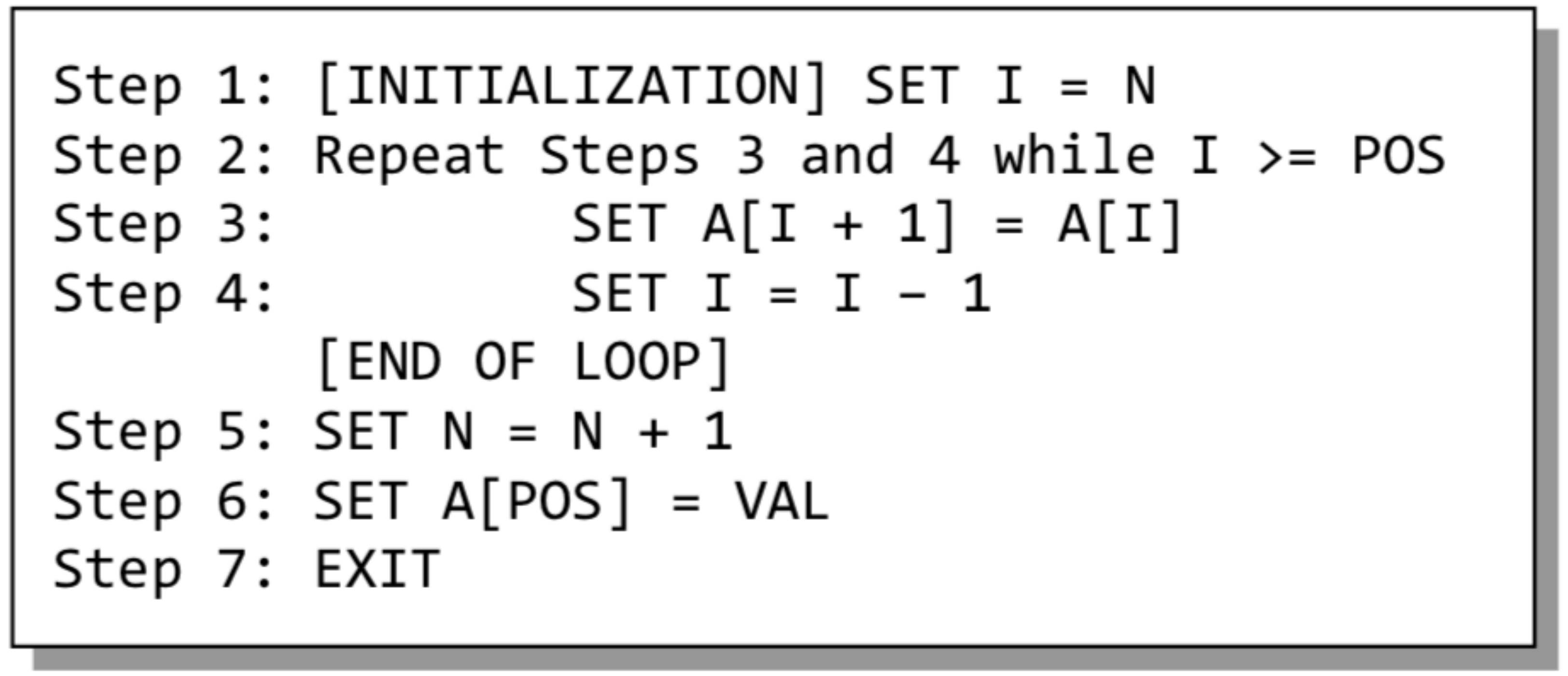


```
Step 1: [INITIALIZATION] SET I = N
Step 2: Repeat Steps 3 and 4 while I >= POS
Step 3:         SET A[I + 1] = A[I]
Step 4:         SET I = I - 1
        [END OF LOOP]
Step 5: SET N = N + 1
Step 6: SET A[POS] = VAL
Step 7: EXIT
```

Figure 3.14 Algorithm to insert an element in the middle of an array.

