Programming Fundamentals

Aamina Batool

Output Formatting

```
#include <iomanip>
```

- **`setprecision(n)`**
  - where **`n`** is the number of decimal places.
  - **`cout << setprecision(2);`**
  - formats the output of decimal numbers to two decimal places

- **`setw(n)`**
  - The manipulator **`setw`** is used to output the value of an expression in a specific number of columns.

setw(n)

```
int miles = 245;
int speed = 55;
double hours = 35.45;
double error = 3.7564;
cout << setw(5) << miles << setw(5) << speed
     << setw(6) << hours << setw(7) << error;
```

OUTPUT:

```
12345678901234567890123456789
  245   55 35.45   3.76
```

How to generate a random number?

#include <cstdlib>
#include <ctime>

- **`rand()`** returns an **`int`** value between **`0`** and **`32767`**
- The statement:
  - **`cout << rand() << ", " << rand() << endl;`**
- will output two numbers that appear to be random.
- However, each time the program is run, this statement will output the same random numbers.
- This is because the function **`rand`** uses an algorithm that produces the same sequence of random numbers each time the program is executed on the same system.

Random numbers

#include <cstdlib>
#include <ctime>

- To generate different random numbers each time the program is executed, you also use the function **`srand(n)`** `where n is an integer.`
- By specifying different seed values, each time the program is executed, the function **`rand`** will generate a different sequence of random numbers.
- How to specify different seed value every time the program executes?
  - You can do it manually
  - You can use **`time(0)`** `(from ctime header file)`
- **time(0)** returns the number of seconds elapsed since January 1, 1970. So, its value is changing every second.

Random numbers between a range

- `srand(time(0));`
- `num = rand() % 100;`
- The first statement sets the seed, and the second statement generates a random number greater than or equal to **0** and less than **100**.
- How to generate numbers between 10 and 100?

Two-Dimensional Arrays

- Two-dimensional Array: a collection of a fixed number of components arranged in two dimensions
  - All components are of the same type

- The syntax for declaring a two-dimensional array is:

```
dataType arrayName[intexp1][intexp2];
```

where `intexp1` and `intexp2` are expressions yielding positive integer values

Two-Dimensional Arrays (continued)

- The two expressions `intexp1` and `intexp2` specify the number of rows and the number of columns, respectively, in the array
- Two-dimensional arrays are sometimes called matrices or tables
- Array of arrays

