CS118 – Programming Fundamentals

Lecture # 24
Tuesday, November 19, 2019
FALL 2019
FAST – NUCES, Faisalabad Campus

Zain Iqbal

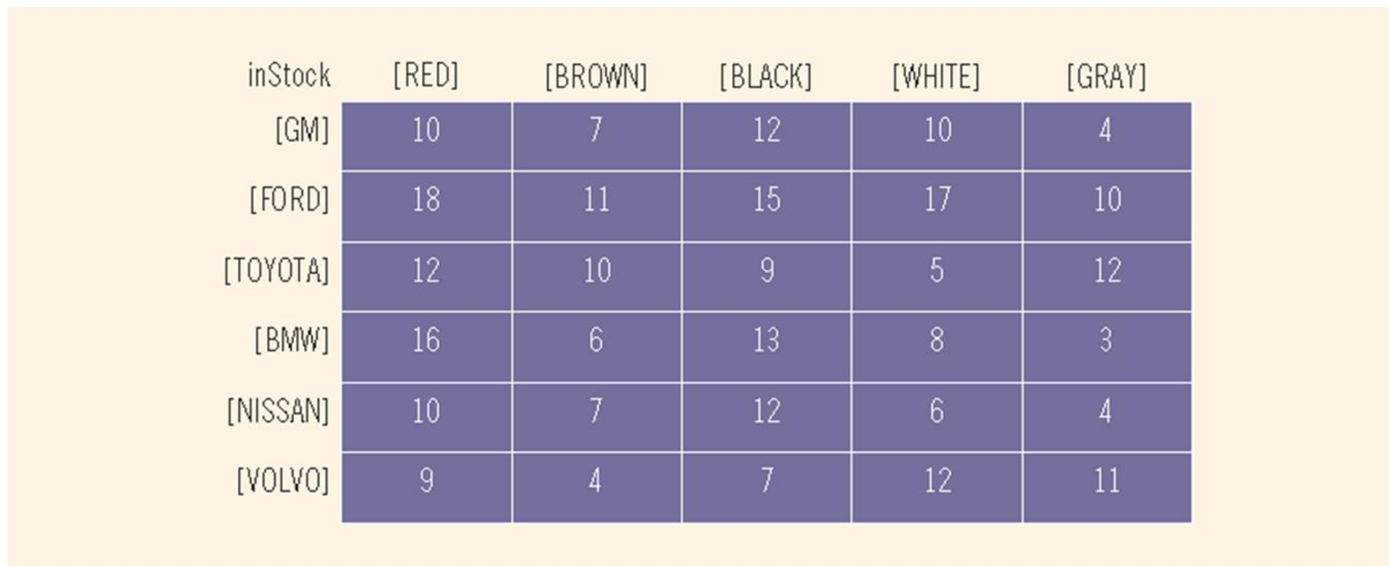
Two- and Multidimensional Arrays

- **Two-dimensional array:** collection of a fixed number of components (of the same type) arranged in two dimensions
- Sometimes called matrices or tables
- Declaration syntax:

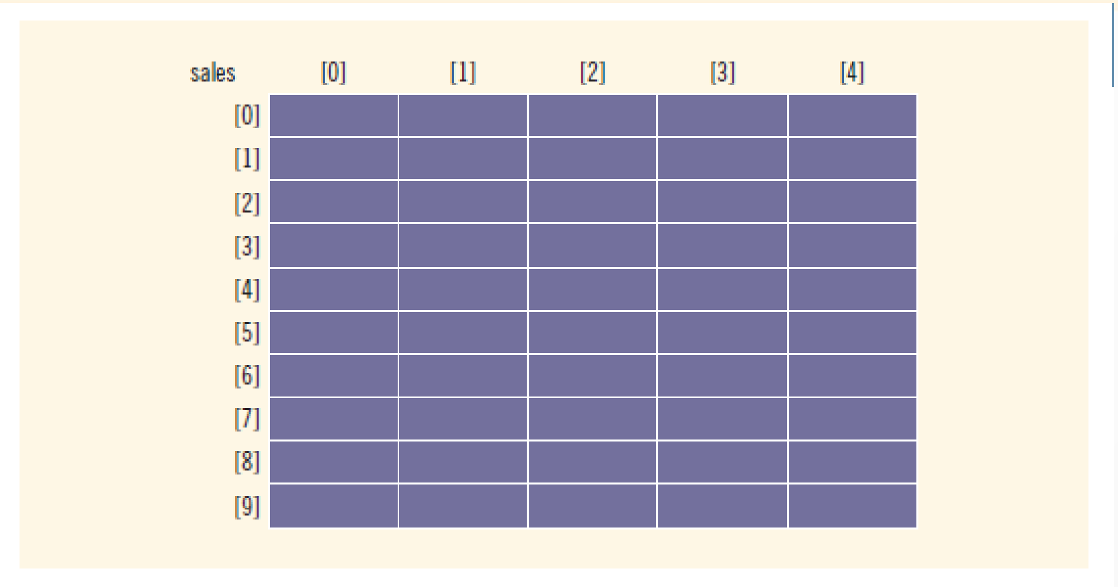
```
dataType  arrayName[intExp1][intExp2];
```

where **intexp1** and **intexp2** are expressions yielding positive integer values, and specify the **number of rows** and the **number of columns**, respectively, in the array

Two- and Multidimensional Arrays (cont'd.)



inStock	[RED]	[BROWN]	[BLACK]	[WHITE]	[GRAY]
[GM]	10	7	12	10	4
[FORD]	18	11	15	17	10
[TOYOTA]	12	10	9	5	12
[BMW]	16	6	13	8	3
[NISSAN]	10	7	12	6	4
[VOLVO]	9	4	7	12	11

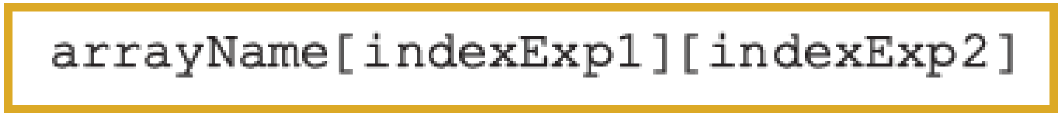


sales	[0]	[1]	[2]	[3]	[4]
[0]					
[1]					
[2]					
[3]					
[4]					
[5]					
[6]					
[7]					
[8]					
[9]					

FIGURE 9-10 Two-dimensional array `sales`

Accessing Array Components

- Syntax:



```
arrayName[indexExp1][indexExp2]
```

where indexexp1 and indexexp2 are expressions yielding nonnegative integer values, and specify the row and column position