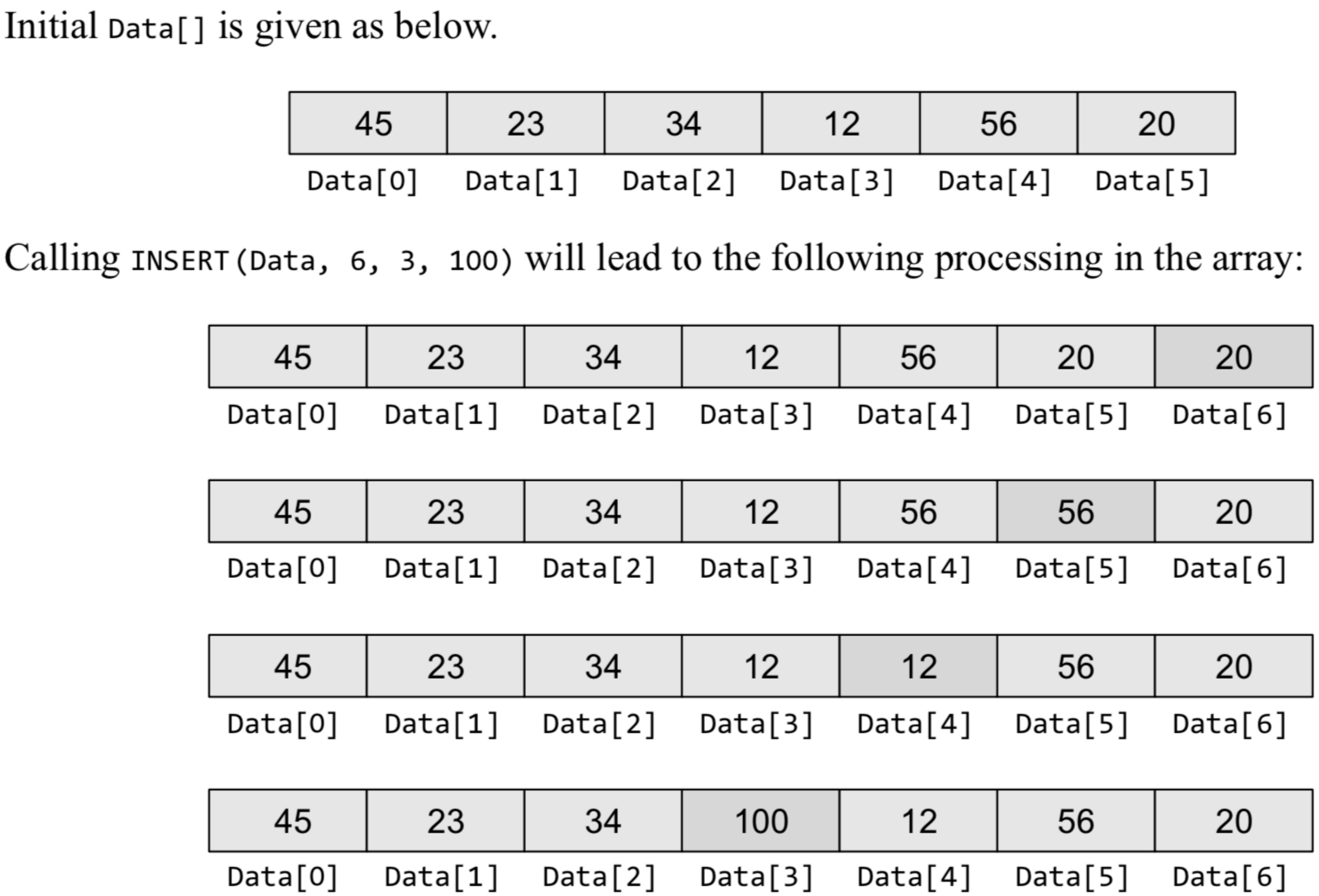
Array

Inserting an element in an array



```
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
```

Figure 3.14 Algorithm to insert an element in the middle of an array.



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]

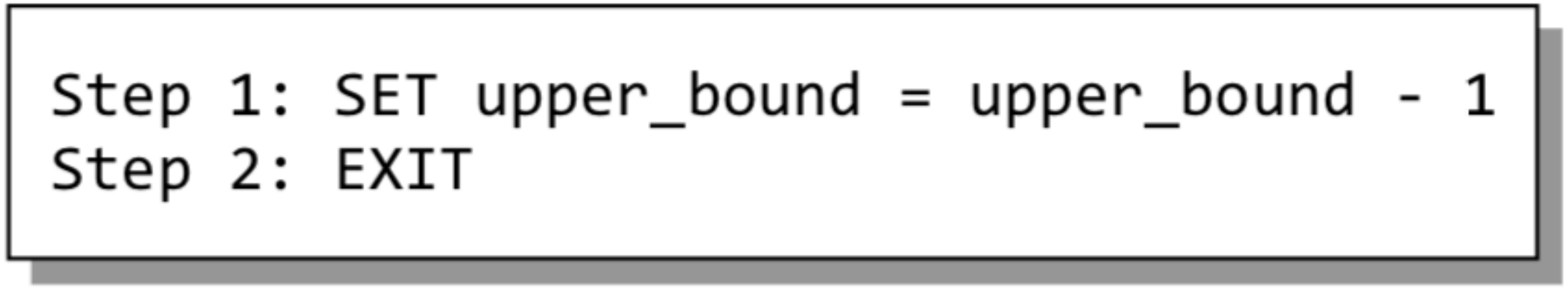
45	23	34	12	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

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

Array

Deleting an element from an array

- Deleting an element from an array means removing a data element from an already existing array. If the element has to be deleted from the end of the existing array, then the task of deletion is quite simple.



