MODULE 1-2 [CPE11201]

ARRAY

อาร์เรย์

Assoc. Prof. Dr. Natasha Dejdumrong

OUTLINES

- Introduction to Array
- Declaration of Array
- Accessing the Elements of Array
- Storing Values in Array

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.

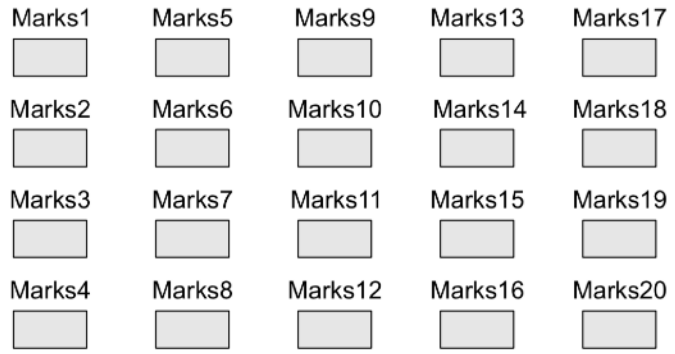


Figure 3.1 Twenty variables for 20 students

ARRAY

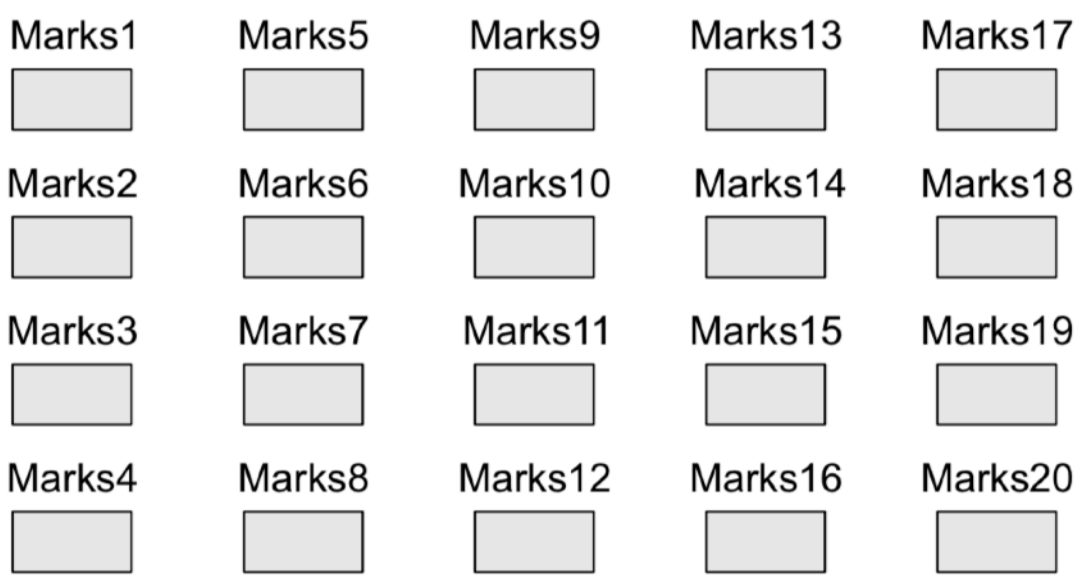


Figure 3.1 Twenty variables for 20 students

Declaration of Array

★ Arrays are declared using the following syntax:

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

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

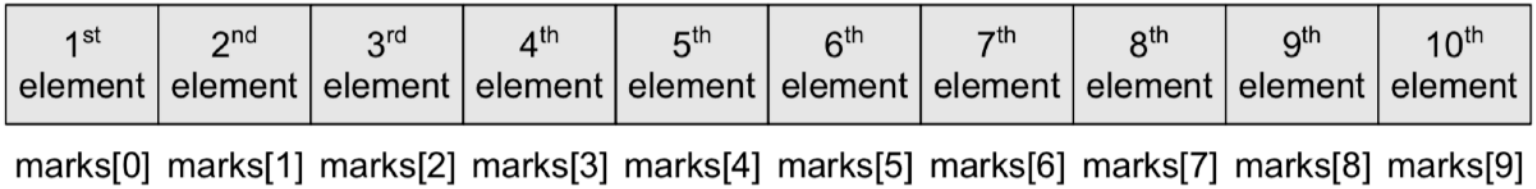


Figure 3.2 Memory representation of an array of 10 elements

Declaration of Array

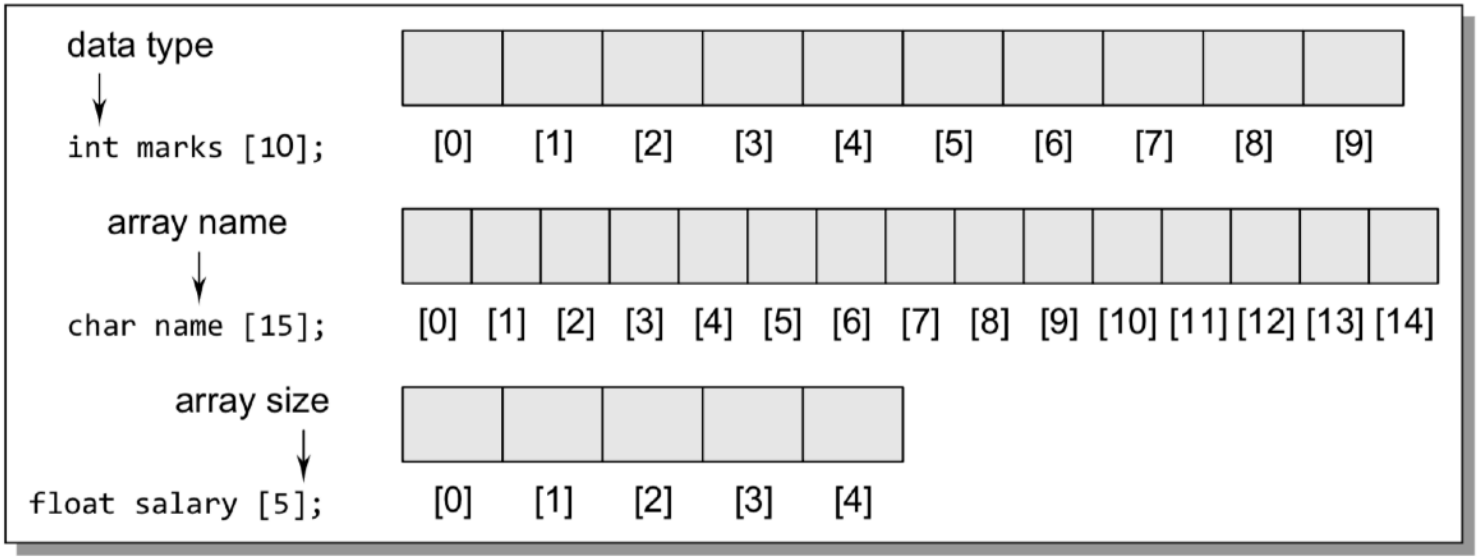


Figure 3.3 Declaring arrays of different data types and sizes

Accessing the Elements of Array

★ Arrays are declared using the following syntax:

```
// 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

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

Calculating the address of array elements

★ The formula to perform this calculation is,

int : 2 bytes

ต.น.ที่ต้องการ ต.น.ตัวแรก size of datatype

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

the memory address of the first element of the array.

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.

Calculating the address of array elements

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

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
```

Calculating the Length of an Array

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.

Calculating the Length of an Array

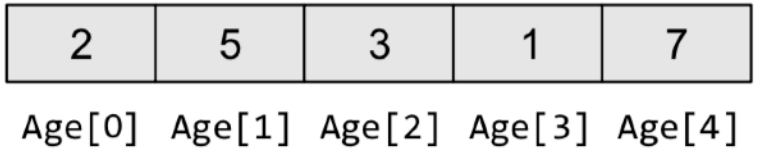
Example 3.2 Let `Age[5]` be an array of integers such that

`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.



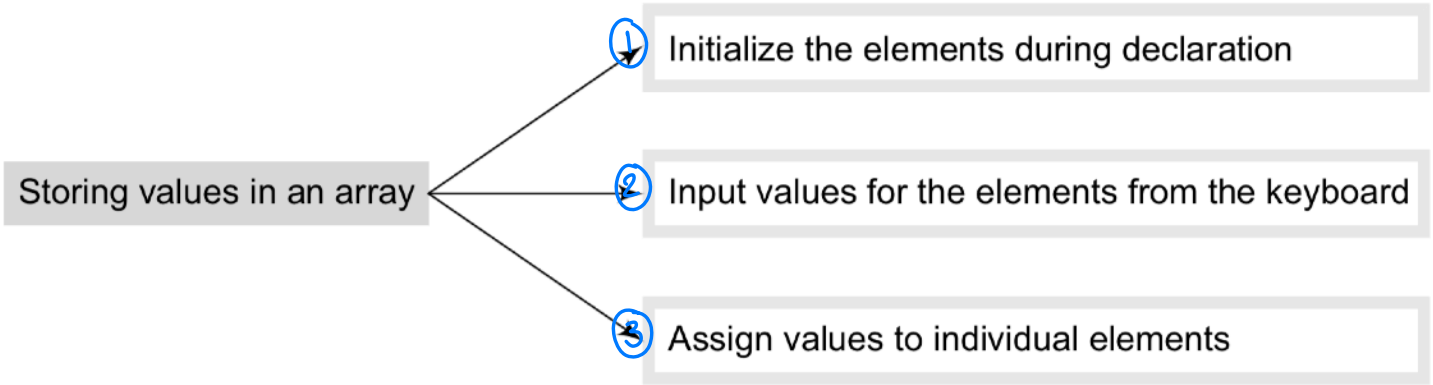

```
Length = upper_bound – lower_bound + 1
```