Accessing Array Components (c

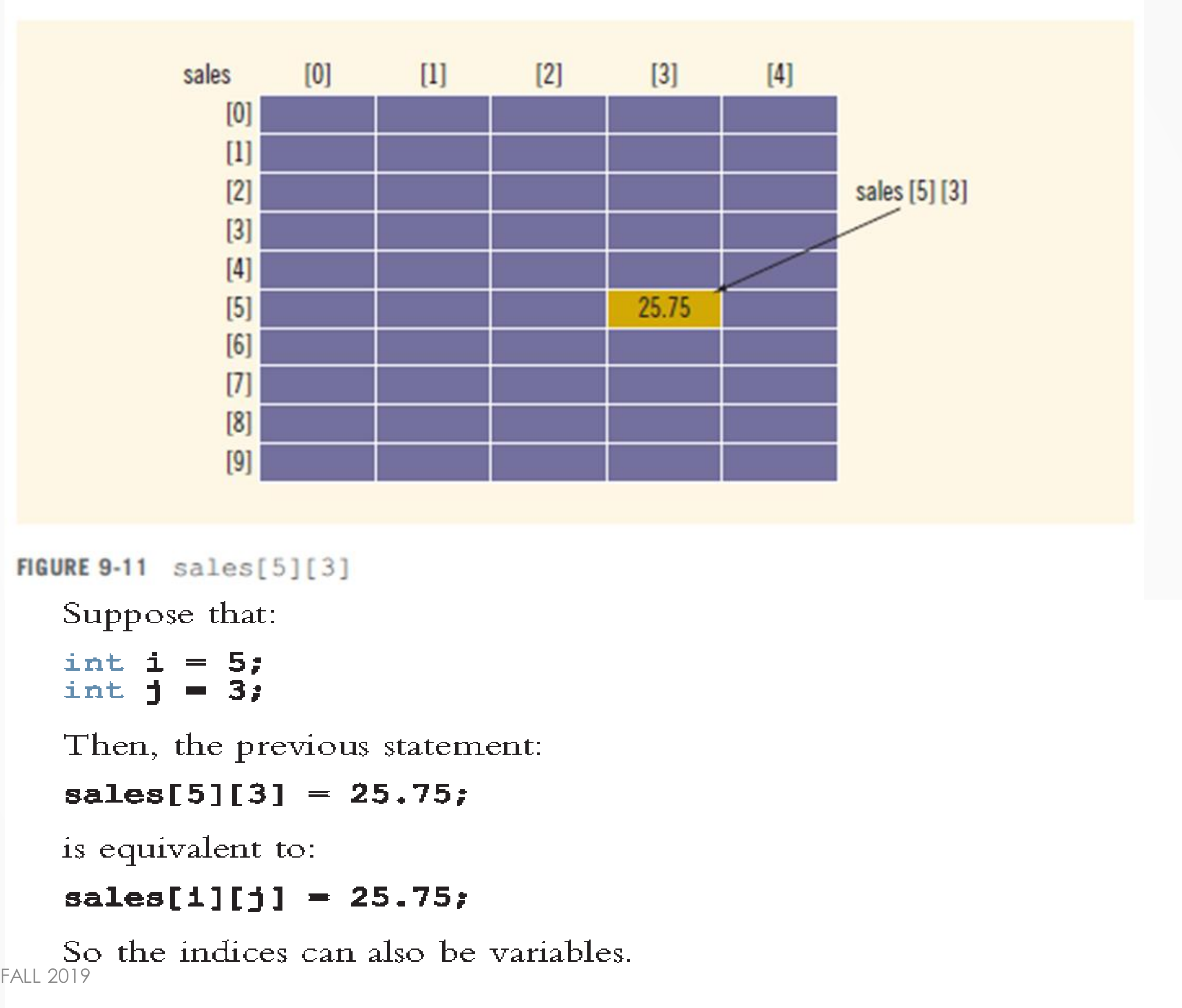

sales
[0]
[1]
[2]
[3]
[4]
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
25.75
sales [5] [3]

FIGURE 9-11 sales[5][3]

Suppose that:

```
int i = 5;
int j = 3;
```

Then, the previous statement:

```
sales[5][3] = 25.75;
```

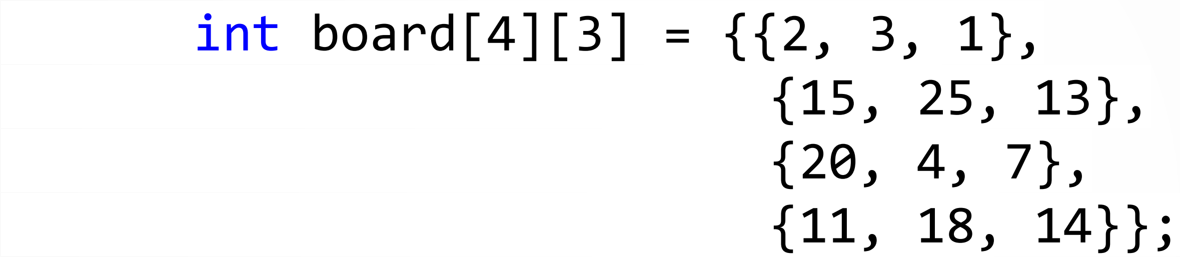
is equivalent to:

```
sales[i][j] = 25.75;
```

So the indices can also be variables.