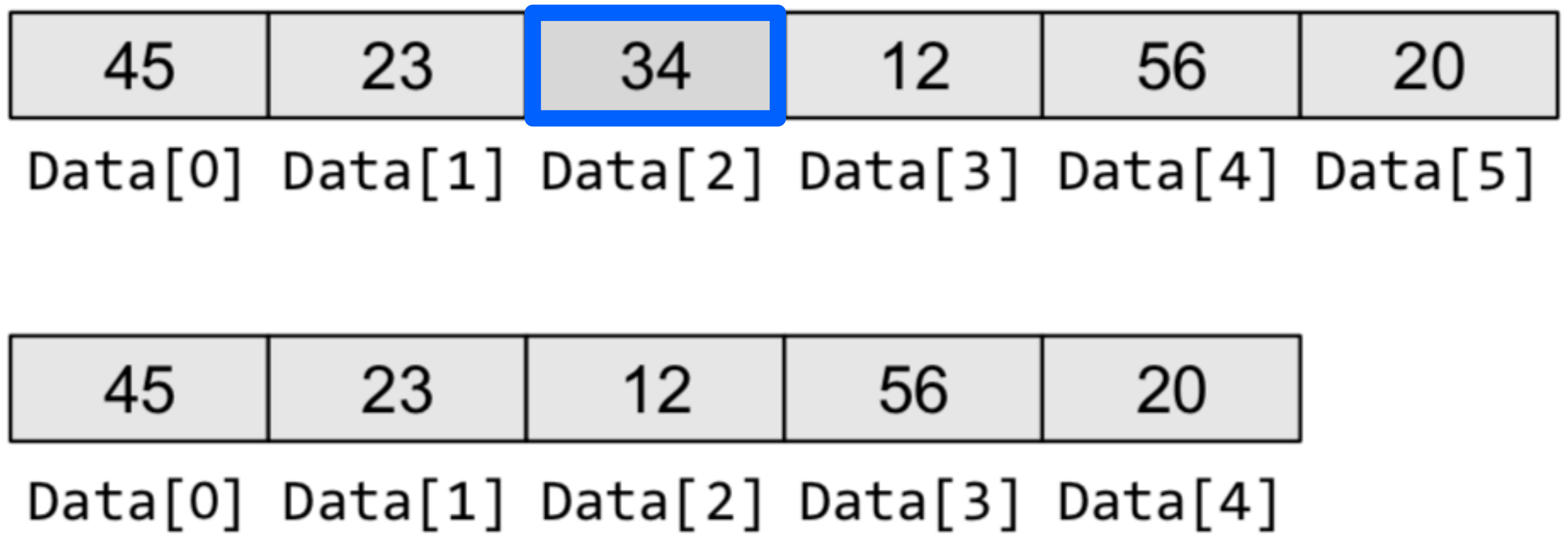
```
Step 1: SET upper_bound = upper_bound - 1
Step 2: EXIT
```

