Introducing Arrays

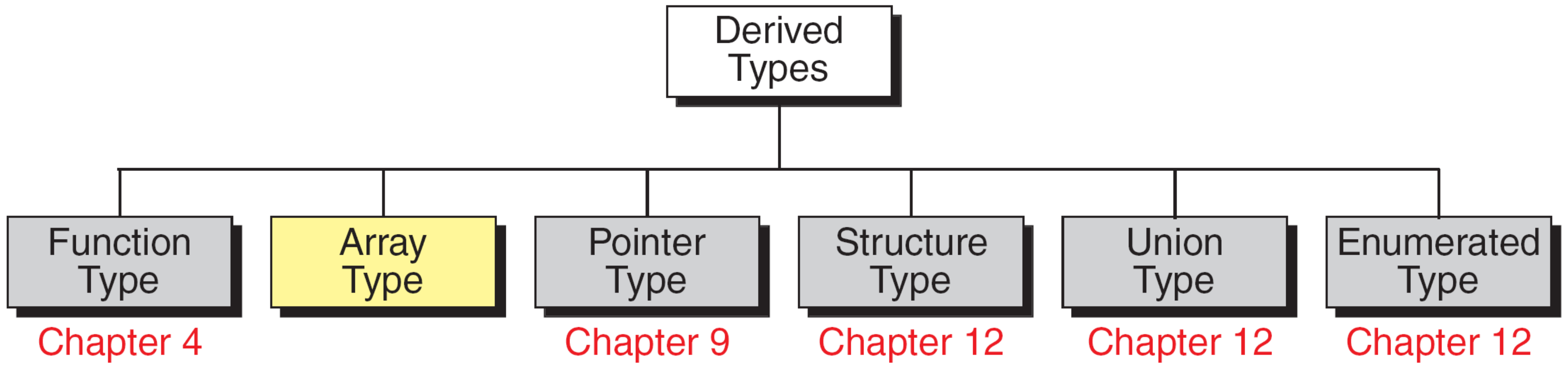
Goals

By the end of this unit, you should understand …

- … how to create and use arrays.
- … how to process arrays using for loops.
- … how to pass arrays to functions.
- … how to create and use two-dimensional arrays.

What is an Array?

- An *array* is a sequenced collection of elements that share the same data type.
- Elements in an array share the same name.



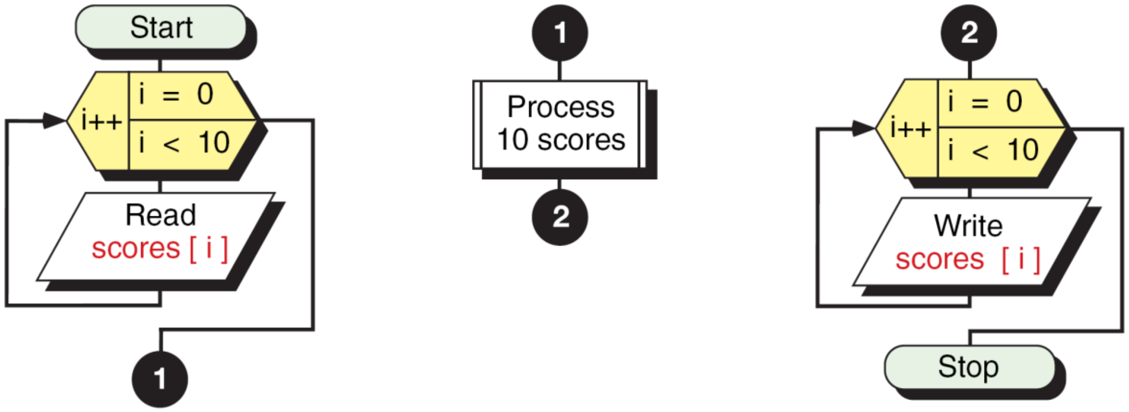
from Figure 8-1 in* Forouzan & Gilberg*, p. 459