Two-Dimensional Array Initialization During Declaration

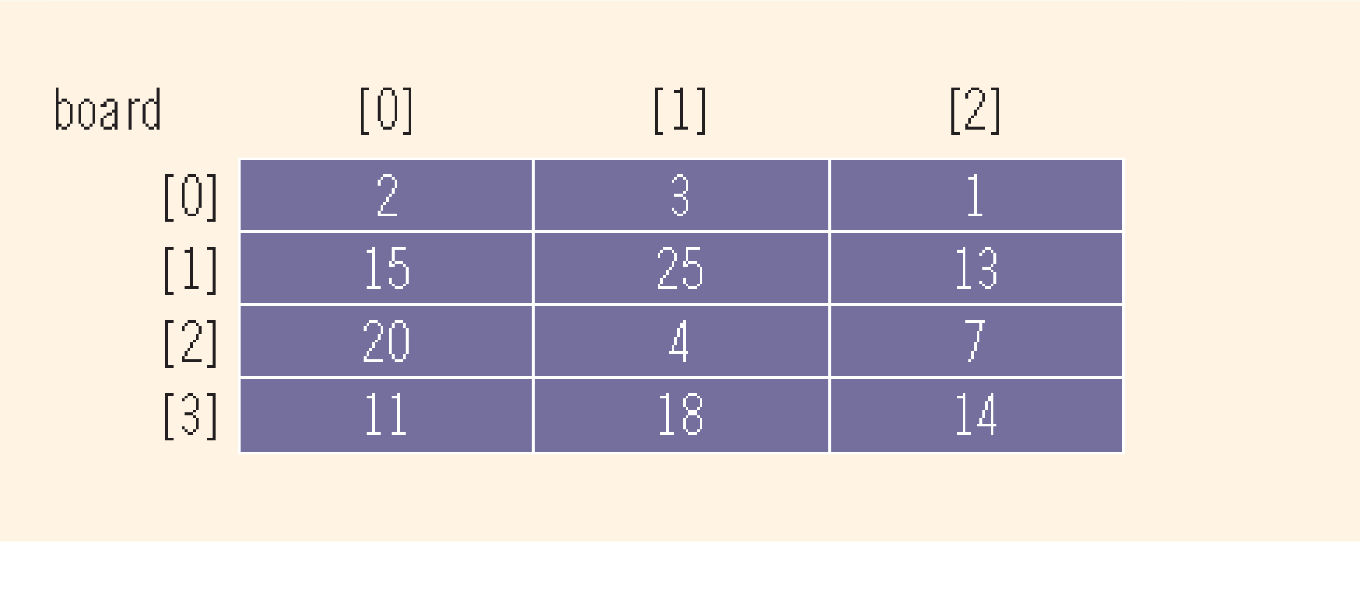
- Two-dimensional arrays can be initialized when they are declared:



```
int board[4][3] = {{2, 3, 1},
                   {15, 25, 13},
                   {20, 4, 7},
                   {11, 18, 14}};
```

- Elements of each row are enclosed within braces and separated by commas
- All rows are enclosed within braces
- For number arrays, if all components of a row aren’t specified, unspecified ones are set to 0

Example



board	[0]	[1]	[2]
[0]	2	3	1
[1]	15	25	13
[2]	20	4	7
[3]	11	18	14

Processing Two-Dimensional Arrays

- Ways to process a two-dimensional array:
  - Process the entire array
  - Process a particular row of the array, called row processing
  - Process a particular column of the array, called column processing
- Each row and each column of a two-dimensional array is a one-dimensional array
  - To process, use algorithms similar to processing one-dimensional arrays