Figure 3.15 Algorithm to delete the last element of an array

Array

Deleting an element from an array

- If an element has to be inserted at the middle of an existing array, then …

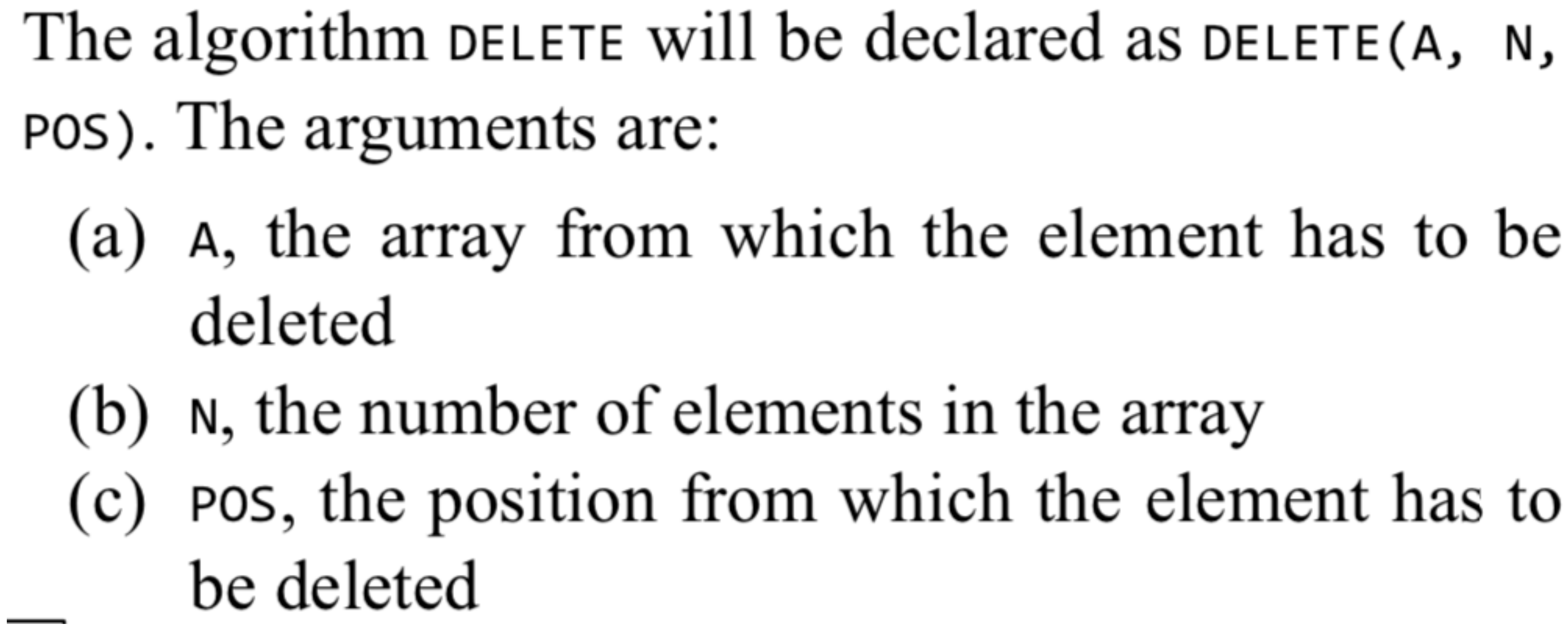


45	23	34	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

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

Array

Deleting an element from an array

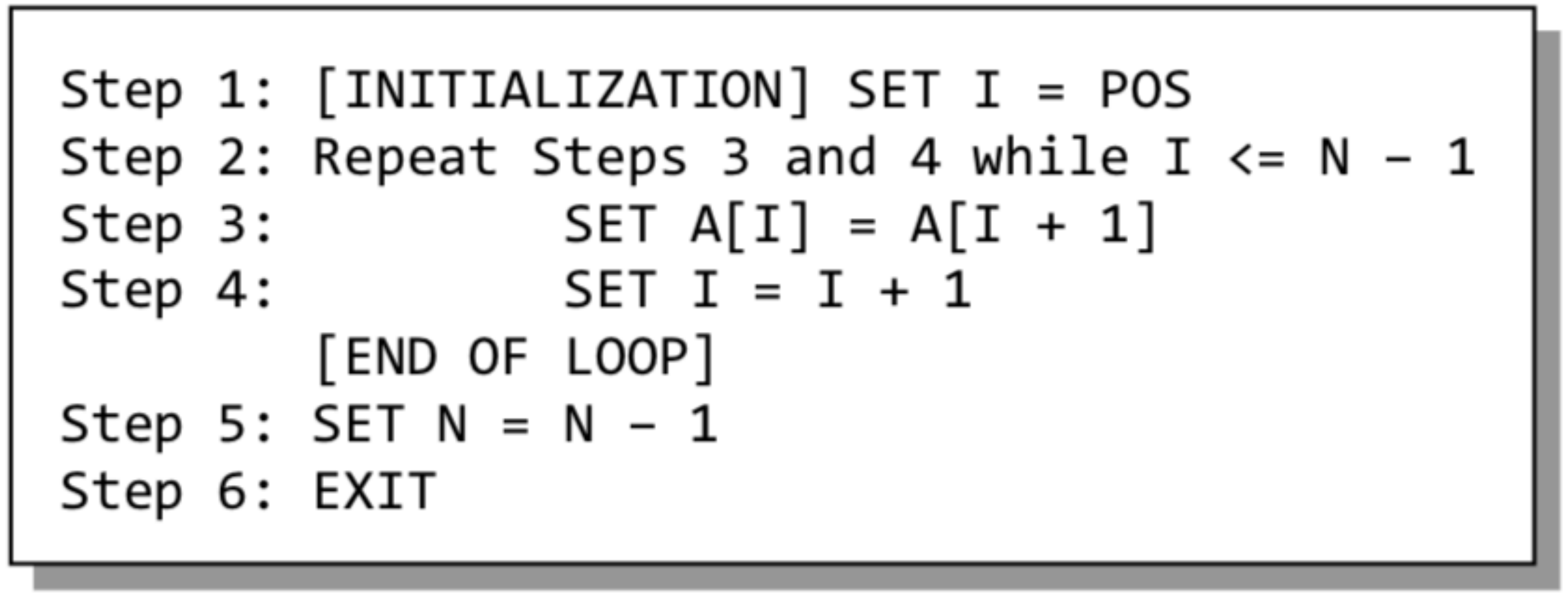


The algorithm DELETE will be declared as DELETE(A, N, POS). The arguments are:

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

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

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

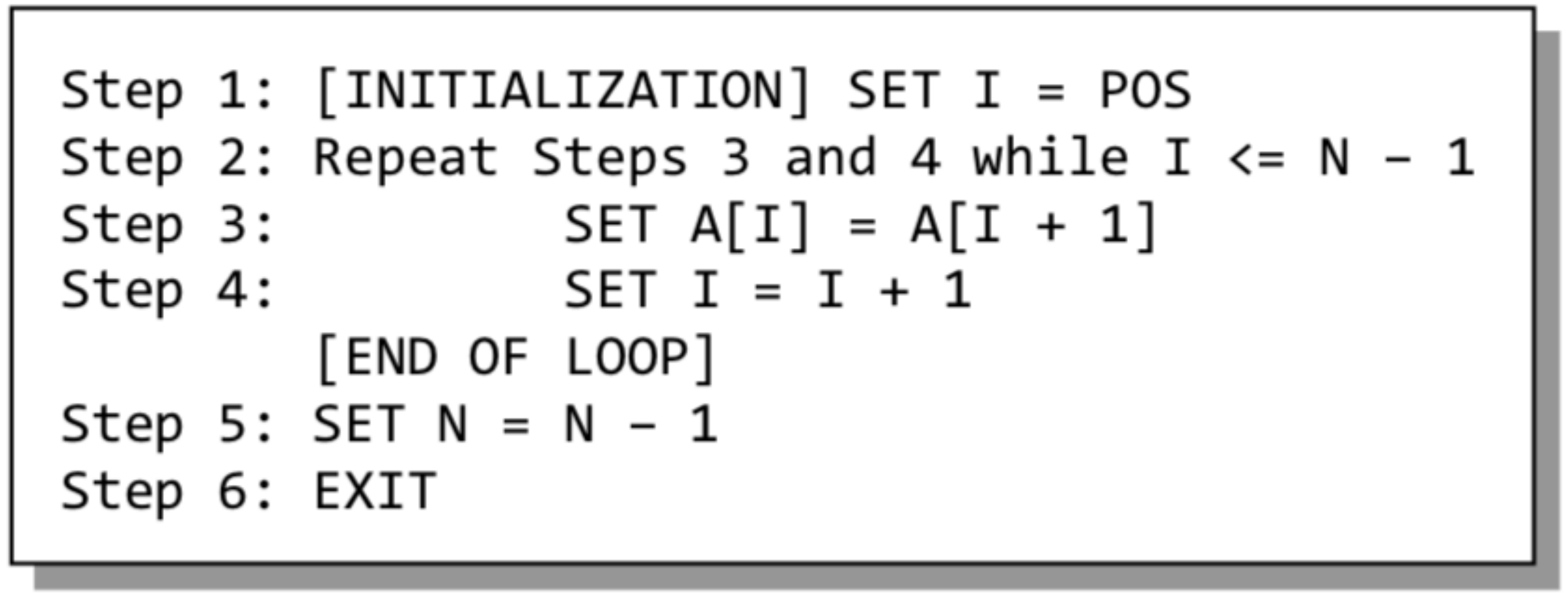


```
Step 1: [INITIALIZATION] SET I = POS
Step 2: Repeat Steps 3 and 4 while I <= N – 1
Step 3:         SET A[I] = A[I + 1]
Step 4:         SET I = I + 1
        [END OF LOOP]
Step 5: SET N = N – 1
Step 6: EXIT
```