Referencing Array Elements

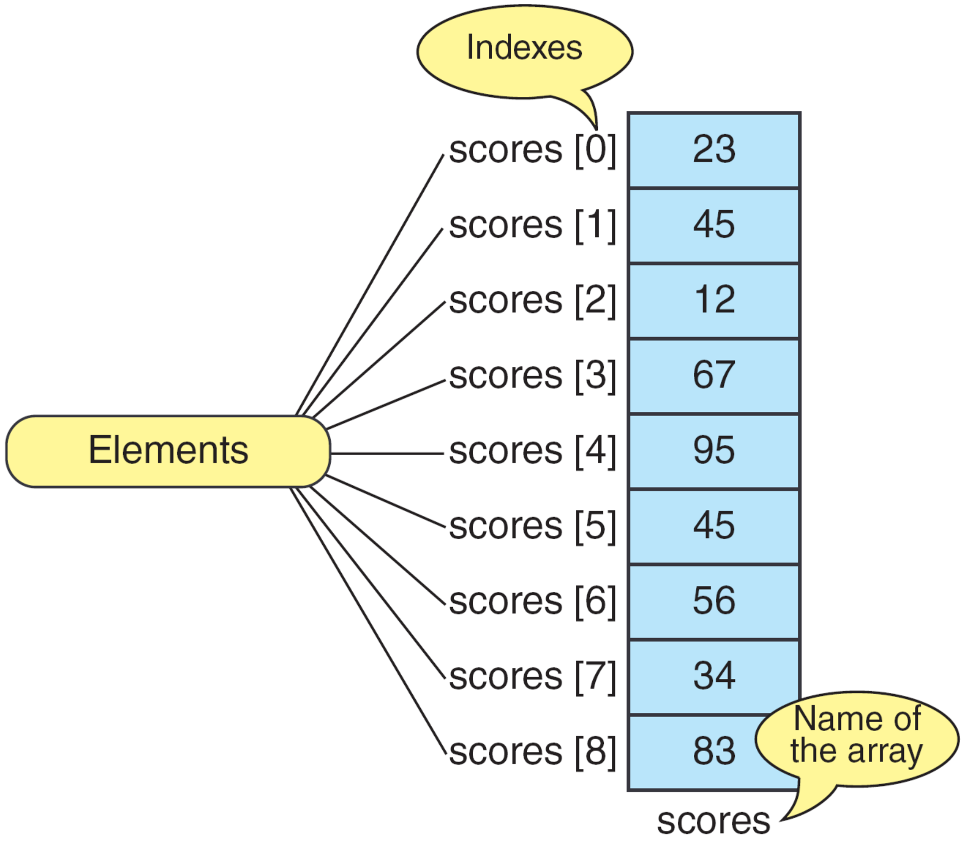
- We can reference array elements by using the array's subscript. The first element has a subscript of 0. The last element in an array of length ***`n`*** has a subscript of ***`n-1`***.
- When we write a program, we refer to an individual array element using *indexing*. To index a subscript, use the array name and the subscript in a pair of square brackets: **`anyArray[12];`**

Arrays and Loops

- Since we can refer to individual array elements using numbered indexes, it is very common for programmers to use **`for`** loops when processing arrays.