Processing Two-Dimensional Arrays (cont'd.)

```
const int NUMBER_OF_ROWS = 7;//This can be set to any number
const int NUMBER_OF_COLUMNS = 6;//This can be set to any number

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

Figure 9-15 shows Two-dimensional array matrix

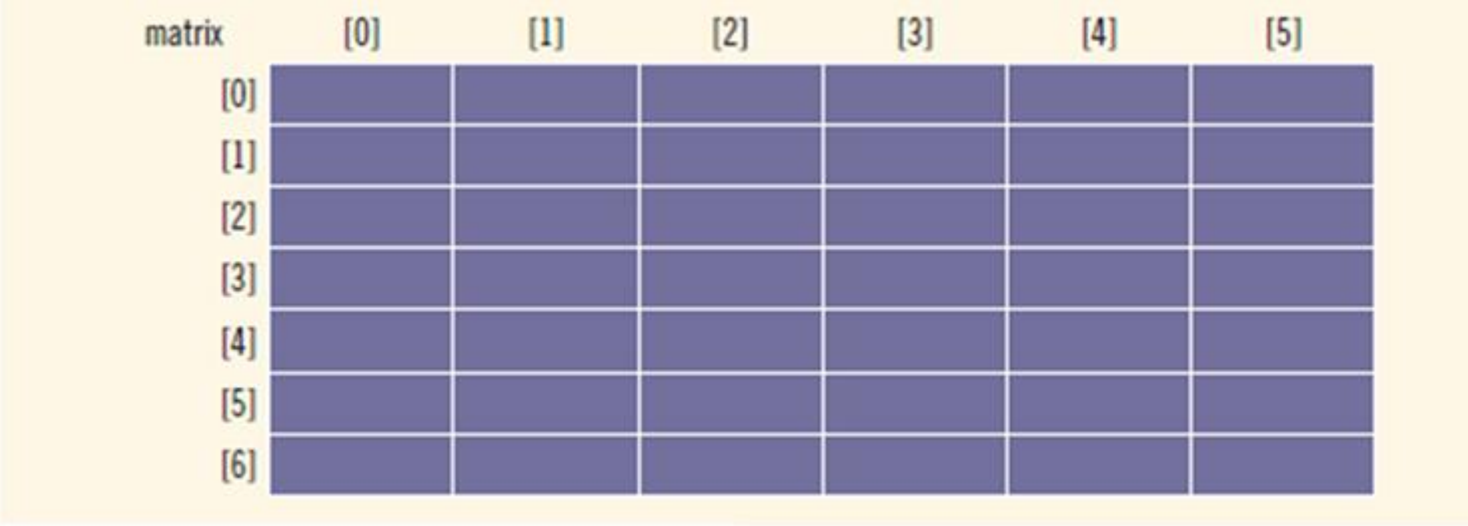


FIGURE 9-15 Two-dimensional array `matrix`

Initialization

```
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

- To initialize row number 5 (i.e., sixth row) to 0:

```
row = 5;
    for(int col = 0 ; col < NUMBER_OF_COLUMNS ; col++)
        matrix[row][col] = 0;
```

- To initialize the entire matrix to 0:

```
for(row = 0 ; row < NUMBER_OF_ROWS ; row++)
    for(col = 0 ; col < NUMBER_OF_COLUMNS ; col++)
        matric[row][col] = 0 ;
```

Print

```
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

- To input data into each component of matrix:

```
    for(row = 0 ; row < NUMBER_OF_ROWS ; row++)
    {
        for(col = 0 ; col < NUMBER_OF_COLUMNS ; col++)
            cout << setw(5) << matrix[row][col] << “ ” ;
        cout << endl;
    }
```

Input

```
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

- To input data into each component of matrix:

```
    for(row = 0 ; row < NUMBER_OF_ROWS ; row++)
        for(col = 0 ; col < NUMBER_OF_COLUMNS ; col++)
            cin >> matrix[row][col] ;
```

