SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF AIML

23ITT101-PROGRAMMING IN C AND DATA STRUCTURES

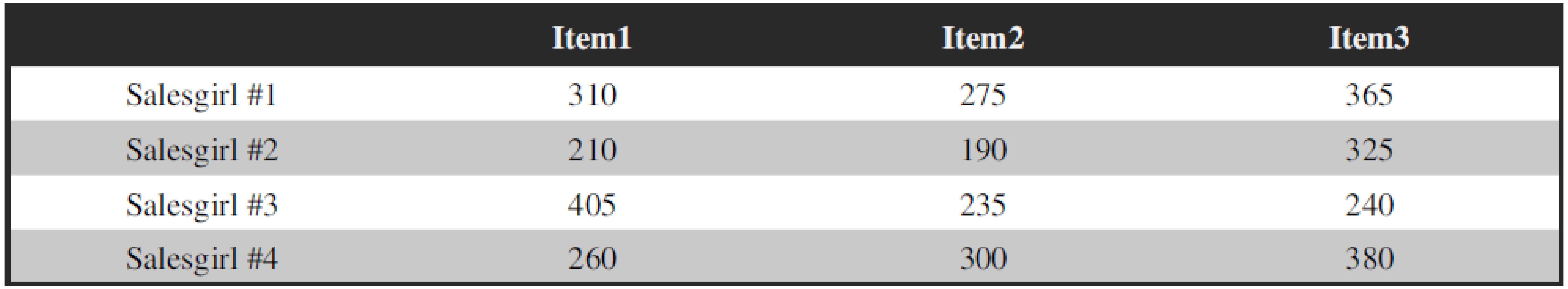
I YEAR - II SEM

UNIT 3 – ARRAYS AND INTRODUCTION TO DATA STRUCTURES

TOPIC 3 – Two – Dimensional Arrays

TWO-DIMENSIONAL ARRAYS

- So far we have discussed the array variables that can store a list of values.
- There could be situations where a table of values will have to be stored



	Item1	Item2	Item3
Salesgirl #1	310	275	365
Salesgirl #2	210	190	325
Salesgirl #3	405	235	240
Salesgirl #4	260	300	380

- Consider the following data table, which shows the value of sales of three items by four sales girls:
- The table contains a total of 12 values, three in each line.
- We can think of this table as a matrix consisting of **four rows** and **three columns**.
- Each row represents the values of sales by a particular salesgirl
- Each column represents the values of sales of a particular item.
- In mathematics, we represent a particular value in a matrix by using two subscripts such as vij.
- Here **v** denotes the entire matrix and vij refers to the value in the **ith** row and **jth** column.
- For example, in the above table v23 refers to the value 325.