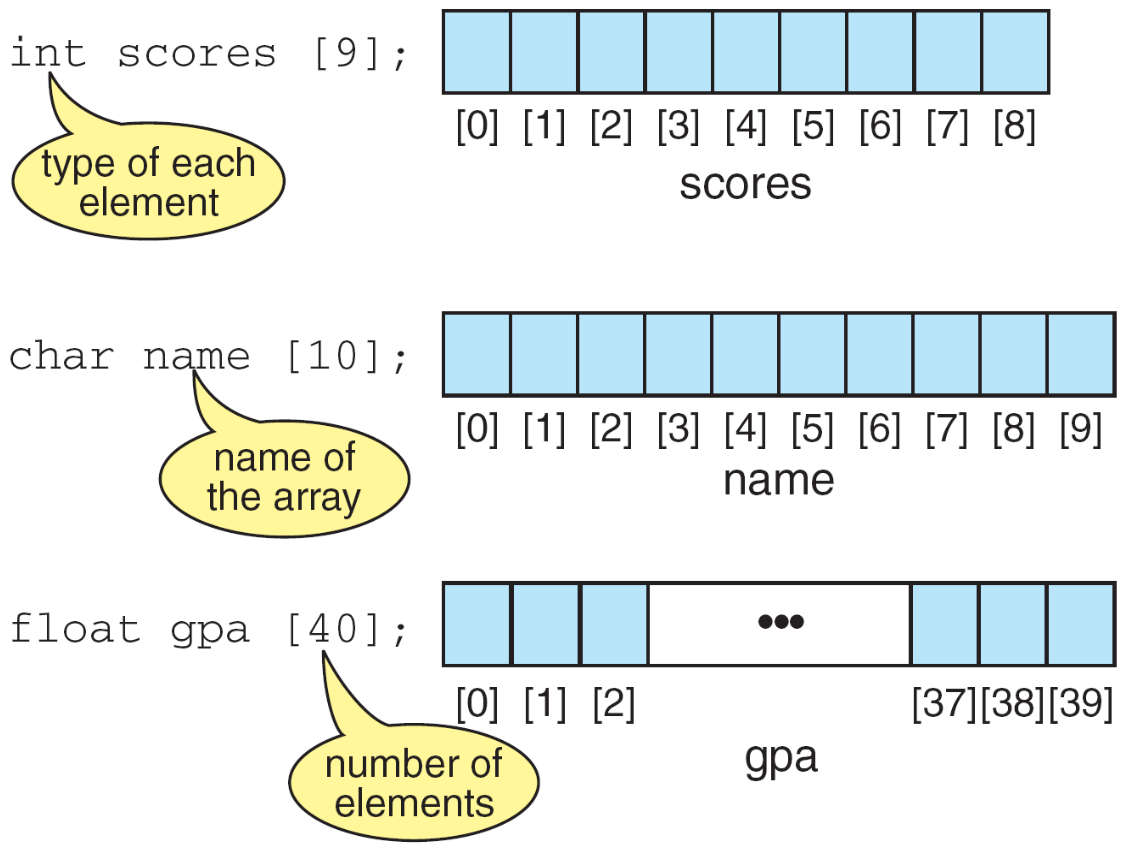
Array Example



Declaring an Array

- To declare an array, we need to specify its data type, the array's identifier and the size:
  ```
  type arrayName[arraySize];
  ```
- Before using an array (even if it is a variable-length array), we must declare and initialize the arraySize!

Declaration Examples



Accessing Elements

- To access an array’s element, we need to provide an integral value to identify the index we want to access.
- Array elements range from 0 to arraySize-1.
- We can also use a variable:

```
for(i = 0; i < 9; i++)
{
    scoresSum += scores[i];
}
```

- **`Array name is actually a pointer to the first element!`**

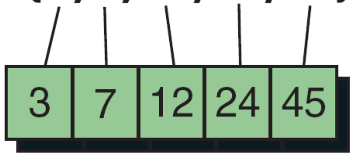
Initialization at Array Definition

- We can initialize array elements when we define an array.
- If we initialize fewer values than the length of the array, C assigns zeroes to the remaining elements.