Figure 3.16 Algorithm to delete an element from the middle of an array

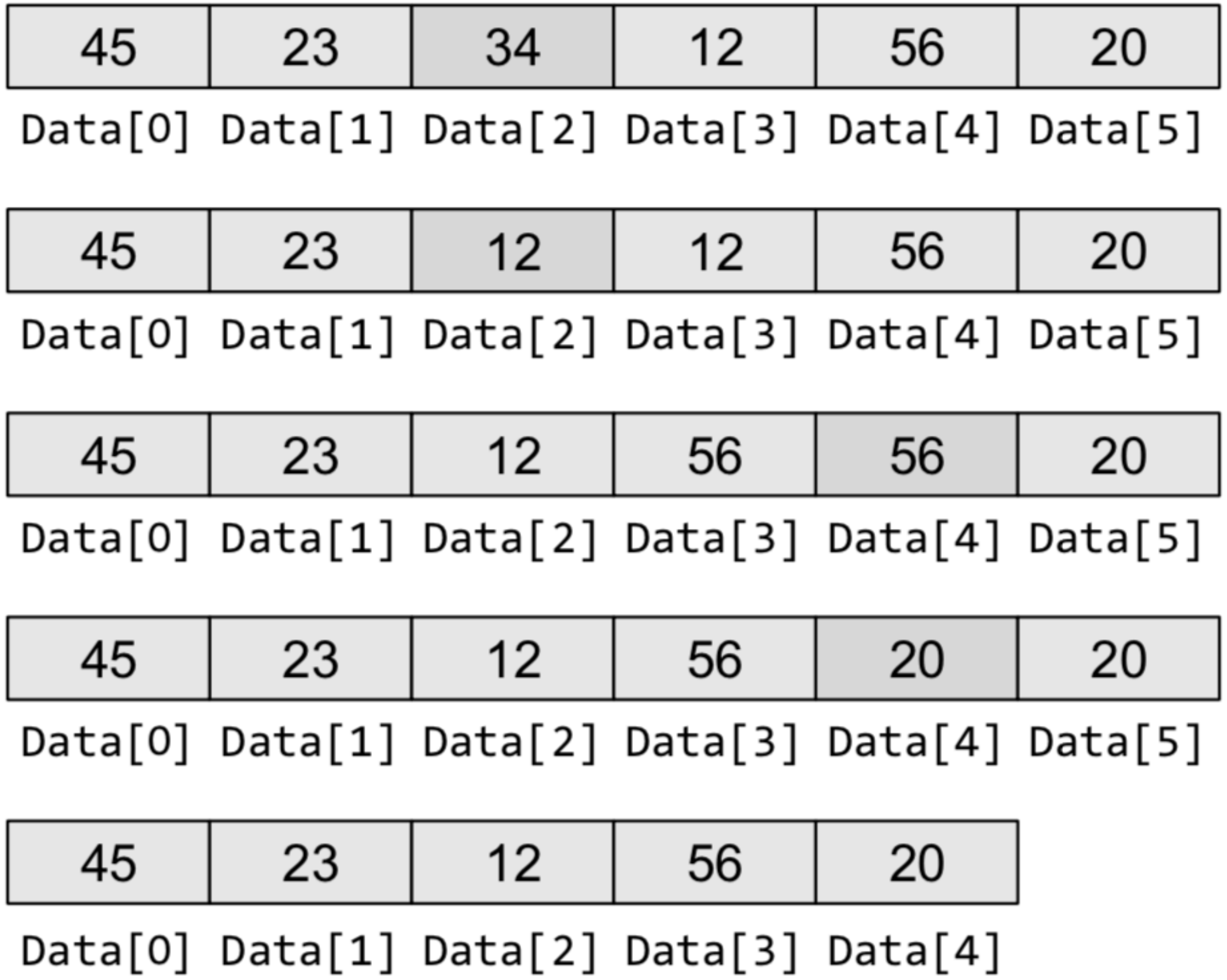
Array

Deleting an element from an array



```
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
```

