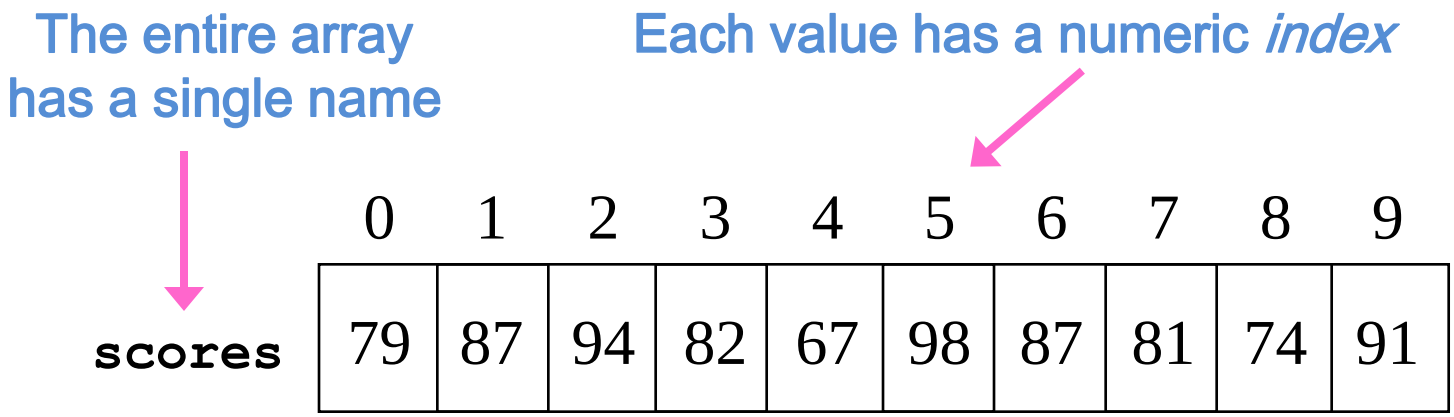
CC213 Programming Applications

Week #3

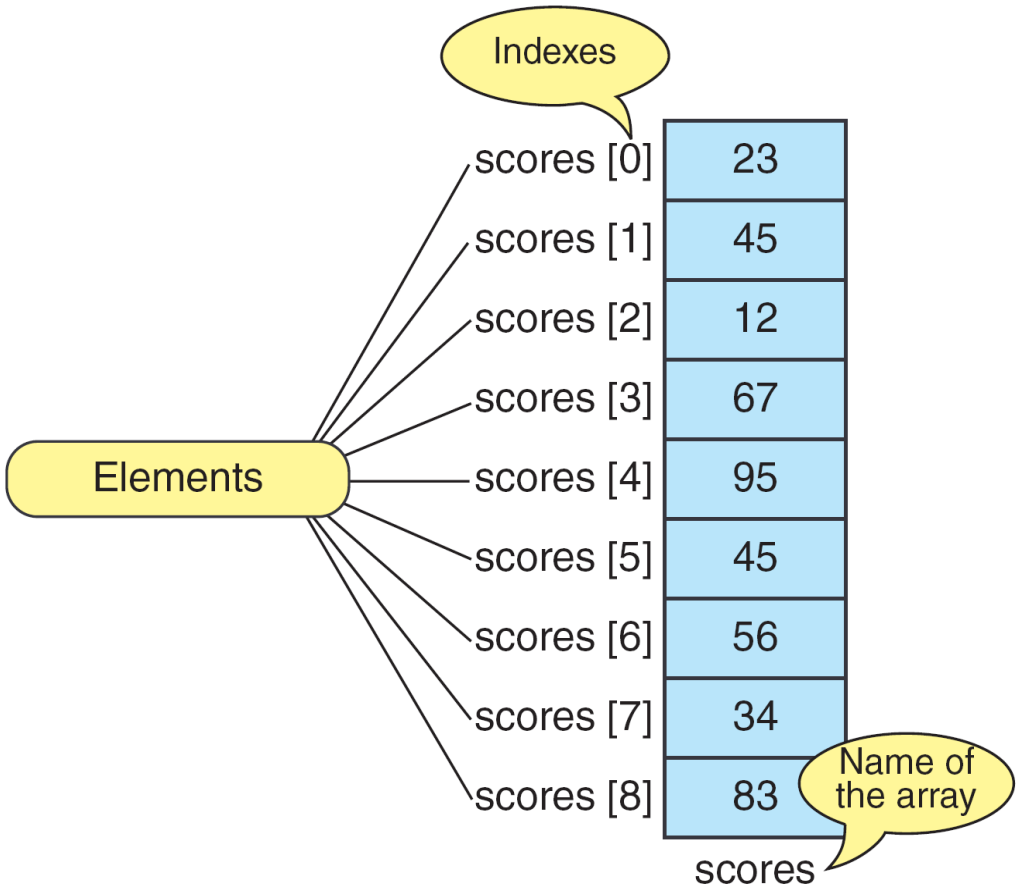
Arrays

- An *array* is an ordered list of values

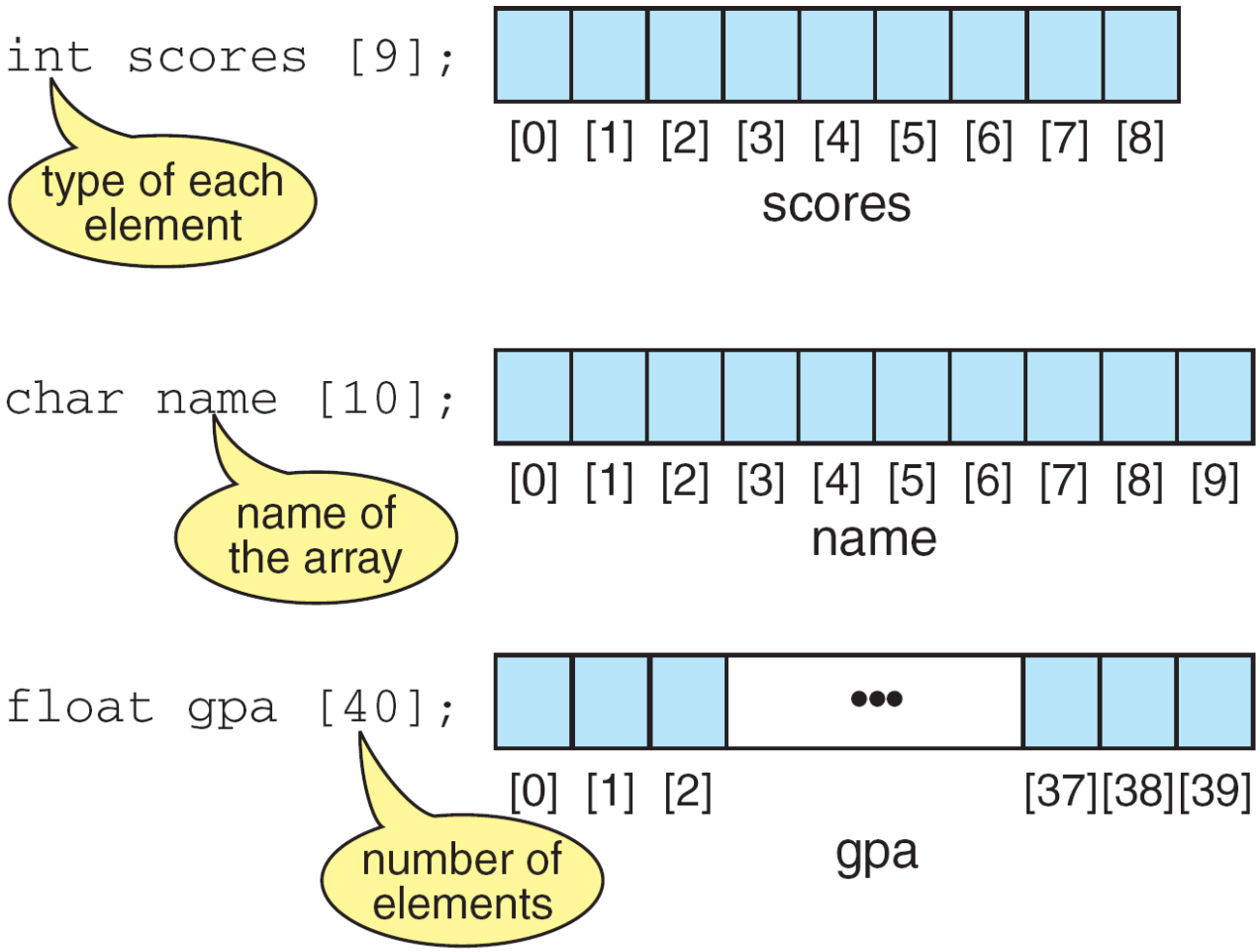


An array of size N is indexed from zero to N-1

This array holds 10 values that are indexed from 0 to 9



The Scores Array



Declaring and Defining Arrays

(a) Basic Initialization

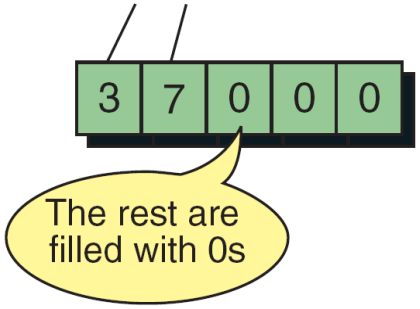
```
int numbers[5] = {3,7,12,24,45};
```

(b) Initialization without Size

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