Array Initialization Examples

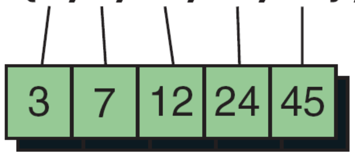
(a) Basic Initialization

```
int numbers[5] = {3,7,12,24,45};
```



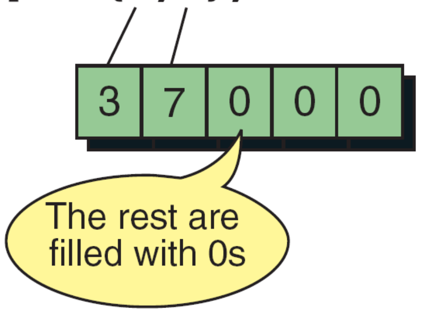
(b) Initialization without Size

```
int numbers[ ] = {3,7,12,24,45};
```



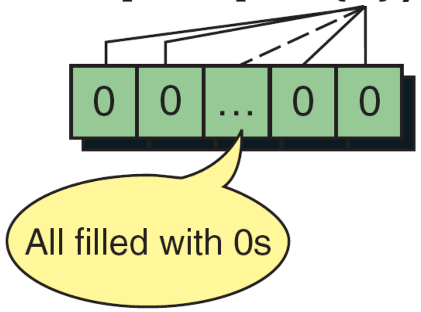
(c) Partial Initialization

```
int numbers[5] = {3,7};
```



(d) Initialization to All Zeros

```
int lotsOfNumbers [1000] = {0};
```



Inputting Values using a `for` Loop

- Once we know the length of an array, we can input values using a `for` loop:

```
arrayLength = 9;
for (j=0; j<arrayLength; j++)
{
   scanf("%d", &scores[j]);
}
```

Assigning Values to Individual Elements

- We can assign any value that reduces to an array’s data type:

```
scores[5] = 42;
scores[3] = 5 + 13;
scores[8] = x + y;
scores[0] = pow(7, 2);
```

“Copying” Arrays

- We **cannot** directly copy one array to another, even if they have the same length and share the same data type.
- Instead, we can use a `for` loop to copy values:

```
for(m = 0; m < 25; m++)
{
   array2[m] = array1[m];
}
```

Swapping Array Elements

- To swap (or exchange) values, we **must** use a temporary variable. A common novice’s mistake is to try to assign elements to one another:

```
/*The following is a logic error*/
numbers[3] = numbers[1];
numbers[1] = numbers[3];

/*A correct approach …*/
temp = numbers[3];
numbers[3] = numbers[1];
numbers[1] = temp;
```

Printing Array Elements

- To print an array’s contents, we would use a **`for`** loop:

```
for(k = 0; k < 9; k++)
{
    printf(“%d”, scores[k];
}
```

Range Checking

- Unlike some other languages, C does not provide built-in range checking. Thus, it is possible to write code that will produce “out-of-range” errors, with unpredictable results.
- Common Error (array length is 9):

```
for(j = 1; j <= 9; j++)
{
   scanf(“%d”, &scores[j]);
}
```

Code Examples of Arrays

- Examples:

PrintArrayUsingFor.c
SquaresArray.c
ReversePrint.c

Passing Elements to Functions

- We can pass the array elements directly to the function, but the data type of that element should match the data type of the formal arguments of the function. So, if we want to pass that element, we may pass it directly to a function:

```
AnyFunction(myArray[4]);
…
void AnyFunction(int anyValue)
{
    …
}
```

Passing Elements to Functions

- We may also pass the address of an element:

```
anyFunction(&myArray[4]);
…
void anyFunction(int*
anyValue)
{
   *anyValue = 15;
}
```

Arrays are Passed by Reference

- Unlike individual variables or elements, which we pass by value to functions, C passes arrays to functions by reference.
- The array’s name identifies the address of the array’s first element.
- To reference individual elements in a called function, use the indexes assigned to the array in the calling function.

