Object Oriented Programming Arrays

Week 2 Lecture 1 Spring 2024

Arrays

- An array is a collection of data elements of same type in contiguous memory e.g. list of names, list of scores
- Easier way to compare and use data than having separate variables for data elements

Example

```
//Program that takes five numbers print their
average //and the numbers again
#include<iostream>
using namespace std;
int main(){
   int n1, n2, n3, n4, n5;
   double average;
   cout << "Enter five integers : " ;
   cin >> n1 >> n2 >> n3 >> n4 >> n5 ;

   average = (n1 + n2 + n3 + n4 + n5) / 5.0 ;

   cout << "The average of the given numbers = " <<
average ;
   cout << "\nand the numbers are n1 = " << n1 << " n2 =
" << n2
       << " n3 = " << n3 << " n4 = " << n4
       << " n5 = " << n5 << endl ;
   return 0;
}
```

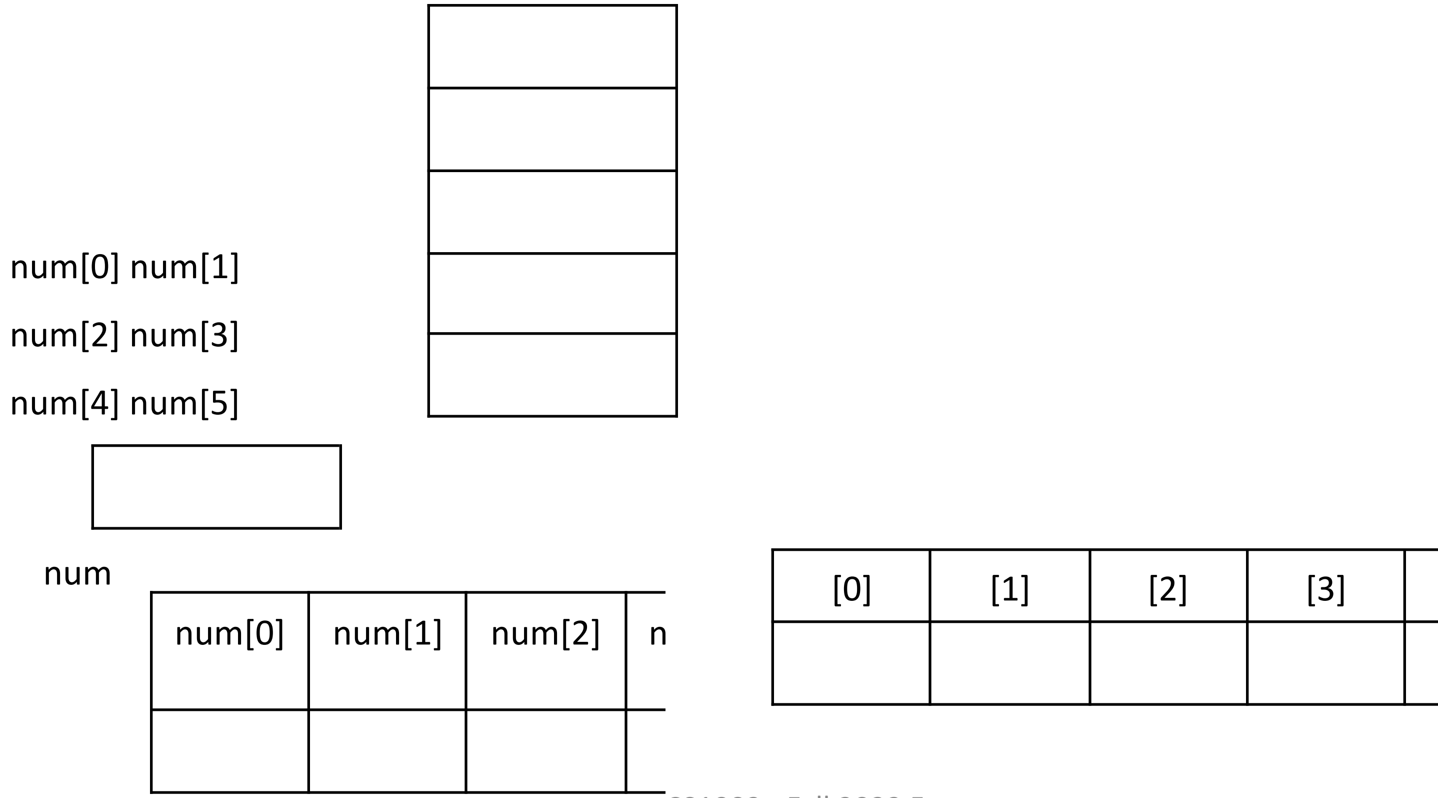
Example

- Five variables must be declared because the numbers are to be printed later
- All variables are of type int, that is, of the same data type
- The way in which these variables are declared indicates that the variables to store these numbers all have the same name, except the last character, which is a number

Arrays

Example:

int num[6];



Defining Arrays

- When defining arrays, specify
  - Name
  - Type of array

- Number of elements

    arrayType arrayName[numberOfElements];

- Examples:

        int c[10];
        float myArray[3284];

- Defining multiple arrays of same type
    - Format similar to regular variables
    - Example:

        int b[100], x[27];

Accessing Array Components (cont'd.)

Array can also be declared as

```
const int SIZE_OF_ARRAY = 20;
int array[SIZE_OF_ARRAY] ;
```

First declare a named constant and then use it to declare an array of this specific size.

When an array is declared its size must be known. You **cannot** do this:

```