DECLARATION OF TWO-DIMENSIONAL ARRAYS

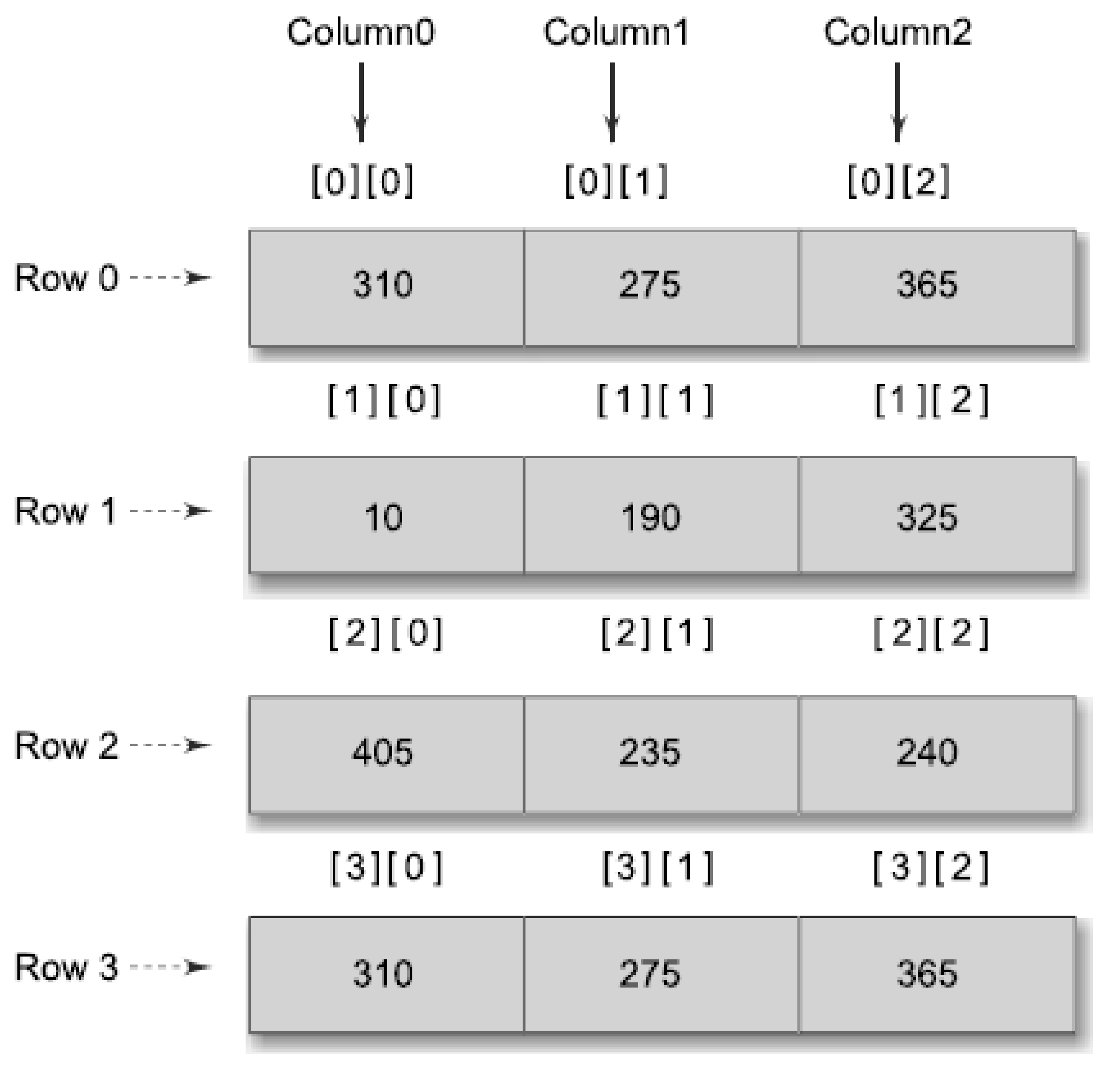
- C allows us to define such tables of items by using two-dimensional arrays.
- The table discussed above can be defined in C as

  v[4][3]

- Two-dimensional arrays are declared as follows:

  type array_name [row_size][column_size];

- Note that unlike most other languages, which use one pair of parentheses with commas to separate array sizes, C places each size in its own set of brackets.



Representation of a two-dimensional array in memory

INITIALIZING TWO-DIMENSIONAL ARRAYS

- As Like the one-dimensional arrays, two-dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in braces.
- For example, `int table[2][3] = { 0,0,0,1,1,1};`
- initializes the elements of the first row to zero and the second row to one.
- The initialization is done row by row.
- The above statement can be equivalently written as `int table[2][3] = {{0,0,0}, {1,1,1}};`
- by surrounding the elements of the each row by braces.
- We can also initialize a two-dimensional array in the form of a matrix as shown below:

```
int table[2][3] = {
                        {0,0,0},
                        {1,1,1}
                  };
```

- Commas are required after each brace that closes off a row, except in the case of the last row.

INITIALIZING TWO-DIMENSIONAL ARRAYS

- When the array is completely initialized with all values, explicitly, we need not specify the size of the first dimension.
- That is, the statement

```
int table [ ] [3] = {
                         { 0, 0, 0},
                         { 1, 1, 1}
                    };
```

- is permitted.
- If the values are missing in an initializer, they are automatically set to zero.
- For instance, the statement

```
int table[2][3] = {
                      {1,1},
                      {2}
                  };
```