Passing an Array

- When we pass an array to a function, we need only to pass the array's name:

```
AnyFunction(myArray);
```

- In the called function, when can declare the formal parameter for the array using empty brackets (preferred method):

```
void AnyFunction(int passArray[ ]);
```

- or, we can use a pointer:

```
void AnyFunction(int* passArray);
```

Protecting Array Elements

- To protect from the called function changing array elements, we need to specify the formal parameter in the called function that represents the array as a constant:

```
void AnyFunction(const int passArray[ ]);
```

More Code Examples of Arrays

- Examples:

AvgFixedArray.c
AvgVariableArray.c
ChangeArrValues.c

Allowed operations

- These operations include the following, for an array named ‘ar’.

(a) To increment the ith element:-

```
ar[i]++;
ar[i] += 1;
ar[i] = ar[i] + 1;
```

(b) To add n to the ith element :-

```
ar[i] += n;
ar[i] = ar[i] + n;
```

(c) To copy the contents of the ith element to the kth element:-

```
        ar[k] = ar[i]
```

(d) To copy the contents of one array ‘ar’ to another array ‘br’, it must again be done one by one:-

```
int ar[10], br[10];
for (i=0;i<10;i++)
        br[i] = ar[i];
```

(e) To exchange the values of ar[i] and ar[k] a temporary variable must be declared to hold one value and it should be the same data type as the array elements being swapped:-

```
        int temp;
        temp = ar[i];
        ar[i] = ar[k];
        ar[k] = temp;
```

Searching an element within an array

- Consider an array of n elements, where each element is a number. The task is to find a particular number in the array.

```
#include<stdlib.h>
main()
{
    int a[30], n, i,key, FOUND=0;

    printf(“\n Enter the array elements one by one \n”);
    for (i=0; i<30;i++)
            scanf(“%d”, &a[i]);

    printf(“\n Enter the key to be searched \n”);
    scanf(“%d”, &key);

    for (i=0;i<30;i++)
            if (a[i] = = key)
            {
                        printf(“\n Fount at %d”, i);
                        FOUND=1;
            }
            if (FOUND==0)
                        printf(“\n NOT FOUND…..”);
}
```

Sorting an array

```
…
for (i=0;i<n-1;i++)
    for (j=0;j<(n-i) -1;j++)
        if (a[j] > a[j+1])
        {
                temp = a[j];
                a[j] = a[j+1];
                a[j+1]=temp;
        }
…
```

Two-Dimensional Arrays

- One common array application is to store data in a table, consisting of rows and columns.
- C represents such a table with a two-dimensional array.
- In a 2D array, we need two indexes to describe the address of a single element.

2-D Array

- The general form of declaration of 2-D Array is -

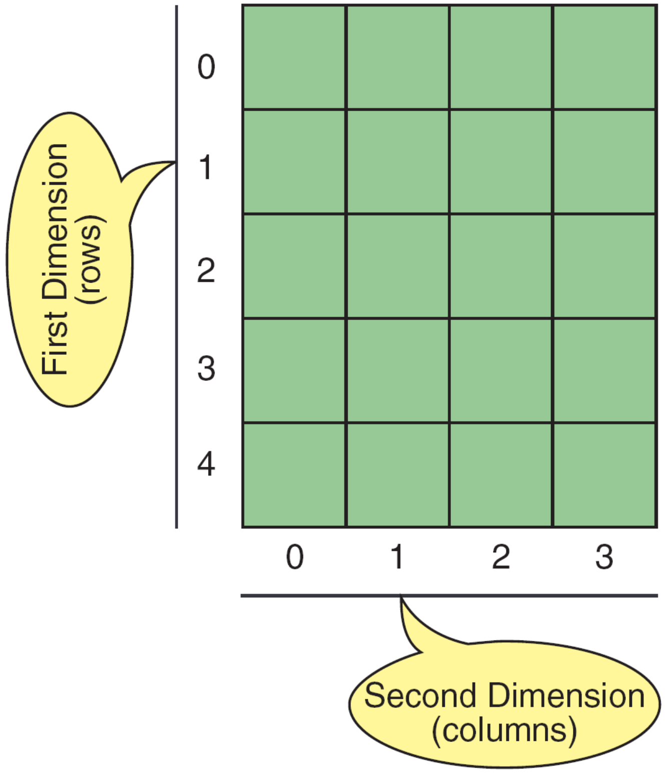
  type name [size1] [size2];