Figure 3.16 Algorithm to delete an element from the middle of an array



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]

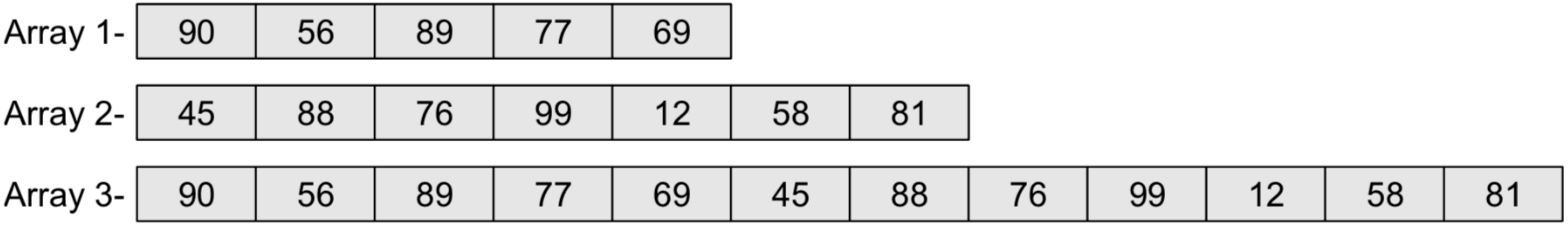
Figure 3.17 Deleting elements from an array

Array

Merging two arrays

- Merging two arrays in a third array means first copying the contents of the first array into the third array and then copying the contents of the second array into the third array.

- Hence, the merged array contains the contents of the first array followed by the contents of the 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

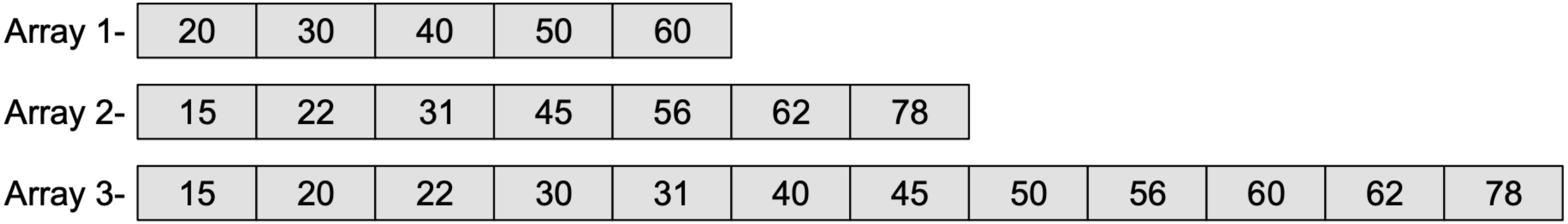
Figure 3.18 Merging of two unsorted arrays

Array

Merging two arrays

- If the arrays are unsorted, then merging the arrays is very simple, as one just needs to copy the contents of one array into another.

- But merging is not a trivial task when the two arrays are sorted and the merged array also needs to be sorted.



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

Array

Passing array to functions

- Like variables of other data types, we can also pass an array to a function.

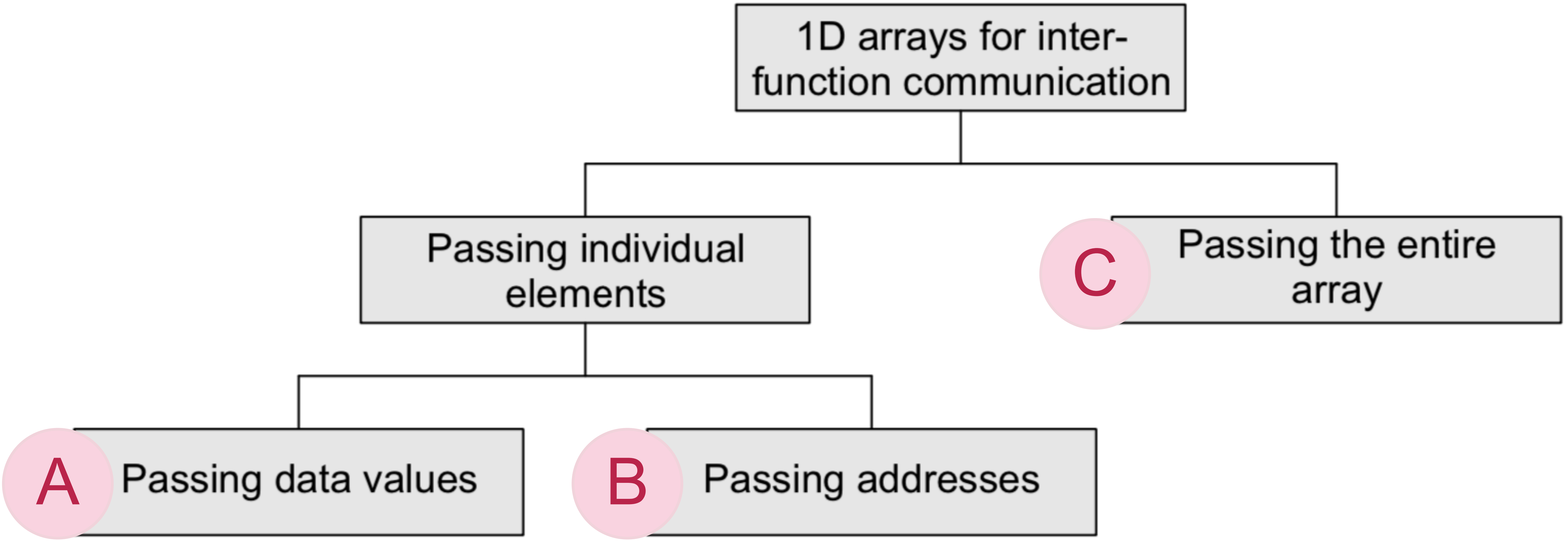

1D arrays for inter-function communication
Passing individual elements
C
Passing the entire array
A
Passing data values
B
Passing addresses