Here, `lower_bound = 0, upper_bound = 4`

Therefore, `length = 4 – 0 + 1 = 5`

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 three ways to store values in an array.



Initializing Arrays during Declaration

1

★ 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.

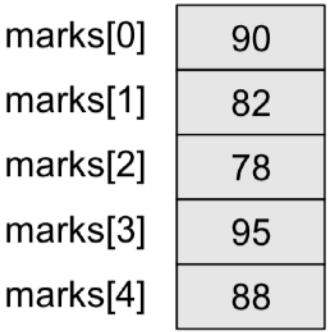


Figure 3.7 Initialization of array `marks[5]`

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

Initializing Arrays during Declaration

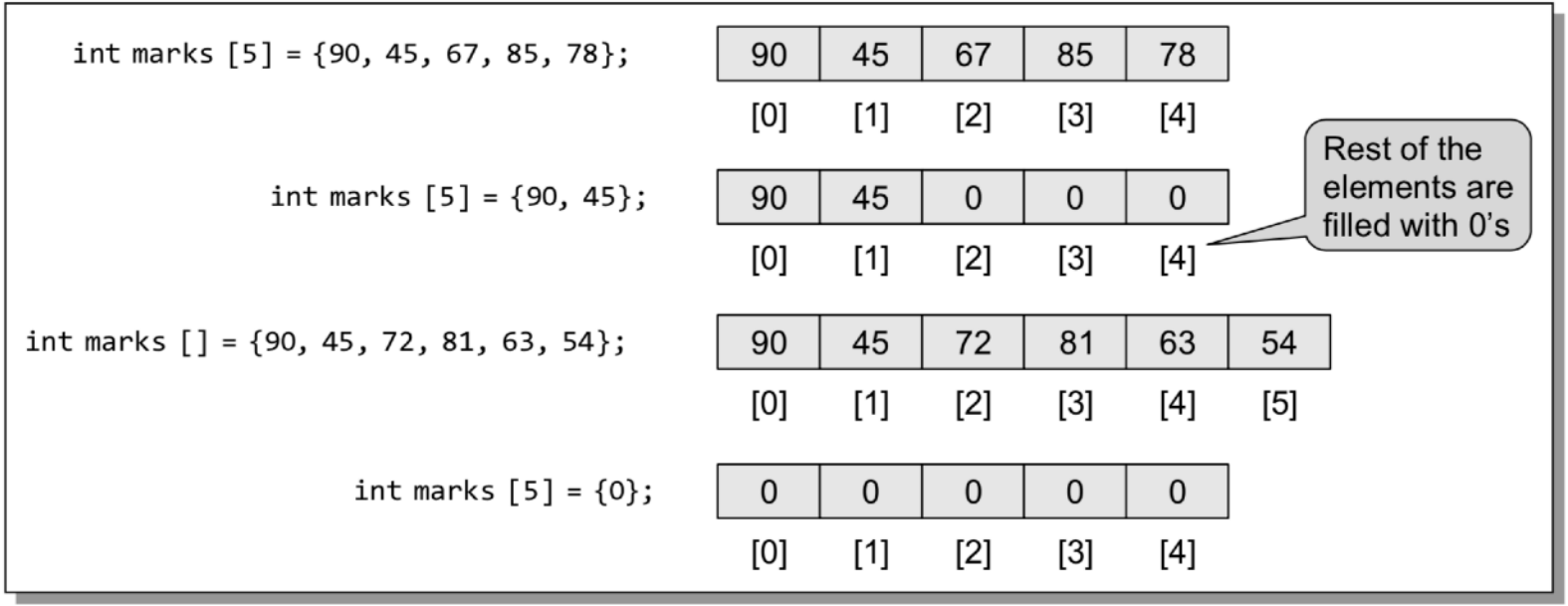


Figure 3.8 Initialization of array elements

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

Assigning Values to Individual Elements

3

```
int i, arr1[10], arr2[10];
arr1[10] = {0,1,2,3,4,5,6,7,8,9};
for(i=0;i<10;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[10];
for(i=0;i<10;i++)
        arr[i] = i*2;
```

Figure 3.11 Code for filling an array with even numbers

Operations on Arrays

- There are a number of operations that can be preformed on arrays. These operations include:
  - 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.