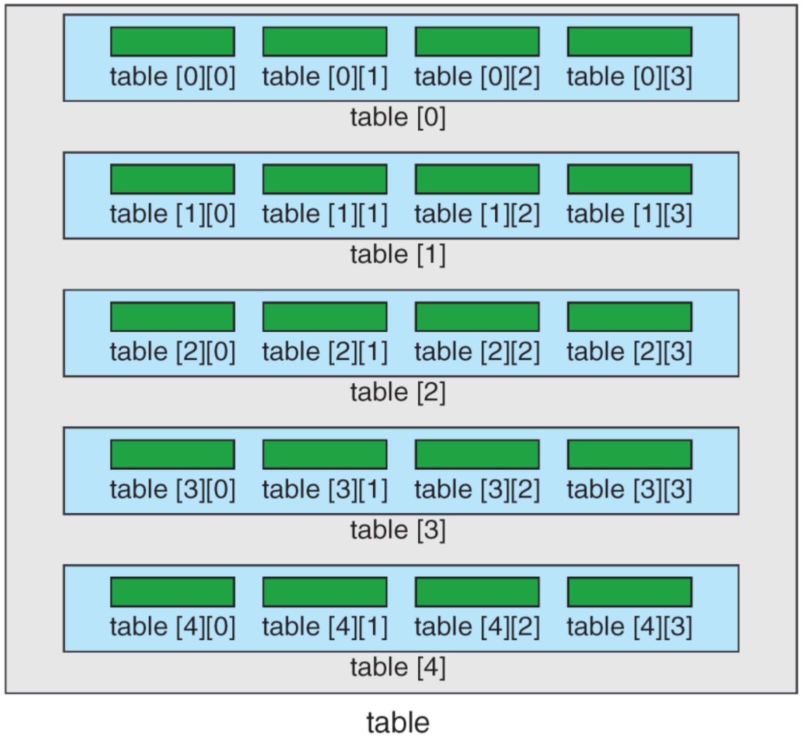
- Total size of the 2-D array

  = size1 * size2 * size of each variable.

- In 2-D array also the index begins with 0 for both parameters.
- Row value can have possible values 0 to size1 - 1 and columns can have possible values from 0 to size2 - 1.

Two Dimensional Array Concepts





2D Array Definition

- To define a two-dimensional array:
```
int table[5][4];
```

Two-Dimensional Initialization

- To initialize a 2D array:

```
int table[5][4] =
{
    {0,  1,  2,  3}
    {10, 11, 12, 13}
    {20, 21, 22, 23}
    {30, 31, 32, 33}
    {40, 41, 42, 43}
}
```

Inputting & Outputting 2D Values

- We must reference both a row number and a column number to input or output values in a two-dimensional array. We use a nested `for` loop for this (next slides) …

Inputting 2D Values

```
  for(row=0; row<5; row++)
  {
    for(col=0; col<4; col++)
    {
    scanf("%d",&table[row][col])
;
    }
  }
```

Outputting 2D Values

```
for(row=0; row<5; row++)
{
  for(col=0; col<4; col++)
  {
  printf(“%d”, table[row][col]);
  }
}
```

Accessing Individual Elements

- To access an individual element in a two-dimensional array, we need to specify two indices (row & column):

```
table[2][0] = 42;
table[0][1] = table[3][2] +
15;
```

Code Examples of Two Dimensional Arrays

- Examples:

Fill2DArray.c
Map2DArray.c

Avg2DArray.c

Exercise

- Consider 2 3x3 integer matrix. And do the following operations on it.
  - -input each elements of it.
  - -print each elements of it in matrix form.
  - -do the sum of elements.
  - -subtract first matrix from the second one.
  - -multiply both matrices.