Declaring a 2-D array

```
double sales[10][5];
```

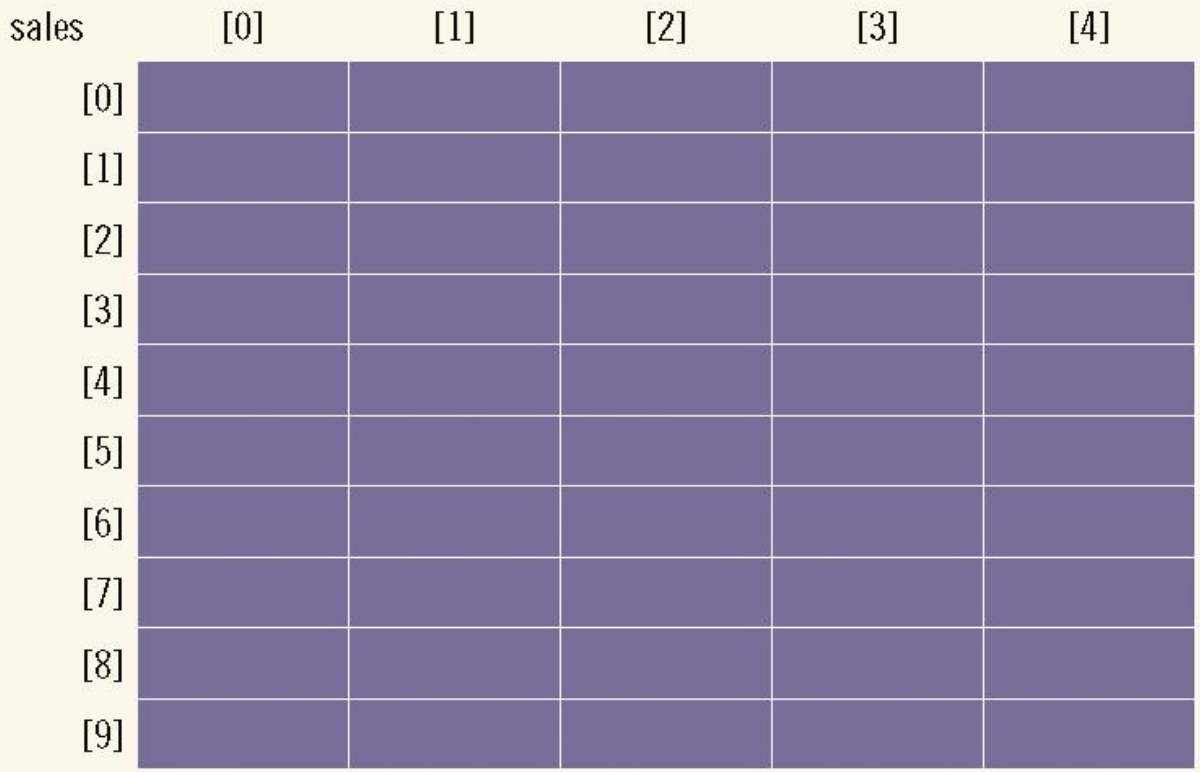


FIGURE 9-8 Two-dimensional array `sales`

Accessing Array Components

- The syntax to access a component of a two-dimensional array is:

  ```
  arrayName[indexexp1][indexexp2]
  ```

  where `indexexp1` and `indexexp2` are expressions yielding nonnegative integer values
- `indexexp1` specifies the row position and `indexexp2` specifies the column position