Figure 3.20 One dimensional arrays for inter-function communication

Array

Passing array to functions

```
int main()
{
    int arr[5] = {1,2,3,4,5};
    func1(arr[3]);
    func2(&arr[3]);
    func3(arr);
    return 0;
}
```

Address	Variable	Value
1000	arr[0]	1
1001		
1002	arr[1]	2
1003		
1004	arr[2]	3
1005		
1006	arr[3]	4
1007		
1008	arr[4]	5
1009		

A

```
void func1(int num)
{
    printf(“%d”, num);
}
```

B

```
void func2(int *num)
{
    printf(“%d”, *num);
}
```

C

```
void func3(int arr[])
{
    int i;
    for(i=0;i<5;i++)
        printf(“%d”, arr[i]);
}
```

Array

Pointer & array

Address	Variable	Value
1000	arr[0]	1
1001		
1002	arr[1]	2
1003		
1004	arr[2]	3
1005		
1006	arr[3]	4
1007		
1008	arr[4]	5
1009		
1010	ptr	
1011		
1012		
1013		

```
int main()
{
    int arr[5] = {1,2,3,4,5};
    printf(“%p %p %p \n”, arr, &arr, &arr[0]);

    int *ptr;
    ptr = &arr[0];
    printf(“%p %p”, ptr, ++ptr);
    return 0;
}
```

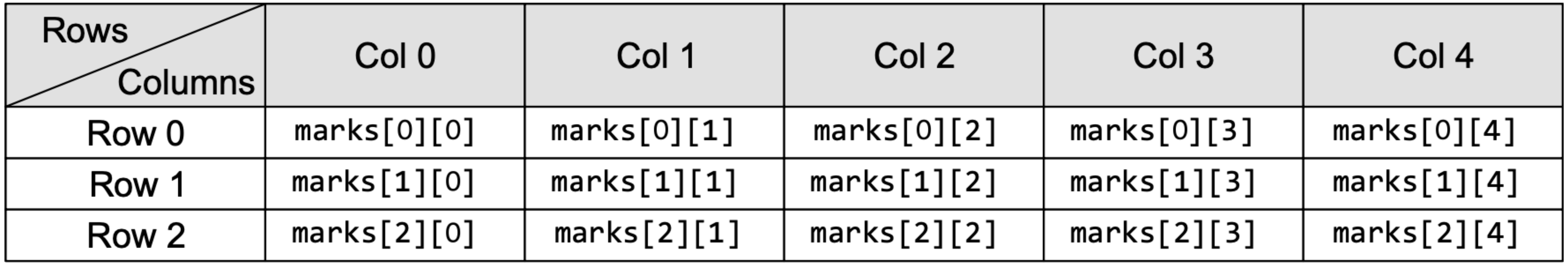
Array

2D array

- Declaration of array

```
data_type array_name[row_size][column_size];
```

```
int marks[3][5];
```



Rows Columns	Col 0	Col 1	Col 2	Col 3	Col 4
Row 0	marks[0][0]	marks[0][1]	marks[0][2]	marks[0][3]	marks[0][4]
Row 1	marks[1][0]	marks[1][1]	marks[1][2]	marks[1][3]	marks[1][4]
Row 2	marks[2][0]	marks[2][1]	marks[2][2]	marks[2][3]	marks[2][4]

Array

2D array

- Accessing the Elements of Array
- Storing Values in Array
- Operations on Array
- Pointer & Array

```
&arr[0][0] + 1  point to  arr[0][1]
&arr[0] + 1     point to  arr[1][0]
arr[0] + 1      point to  …
arr + 1         point to  …
```