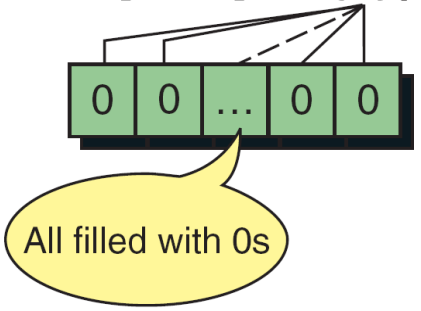
(c) Partial Initialization

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



(d) Initialization to All Zeros

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



Initializing Arrays

Note

One array cannot be copied to another using assignment.

Array Input/ Output

- We typically use for loops for any kind of array processing.
- To input an array, one by one:

```
for (i=0; i<10 ; i++ )
{
  printf(" Enter element %d  : ", i );
  scanf ( " %d ", &scores[i] );
}
```

Array Output

- To display an array, one element per line:

```
for (i=0; i<10 ; i++ )
{
  printf(" scores [%d] : %d\n", i ,
 scores[i] );
}
```

Example

- double x[ ]= {16.0, 12.0, 6.0, 8.0, 2.5, 12.0, 14.0, -54.5}
- int i=5;
- printf(“%f”, x[4]);
- printf(“%f”, x[i]+1);
- printf(“%f”, x[i+1]);
- printf(“%f”, x[2*i]);

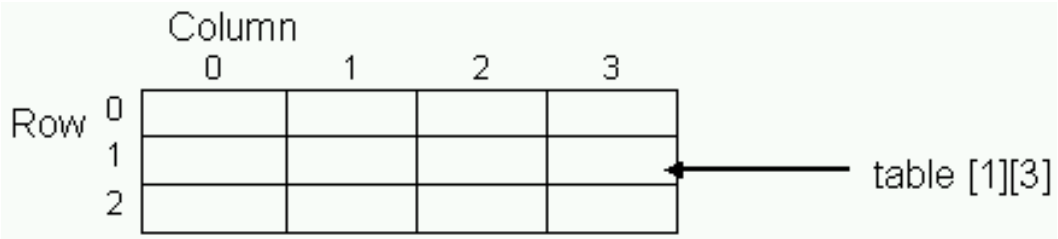
Example

- double x[ ]= {16.0, 12.0, 6.0, 8.0, 2.5, 12.0, 14.0, -54.5}
- int i=5;
- printf(“%f”, x[2*i-3]);
- printf(“%f”, x[(int) x[4]]);
- printf(“%f”, x[i++]);
- printf(“%f”, x[--i]);
- x[i]= x[i+1];

Write a program to get the average of 10 integers in an array.