Sum by Row

```
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

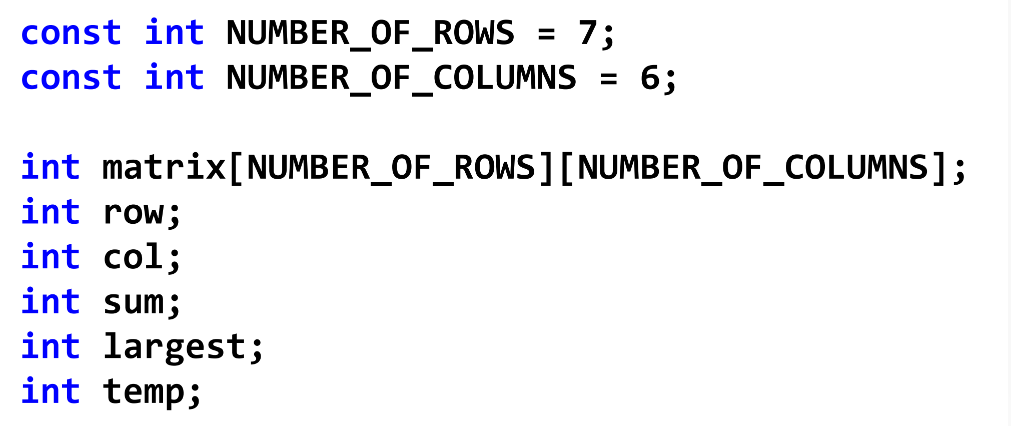
- To find the sum of row number 3 of matrix:

```
    sum = 0 ;
    row = 3 ;
    for(col = 0 ; col < NUMBER_OF_COLUMNS ; col++)
            sum += matrix[row][col] ;
```

- To find the sum of each individual row:

```
    //Sum of each individual row

    for (row = 0; row < NUMBER_OF_ROWS; row++)
    {
        sum = 0;
        for (col = 0; col < NUMBER_OF_COLUMNSS; col++)
            sum += matrix[row][col];
        cout << “Sum of Row ” << row + 1
            << “ = ” << sum << endl;
    }
```

Sum by Column

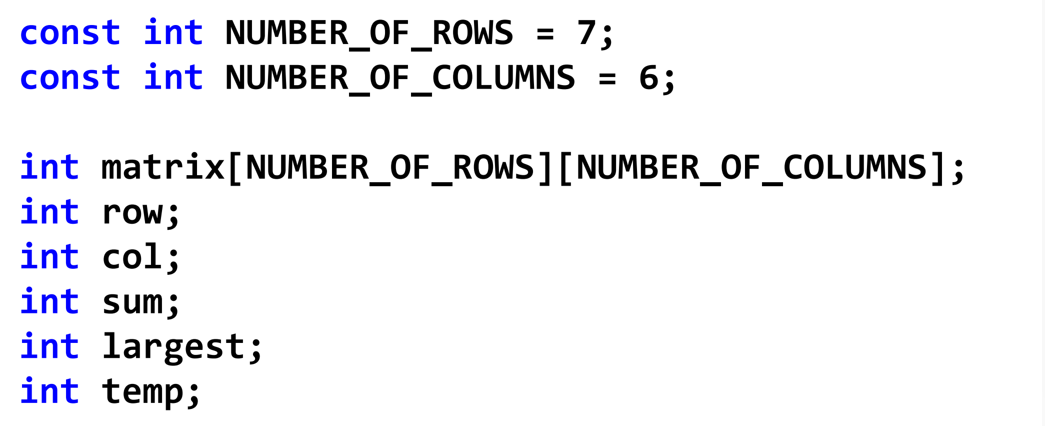


```
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

- To find the sum of each individual column:

```
        //Sum of each individual column
        for (col = 0 ; col < NUMBER_OF_COLUMNS ; col++)
        {
            sum = 0 ;
            for(row = 0 ;  row < NUMBER_OF_ROWS ; row++)
                sum += matrix[row][col] ;

            cout << “Sum of Column ” << col + 1
                << “ = ” << sum << endl ;
        }
```

Sum of Matrix



```
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

- To find the sum of complete matrix:

```
    //Sum of complete matrix
        sum = 0 ;
        for (row = 0 ; row < NUMBER_OF_ROWS ; row++)
        {
            for(col = 0 ;  col < NUMBER_OF_COLUMNS ; col++)
                sum += matrix[row][col] ;
        }
        cout << “Sum of Matrix = ” << sum << endl ;
```

Largest Element in Each Row