int arr_size;
cout << "Enter size of array ";
cin >> arr_size;

int arr[arr_size];
```

Processing One-Dimensional Arrays

- Some basic operations performed on a one dimensional array are:

    - **Initializing**
    - **Inputting** data
    - **Outputting** data stored in an array
    - **Finding** the largest and/or smallest element
- Each operation requires ability to **step through** the elements of the array
    - Easily accomplished by a **loop**

Processing One-Dimensional Arrays (cont'd.)

- Consider the declaration

```
int list[100]; //array of size 100

int i;
```

- Using for loops to access array elements:

```
for (i = 0; i < 100; i++) //Line 1
      //process list[i] //Line 2
```

- Example:

```
for (i = 0; i < 100; i++) //Line 1
      cin >> list[i]; //Line 2
```

Processing One-Dimensional Arrays (cont'd.)

```
double scores[10];
int index; //index also called subscript.
```

Initializing an array

```
    for (index = 0 ; index < 10 ; ++index)
          scores[index] = 0.0 ;
```

Reading data into array

```
    for (index = 0 ; index < 10 ; ++index)
          cin >> scores[index] ;
```

Printing the array

```
    for (index = 0 ; index < 10 ; ++index)
          cout << scores[index] << “ ”;
```

Two-dimensional 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[intDimension1][intDimension2];
```

- where **`intDimension1`** and **`intDimension2`** are expressions yielding positive integer values, and specify the **number of rows** and the **number of columns**, respectively, in the array

Two-dimensional Arrays (cont'd.)

Sales [0] [1] [2] [3] [4]

[0]

[1]

[2]

[3]

[4]

[5]

[6]

[7]

[8]

[9]

Accessing Array Components (cont'd.)

Suppose Sales is a 2D array and:

```
int i = 6;
int j = 2;
```

Then, the following statement:

```
        sales[6][2] = 69.85;
```

Is equivalent to:

```
sales[i][j] = 69.85;
```

So the indices can also be variables.

Sales [0] [1] [2] [3] [4]

		69.85		

[0]

[1]

[2]

[3]

[4]

[5]

[6]

[7]

[8]

[9]

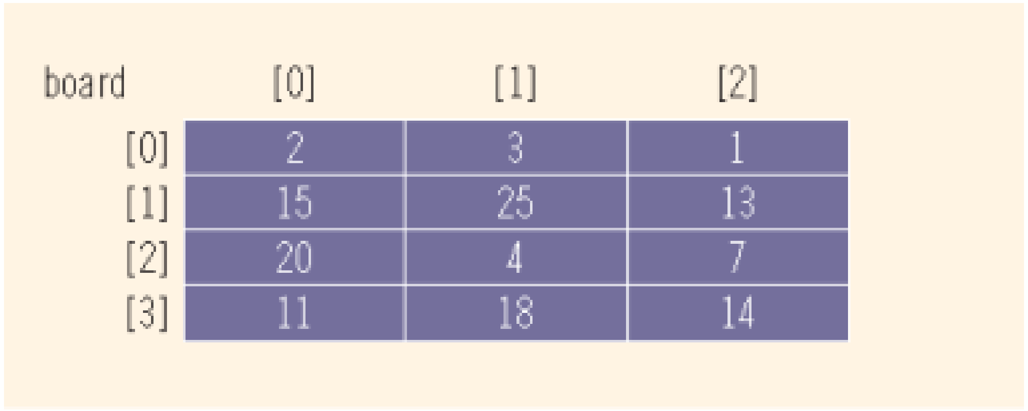
Two-Dimensional Array Initialization During Declaration •

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

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

• 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



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;
```

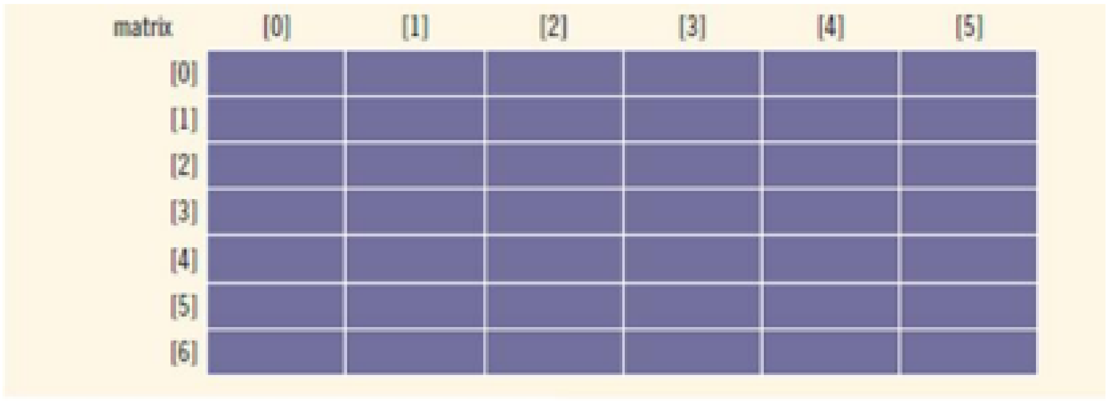


FIGURE 9-15 Two-dimensional array matrix

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 ;
```

Printing the 2D array

```
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 print data from each component of matrix:

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

Input to the 2D array

```
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] ;
```

Class activity#2: Find 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 largest;
int temp;
```

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;
```

```
  }
```

Class Activity#3

- Write a program that multiplies two matrices using C++:

```
int main() {
 int a[10][10], b[10][10], mult[10][10], r1, c1, r2, c2, i, j,
k;  // Storing elements of first matrix.
 cout << endl << "Enter elements of matrix 1:" << endl;
for (i = 0; i < r1; ++i)
 for (j = 0; j < c1; ++j) {
 cout << "Enter element a" << i + 1 << j + 1 << " : ";  cin >>
a[i][j];
 }
 // Storing elements of second matrix. Same as above.




    // Multiplying matrix a and b and storing in array
mult.  for (i = 0; i < r1; ++i)
 for (j = 0; j < c2; ++j) {
 mult[i][j] = 0;
 for (k = 0; k < c1; ++k) {
 mult[i][j] += a[i][k] * b[k][j];
 } }
return 0;
}
```