Introduction to 2-D Arrays

- A 2-D array is a contiguous collection of variables of the same type, that may be viewed as a table consisting of rows and columns.



- The same reason that necessitated the use of 1-D arrays can be extended to 2-D and other multi-D Arrays.
- For example, to store the grades of 30 students, in 5 courses require multiple 1-D arrays.
- A 2-D array allows all these grades to be handled using a single variable.

Declaration of 2-D Arrays

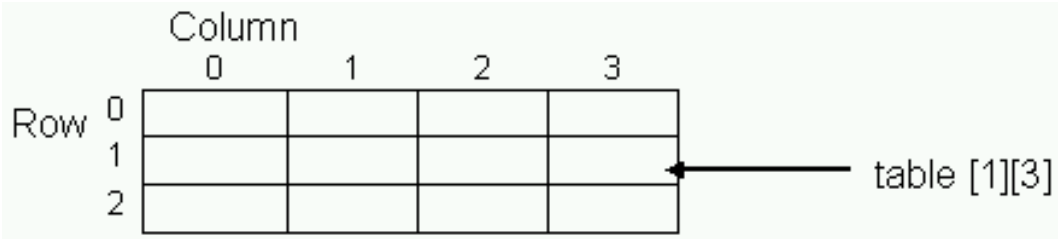
- A 2-D array variable is declared by specifying the type of elements, the name of the variable, followed by the number of rows and number of columns – each is a separate bracket:
- The following declares a 2-D array, *table*, having 3 rows and 4 columns.

```
int table[3][4];
```

- Both rows and columns are indexed from zero. So the three rows have indexes 0, 1 and 2 and four the columns have 0, 1, 2, 3.
- As we saw in 1-D array, it is a good practice to declare the sizes as constants. So a better declaration for the above is:

```
#define ROWS 3
#define COLS 4

int table[ROWS][COLS];
```



Accessing 2-D Array elements

- A particular element of a 2-D array, table, is referenced by specifying its row and column indexes:

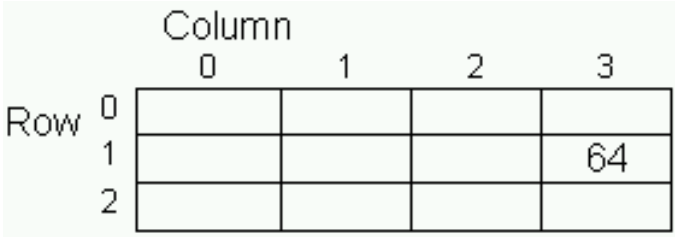
  table[RowIndex][columnIndex]

- For example, given the declaration:

  int table[3][4];

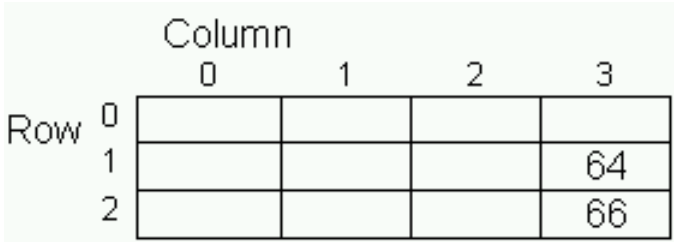
- The following stores 64 in the cell with row index 1, column index 3.

  table[1][3] = 64;



- We use the same format to refer to an element in an expression:

  table[2][3] = table[1][3] + 2;



Scanning 2D Arrays

- To use 2-dimensional array we use Two nested for-loop as follows:

```
for(i=0; i<3;i++)
  for(j=0; j<3; j++)
    A[i][j]=i*j;
```

Example
Addition of two matrices

- Write a program that reads two matrices of size 5x5, and adds the two matrices to another one which will be displayed.