```
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

```
//Largest number in each row
for (row = 0 ; row < NUMBER_OF_ROWS ; row++)
{
       largest = matrix[row][0] ;
       for(col = 1 ;  col < NUMBER_OF_COLUMNS ; col++)
           if(matrix[row][col] > largest)
               largest = matrix[row][col];
       cout << “The largest element in row ” << row + 1
       << “ = ” << largest << endl;
}
```

Largest Element in Each Column

```
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

```
//Largest number in each col
for (col = 0 ; col < NUMBER_OF_COLS ; col++)
{
    largest = matrix[0][col] ;
    for(row = 1 ; row < NUMBER_OF_ROWS ; row++)
        if(matrix[row][col] > largest)
            largest = matrix[row][col];
    cout << “The largest element in col ” << col + 1
    << “ = ” << largest << endl;
}
```

Largest Element in Matrix

```
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

```
//Largest number in the matrix
largest = matrix[0][0] ;
for (row = 0 ; row < NUMBER_OF_ROWS ; row++)       {
    for(col = 0 ;  col < NUMBER_OF_COLUMNS ; col++)
        if(matrix[row][col] > largest)
            largest = matrix[row][col] ;
}
cout << “The largest element in matrix = ”
    << largest << endl;
```

Passing Two-Dimensional Arrays as Parameters to Functions

- Two-dimensional arrays can be passed as parameters to a function
  - Pass by reference
    - Base address (address of first component of the actual parameter) is passed to formal parameter
- Two-dimensional arrays are stored in row order
- When declaring a two-dimensional array as a formal parameter, can omit size of first dimension, but not the second

Example

Suppose we have following declaration:

```
const int NUMBER_OF_ROWS = 6;
const int NUMBER_OF_COLUMNS = 5;
```

Consider the following definition of function printMatrix:

```
void printMatrix(int matrix[][NUMBER_OF_COLUMNS], int 
noOfRows)
{
    int row = 0, col = 0;
    for( row = 0 ; row < noOfRows ; row++)
    {
        for(col = 0 ; col < NUMBER_OF_COLUMNS ; col++)
            cout << setw(5) << matrix[row][col] << “ ” ;
        cout << endl ;
    }
}
```

Function outputs the sum of the elements of each row

```
void sumRows(int matrix[][NUMBER_OF_COLUMNS], int noOfRows)
{
    int row, col, sum = 0 ;
    for( row = 0 ; row < noOfRows ; row++)
    {
        sum = 0;
        for(col = 0 ; col < NUMBER_OF_COLUMNS ; col++)
            sum = sum + matrix[row][col] << “ ” ;
        cout << “Sum of row ” << row + 1
            << “ = ” << sum << endl ;
    }
}
```

Function determines the largest element in each row:

```
void largestInRows(int matrix[][NUMBER_OF_COLUMNS],
int noOfRows)
{
    int row, col, sum = 0 ;
    //Largest element in each row
    for( row = 0 ; row < noOfRows ; row++)
    {
        largest = matrix[row][0];
        for(col = 0 ; col < NUMBER_OF_COLUMNS ; col++)
            if(matrix[row][col] > largest)
                largest =  matrix[row][col];

        cout << “The Largest element of row ”
            << row + 1 << “ = ” << largest << endl ;
    }
}
```

Array Arithmetic

- If base address of the array is known the address of any index in the two dimensional array can be calculated.
- Address of arr[row][col] provided base address of the array is ‘b’ and size of data type is ‘s’ and columns per row are COLS
- Address of arr[row][col] = b + (row * COLS + col)* s
- Or arr[row][col] = b + (row * COLS * s + col * s

e.g. for int Arr[5][6]; provided base address is 100

Address of Arr[1][2] = 100 + (1*6 + 2) * 4 = 100 + 32 = 132

Address of Arr[0][0] = 100 + (0*6 + 0) * 4 = 100 + 0 = 100

Address of Arr[4][0] = 100 + (4*6 + 0) * 4 = 100 + 96 = 196

Arrays of Strings

- Strings in C++ can be manipulated using either the data type string or character arrays (C-strings)
- On some compilers, the data type **string** may not be available in Standard C++ (i.e., non-ANSI/ISO Standard C++)