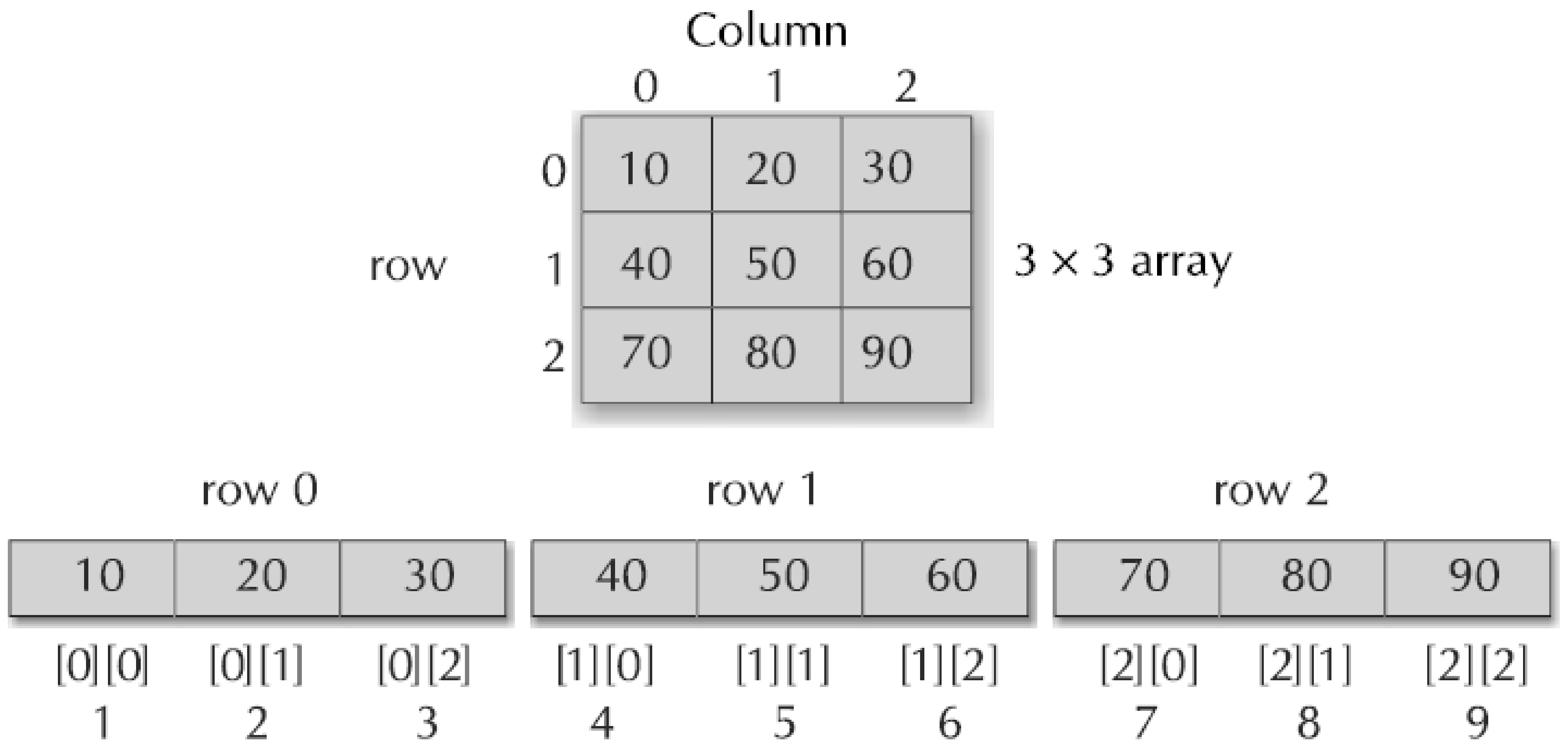
- will initialize the first two elements of the first row to one, the first element of the second row to two, and all other elements to zero.
- When all the elements are to be initialized to zero, the following short-cut method may be used.

```
int m[3][5] = { {0}, {0}, {0}};
```

- The first element of each row is explicitly initialized to zero while other elements are automatically initialized to zero.

MEMORY LAYOUT

- The subscripts in the definition of a two-dimensional array represent rows and columns.
- This format maps the way that data elements are laid out in the memory



Memory Layout

MULTI-DIMENSIONAL ARRAYS

- C allows arrays of **three** or more dimensions.
- The exact limit is determined by the compiler.
- The general form of a multi-dimensional array is

```
type array_name[s1][s2][s3]....[sm];
```

- where si is the size of the ith dimension.
- Some examples are:

```
int survey[3][5][12];
float table[5][4][5][3];
```

- **survey** is a three-dimensional array declared to contain 180 integer type elements.
- Similarly **table** is a four dimensional array containing 300 elements of floating-point type.

DYNAMIC ARRAYS

- **Static Arrays:**
- So far, we have created arrays at compile time.
- An array created at compile time by specifying size in the source code has a fixed size and cannot be modified at run time.
- The process of allocating memory at compile time is known as **static memory allocation**
- The arrays that receive static memory allocation are called **static arrays**.
- This approach works fine as long as we know exactly what our data requirements are.
- Consider a situation where we want to use an array that can vary greatly in size.
- We must guess what will be the largest size ever needed and create the array accordingly.

- **Dynamic Arrays:**
- In C it is possible to allocate memory to arrays at run time.
- This feature is known as **dynamic memory allocation** and the arrays created at run time are called **dynamic arrays**.
- Dynamic arrays are created using what are known as pointer variables and memory management functions malloc, calloc and realloc.
- These functions are included in the header file <stdlib.h>.