Accessing and initializing one element of a 2-D array

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

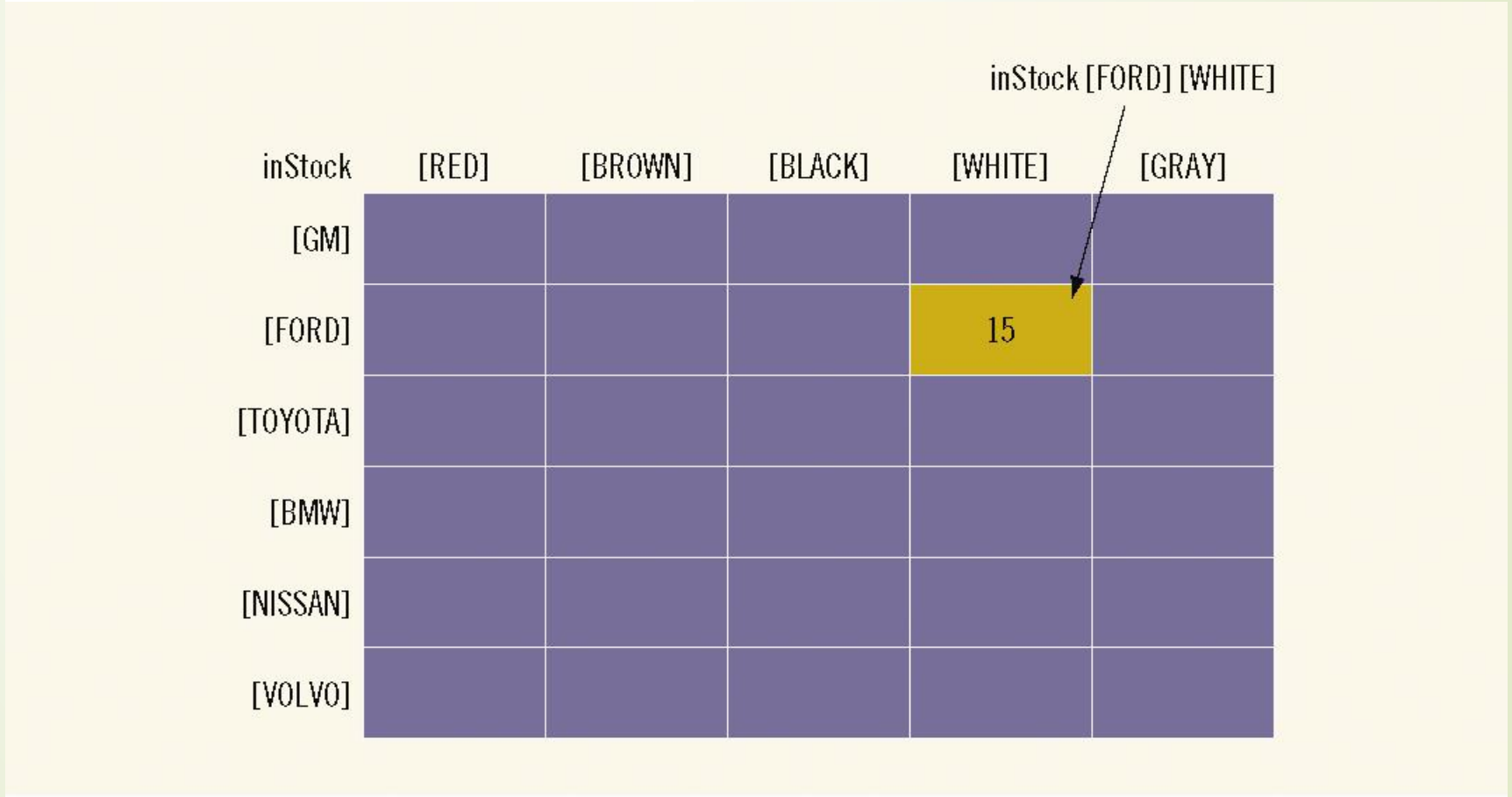


FIGURE 9-12 inStock[FORD][WHITE]

Initialization of complete 2-D array

- Like one-dimensional arrays
  - Two-dimensional arrays can be initialized when they are declared
- To initialize a two-dimensional array when it is declared
  1. Elements of each row are enclosed within curly braces and separated by commas
  2. The set of all rows is enclosed within curly braces
  3. For number arrays, if all components of a row are not specified, the unspecified components are initialized to zero

Declaration & initialization

- int x[3][4];

- Initialization:
- int test[2][3] = {2, 4, -5, 9, 0, 9};
- int test[2][3] = { {2, 4, 5}, {9, 0, 0}};

Initialization during declaration

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

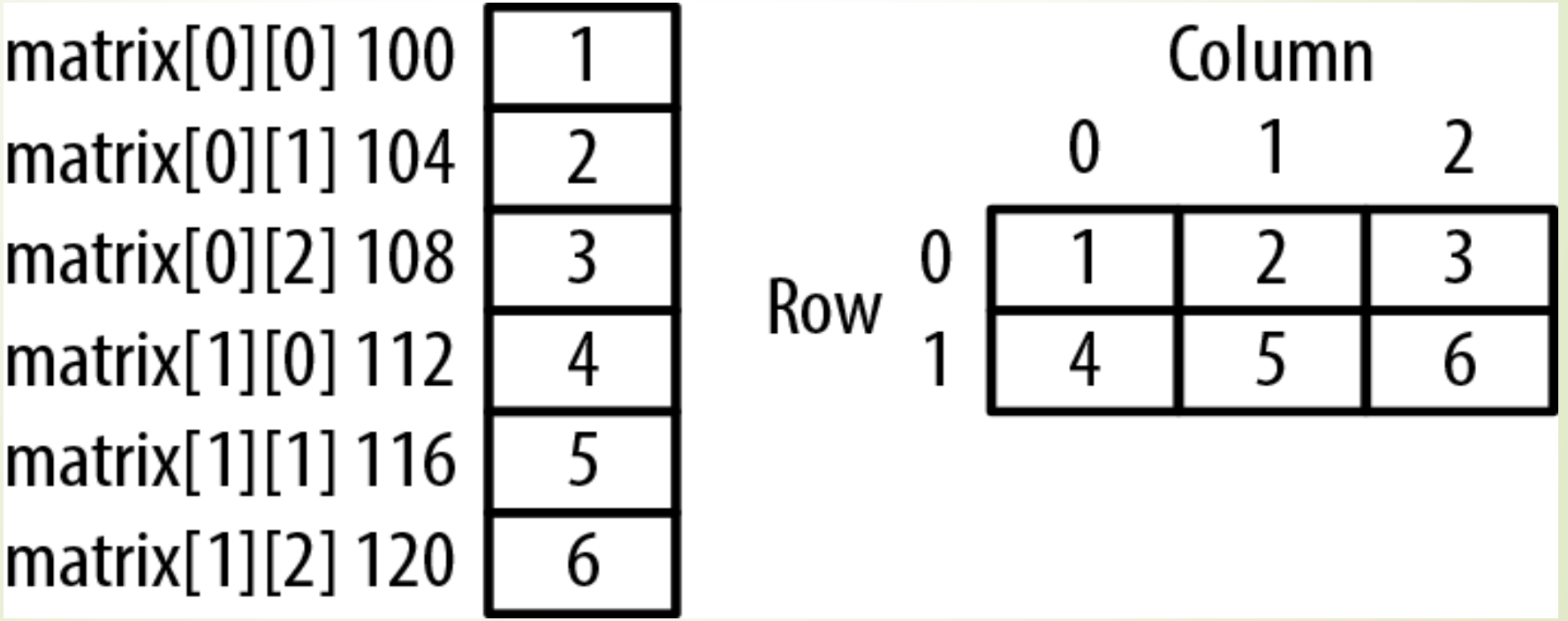
board	[0]	[1]	[2]
[0]	2	3	1
[1]	15	25	13
[2]	20	4	7
[3]	11	18	14