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

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
```

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
```

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

Inserting an Element in an Array

Example 3.3 `Data[]` is an array that is declared as `int Data[20];` and contains the following values:

```
Data[] = {12, 23, 34, 45, 56, 67, 78, 89, 90, 100};
```

(a) Calculate the length of the array.
(b) Find the `upper_bound` and `lower_bound`.
(c) Show the memory representation of the array.
(d) If a new data element with the value 75 has to be inserted, find its position.
(e) Insert a new data element 75 and show the memory representation after the insertion.

Solution

(a) Length of the array = number of elements
Therefore, `length of the array = 10`

(b) By default, `lower_bound = 0` and `upper_bound = 9`

Inserting an Element in an Array

Example 3.3 `Data[]` is an array that is declared as `int Data[20];` and contains the following values:

```
Data[] = {12, 23, 34, 45, 56, 67, 78, 89, 90, 100};
```

(a) Calculate the length of the array.
(b) Find the `upper_bound` and `lower_bound`.
(c) Show the memory representation of the array.
(d) If a new data element with the value 75 has to be inserted, find its position.
(e) Insert a new data element 75 and show the memory representation after the insertion.

(c)

12	23	34	45	56	67	78	89	90	100
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]	Data[8]	Data[9]

(d) Since the elements of the array are stored in ascending order, the new data element will be stored after 67, i.e., at the 6th location. So, all the array elements from the 6th position will be moved one position towards the right to accommodate the new value

(e)

12	23	34	45	56	67	75	78	89	90	100
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]	Data[8]	Data[9]	Data[10]

Insert an Element in the Middle of 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.

Insert an Element in the Middle of 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]

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

Deleting an Element from an Array

Example 3.4 `Data[]` is an array that is declared as `int Data[10];` and contains the following values:

```
Data[] = {12, 23, 34, 45, 56, 67, 78, 89, 90, 100};
```

(a) If a data element with value 56 has to be deleted, find its position.

(b) Delete the data element 56 and show the memory representation after the deletion.

Solution

(a) Since the elements of the array are stored in ascending order, we will compare the value that has to be deleted with the value of every element in the array. As soon as `VAL = Data[I]`, where `I` is the index or subscript of the array, we will get the position from which the element has to be deleted. For example, if we see this array, here `VAL = 56`. `Data[0] = 12` which is not equal to 56. We will continue to compare and finally get the value of `POS = 4`.

(b)

12	23	34	45	67	78	89	90	100
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]	Data[8]

Delete an Element from the Middle of 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

Delete an Element from the Middle of 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