Arrays of Strings and the string Type

- To declare an array of 100 components of type string:

```
string list[100];
```

- Basic operations, such as assignment, comparison, and input/output, can be performed on values of the **string** type
- The data in **list** can be processed just like any one-dimensional array

Arrays of Strings and C-Strings (Character Arrays)

- Consider declaration

  char list[100][16];

- Now list[j] for each j, 0 <= j <= 99, is a string of at-most 15 characters in length

```
strcpy(list[1], "Snow White");
```

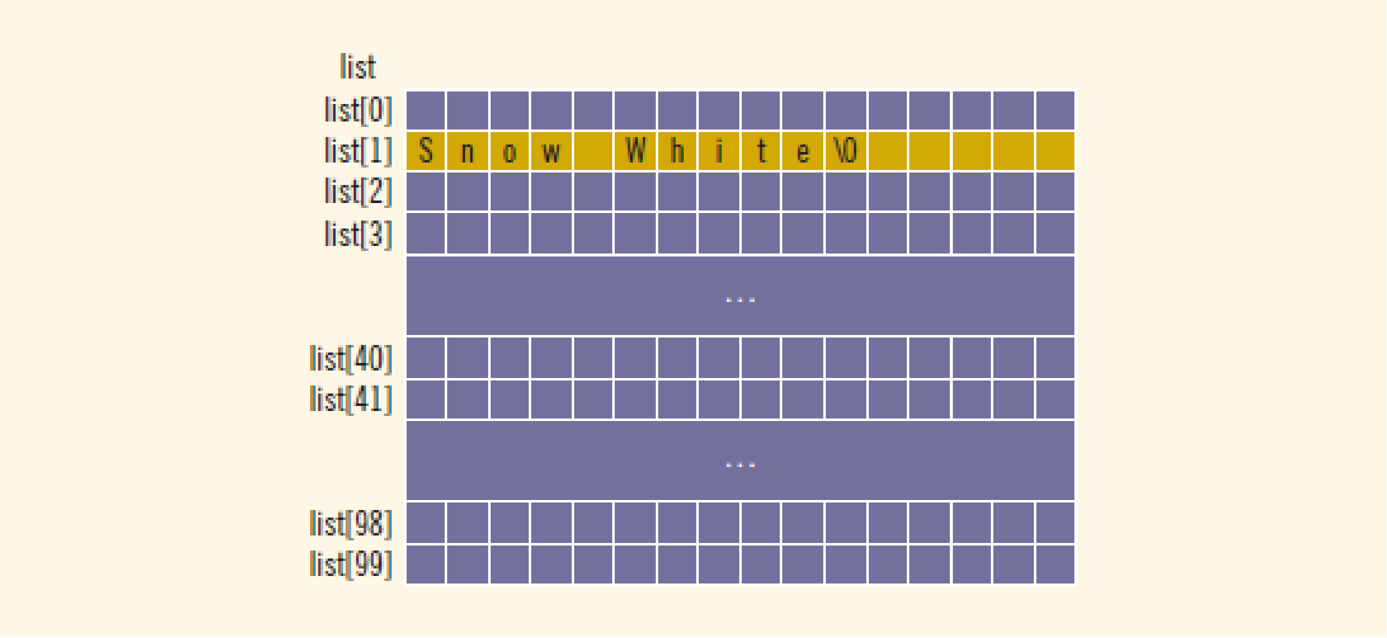


FIGURE 9-17 Array list, showing list[1]

Contd..

Suppose that you want to read and store data in a list and there is one entry per line.

The following code accomplishes this:

```
    char list[100][16];
    for (int i = 0; i < 100; i++)
    {
        cin.get(list[i], 16);
        cin.ignore();
    }
```

The following for loop outputs the string in each row:

```
    for (int i = 0; i < 100; i++)
        cout << list[i] << endl;
```

You can also use other string functions (such as **strcmp** and **strlen**) and for loops to manipulate list

Questions