C++ Program to display all elements of an initialized two dimensional array.

```
int main()
{
    int test[3][2] = { {2, -5}, {4, 0}, {9, 1} };
    for(int i = 0; i < 3; ++i)
    {
        for(int j = 0; j < 2; ++j)
        {
            cout<< "test[" << i << "][" << j << "] = " << test[i][j] << endl;
        }
    }
     return 0;
}
```

How 2D-Arrays are stored in memory?

- Two-dimensional arrays are stored in row order
  - The first row is stored first, followed by the second row, followed by the third row and so on



Wanna know more about storage and memory addresses of 2-D Arrays?

- Go to this link:
- https://cse.engineering.nyu.edu/~mleung/CS1114/f04/ch10/MDmemory.htm

Processing Two-Dimensional Arrays

- A two-dimensional array can be processed in three different ways:
  1. Process the entire array
  2. Process a particular row of the array, called row processing
  3. Process a particular column of the array, called column processing

Processing Two-Dimensional Arrays (continued)

- Each row and each column of a two-dimensional array is a one-dimensional array
- When processing a particular row or column of a two-dimensional array
  - We use algorithms similar to processing one-dimensional arrays

```
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;
```

Initialization

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

Print

```
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

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

Sum by Row

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

    cout << "Sum of row " << row + 1 << " = " << sum << endl;
}
```