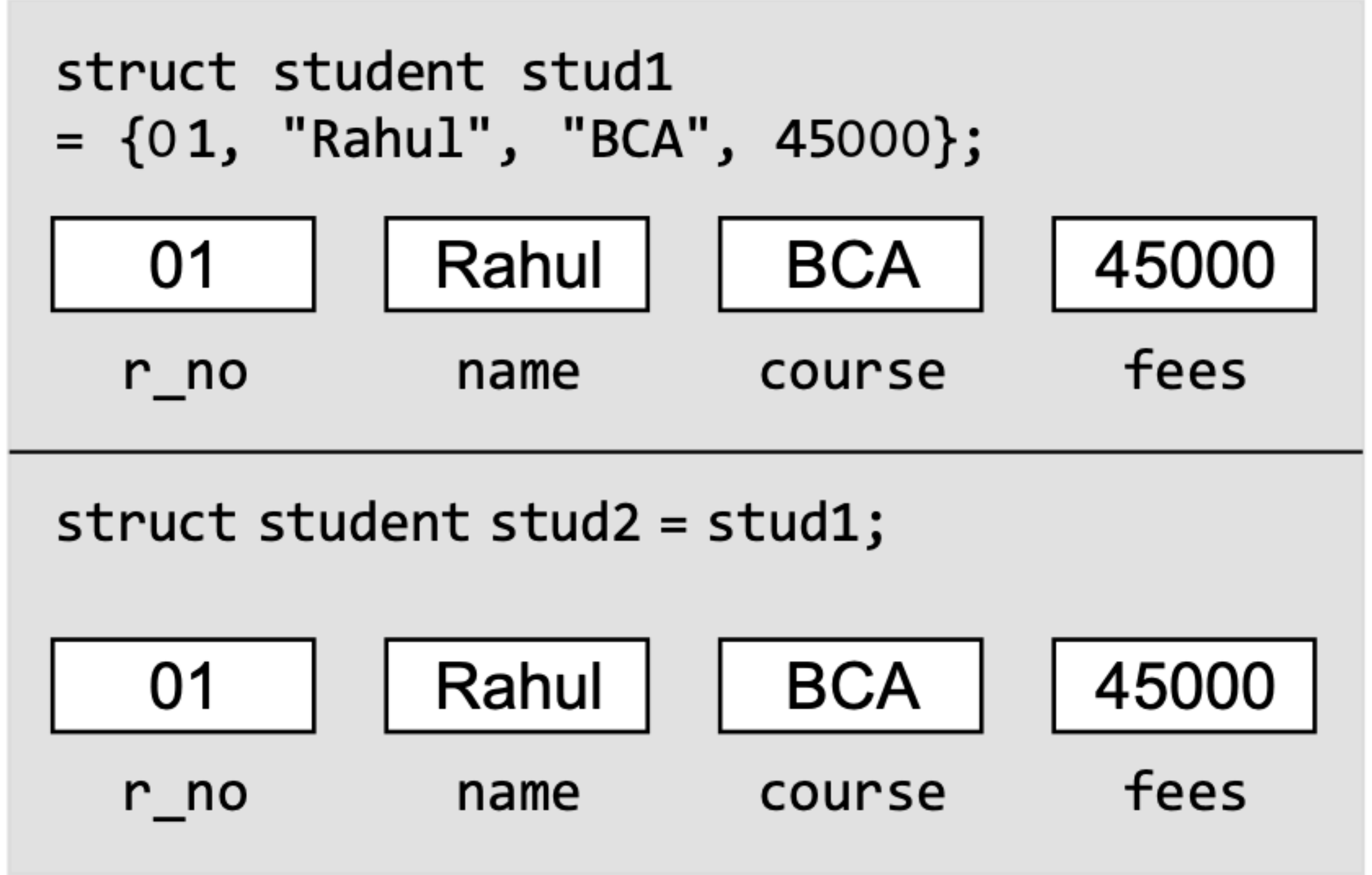
String

- Reading & Writing String
- Operations on Strings
  - Finding length of a string
  - Converting characters of a string into upper/lower case
  - Appending a string into another string
  - Comparing two strings
  - Reversing a string
  - Inserting/Deleting a string in the main string
  - Pattern matching
- Arrays of Strings

Structure

- Declaration
- Accessing the member of a structure
- Copying structures
- Nested structures
- Arrays of structures
- Self-referential structures

```
struct struct-name
{
    data_type var_name;
    data_type var_name;
    ..................................
};
```



```
struct student stud1
= {01, "Rahul", "BCA", 45000};
```

01	Rahul	BCA	45000
r_no	name	course	fees

```
struct student stud2 = stud1;
```

01	Rahul	BCA	45000
r_no	name	course	fees

```
struct node
{
    int val;
    struct node *next;
};
```

Union

- In case of unions, you can only store information in one field at any one time.
- Unions are used to save memory. They are useful for applications that involve multiple members, where values need not to be assigned to all the members at any one time.

Address	Variable
1000	abc.a
1001	
1002	abc.b

```
struct abc
{
    int a;
    char b;
};
```

```
a’s address = 1000
b’s address = 1002
```

Address	Variable
1000	
1001	
1002	

```
union abc
{
    int a;
    char b;
};
```

```
a’s address = 1000
b’s address = 1000
```

Union

```
typedef struct POINT1
{
      int x, y;
};
typedef union POINT2
{
      int x;
      int y;
};
int main()
{
      POINT1 P1 = {2,3};
      // POINT2 P2 ={4,5}; Illegal in case of unions
      POINT2 P2;
      P2.x = 4;
      P2.y = 5;
      printf("\n The coordinates of P1 are %d and %d", P1.x, P1.y);
      printf("\n The coordinates of P2 are %d and %d", P2.x, P2.y);
      return 0;
}
```

Output

```
The coordinates of P1 are 2 and 3
The coordinates of P2 are 5 and 5
```

Union

```
typedef struct
{
    char *name;
    bool is_robot;
    char *personality;
    int firmware_version;
}game_character;
```

```
typedef struct
{
    char *name;
    bool is_robot;
    union{
       char *personality;
       int firmware_version;
    };
}game_character;
```

Wrap up

- Review of arrays
- Brief review of string & structure
- Introduction to union