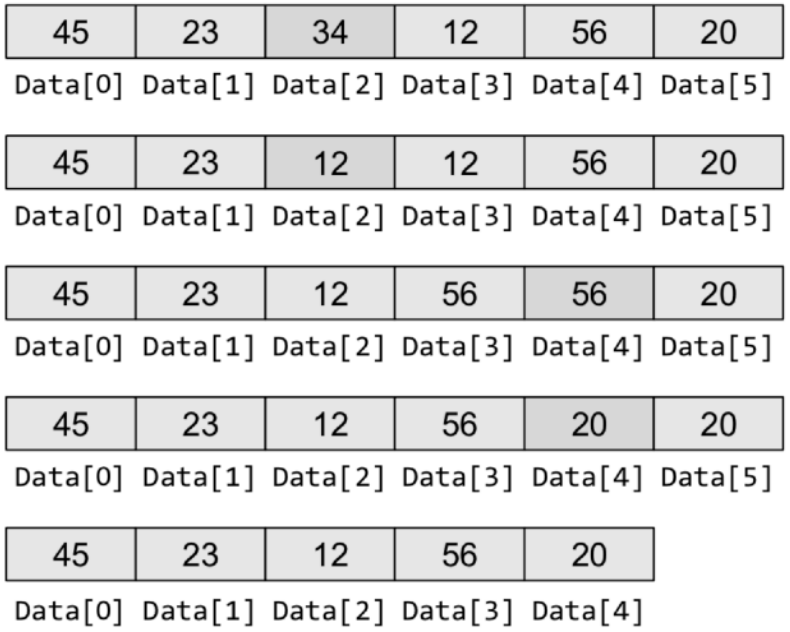


Figure 3.17 Deleting elements from an array

Delete an Element from the Middle of Array

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

```
#include <stdio.h>
#include <conio.h>
int main()
{
        int i, n, pos, arr[10];
        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("\nEnter the position from which the number has to be deleted : ");
        scanf("%d", &pos);
        for(i=pos; i<n-1;i++)
                arr[i] = arr[i+1];
        n--;
        printf("\n The array after deletion is : ");
        for(i=0;i<n;i++)
                printf("\n arr[%d] = %d", i, arr[i]);
        getch();
        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
Enter the position from which the number has to be deleted : 3
The array after deletion is :
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 5
```

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.
- 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.

Merging two Arrays

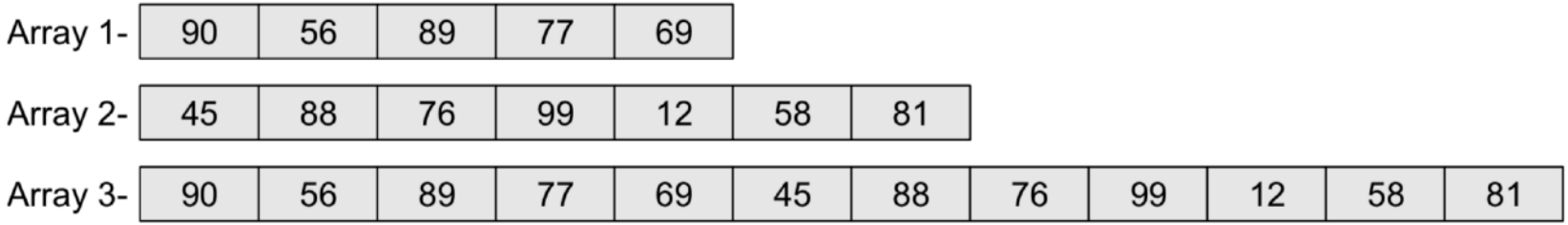


Figure 3.18 Merging of two unsorted arrays

Passing Array to Functions

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

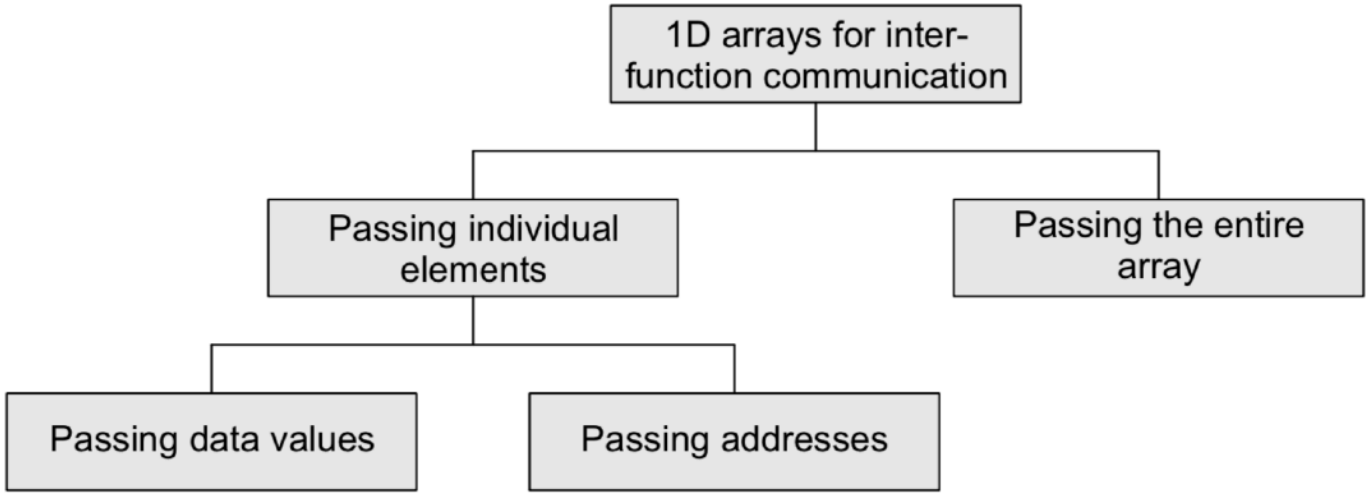


Figure 3.20 One dimensional arrays for inter-function communication

Questions and Answers