Sum by Column

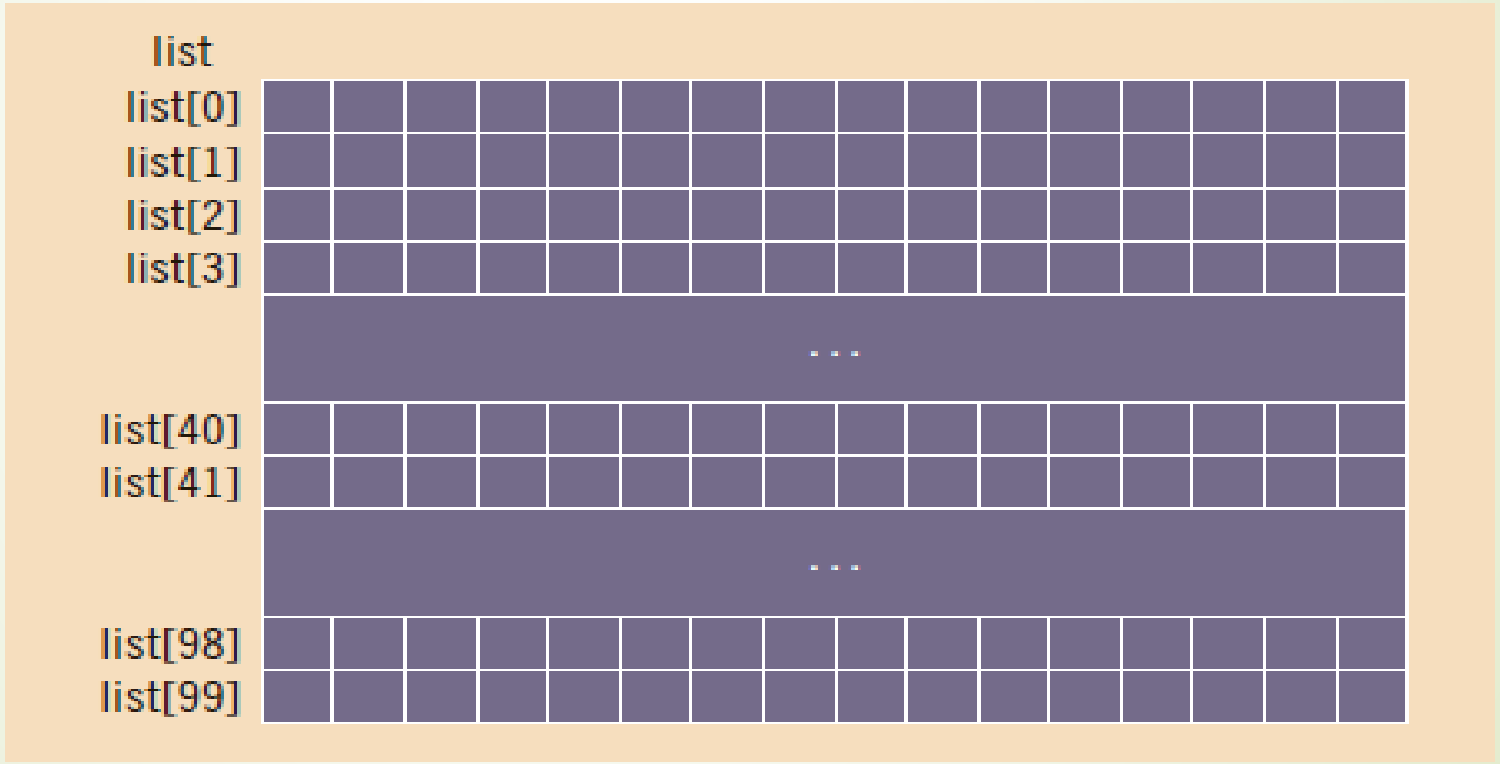
```
  //Sum of each individual column
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
{
     sum = 0;
     for (row = 0; row < NUMBER_OF_ROWS; row++)
          sum = sum + matrix[row][col];

     cout << "Sum of column " << col + 1 << " = " << sum
          << endl;
}
```

Largest Element in Each Row and Each Column

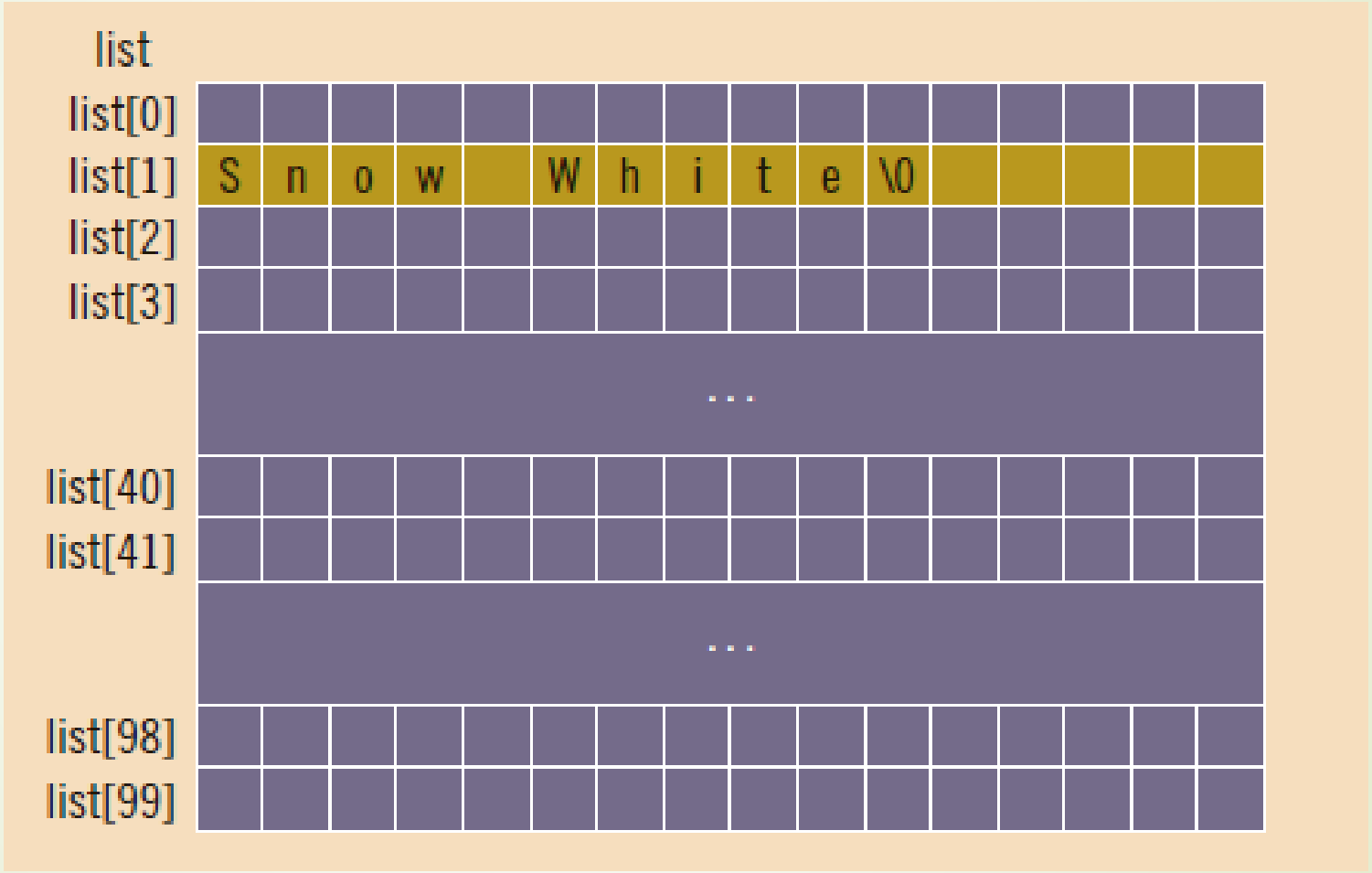
```
  //Largest element in each row
for (row = 0; row < NUMBER_OF_ROWS; row++)
{
    largest = matrix[row][0]; //Assume that the first element
                              //of the row is the largest.
    for (col = 1; col < NUMBER_OF_COLUMNS; col++)
        if (largest < matrix[row][col])
            largest = matrix[row][col];

    cout << "The largest element in row " << row + 1 << " = "
         << largest << endl;
}

  //Largest element in each column
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
{
    largest = matrix[0][col]; //Assume that the first element
                              //of the column is the largest.
    for (row = 1; row < NUMBER_OF_ROWS; row++)
        if (largest < matrix[row][col])
            largest = matrix[row][col];

    cout << "The largest element in column " << col + 1
         << " = " << largest << endl;
}
```

2-D Character Arrays

- `char list[100][16];`



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



Inputting/outputting char 2-d arrays

- Suppose that you want to read and store data in **`list`** and that there is one entry per line.
- The following **`for`** loop accomplishes this task:

```
for (int j = 0; j < 100; j++)
    cin.get(list[j], 16);
```

- The following **`for`** loop outputs the string in each row:

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

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

Passing Two-Dimensional Arrays as Parameters to Functions

- Two-dimensional arrays can be passed as parameters to a function
- By default, arrays are passed by reference
- The base address, that is, the address of the first component of the actual parameter is passed to the formal parameter

Two-Dimensional Arrays

- When declaring a two-dimensional array as a formal parameter
  - Can omit size of first dimension, but not the second
- Number of columns must be specified

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

Consider the following definition of the function `printMatrix`:

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

        cout << endl;
    }
}
```

Exercises

- C++ Program to store temperature of two different cities for a week and display it.
- Find Column/Row wise max, min, average, sum.
- Sort the array Row/Column wise.
- Write a program for adding two matrices of size 2x2, take input from the user.
- Write a program for multiplying two matrices of size 2x2, take input from the user.

References

1. C++ Programming: From Problem Analysis to Program Design, Third Edition
2. https://www.just.edu.jo/~yahya-t/cs115/