Topic 6

Two-Dimensional Arrays

It often happens that you want to store collections of values that have a two-dimensional layout.

Such data sets commonly occur in financial and scientific applications.

An arrangement consisting of *tabular data (rows and columns* of values) is called:

a ***two-dimensional array***, or a ***matrix***

Two-Dimensional Array Example

Consider the medal-count data from the 2014 Winter Olympic skating competitions:

Country	Gold	Silver	Bronze
Canada	0	3	0
Italy	0	0	1
Germany	0	0	1
Japan	1	0	0
Kazakhstan	0	0	1
Russia	3	1	1
South Korea	0	1	0
United States	1	0	1

Defining Two-Dimensional Arrays

C++ uses an array with *two* subscripts to store a *2D* array.

```
 const int COUNTRIES = 8;
const int MEDALS = 3;
int counts[COUNTRIES][MEDALS];
```

An array with 8 rows and 3 columns is suitable for storing our medal count data.

Defining Two-Dimensional Arrays – Initializing

Just as with one-dimensional arrays, you *cannot* change the size of a two-dimensional array once it has been defined.

But you can initialize a 2-D array:

```
int counts[COUNTRIES][MEDALS] =
{
      { 0, 3, 0 },
      { 0, 0, 1 },
      { 0, 0, 1 },
      { 1, 0, 0 },
      { 0, 0, 1 },
      { 3, 1, 1 },
      { 0, 1, 0 }
      { 1, 0, 1 }
};
```

Self Check: Declaring 2D Arrays

Write the code statement to declare an array `"a"` of integers with the specified properties.

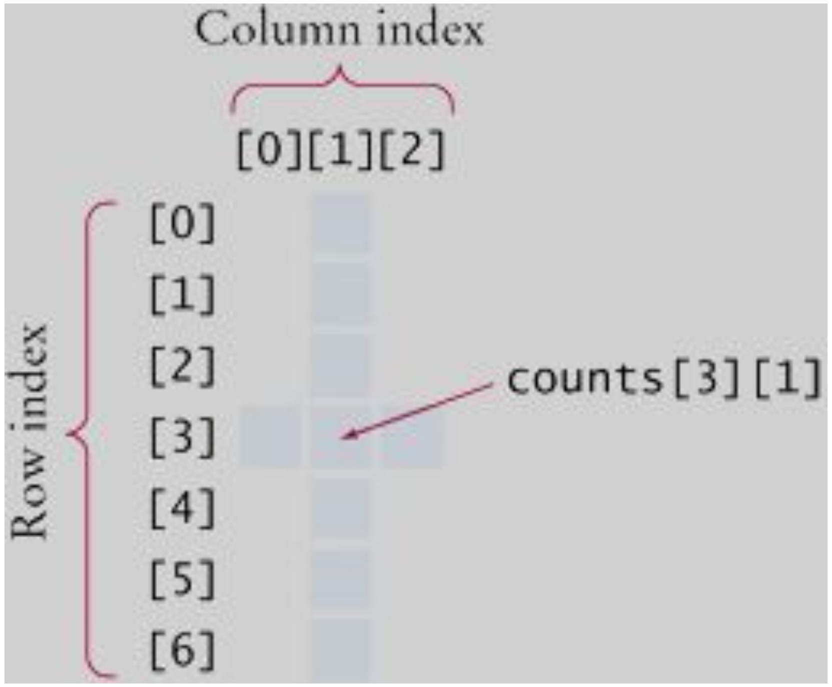
Initialize elements to zero unless otherwise specified.

1. With 3 rows and 2 columns:

2. With 2 rows and 3 columns:

3. With 2 rows and 2 columns, containing 1 when the row and column index are the same:

Two-Dimensional Arrays – Accessing Elements



```
 // copy to value what is currently
// stored in the array at [3][1]
int value = counts[3][1];

 // Then set that position in the array to 8
 counts[3][1] = 8;
```

Nested Loop to Print All Elements in a 2D Array

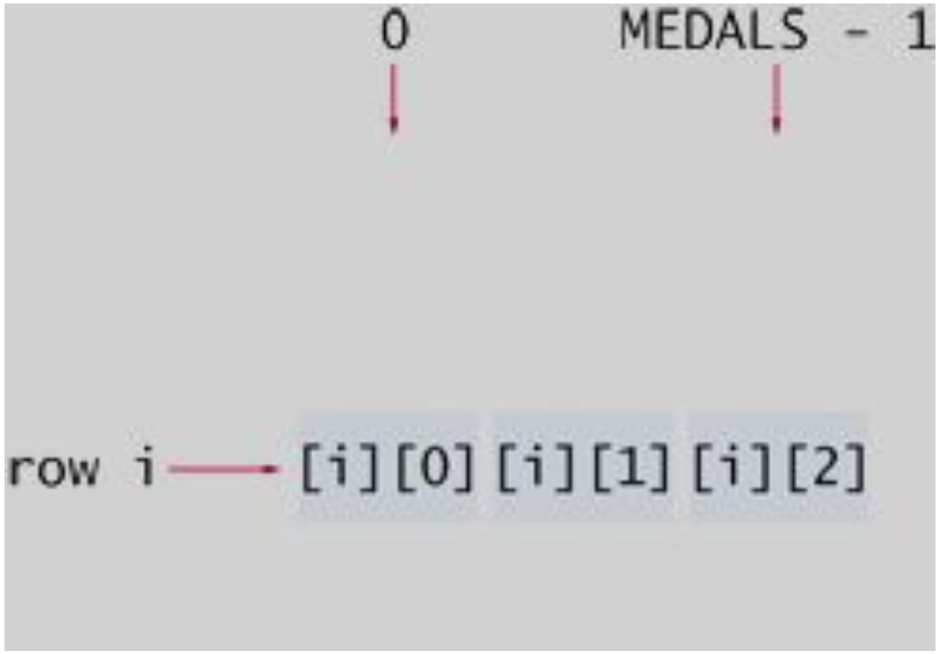
```
for (int i = 0; i < COUNTRIES; i++)
{
   // Process the ith row
   for (int j = 0; j < MEDALS; j++)
   {
      // Process the jth column in the ith row
      cout << setw(8) << counts[i][j];
   }
   // Start a new line at the end of the row
   cout << endl;
}
```

Computing Row and Column Totals

We must be careful to get the right indices.

For each row **`i`**, we must use the column indices:

`0, 1, … (MEDALS -1)`



Computing Row and Column Totals: Code Example

Column totals:

Let `j` be the silver column:

```
 int total = 0; //loop to sum down the rows
for (int i = 0; i < COUNTRIES; i++)
{
   total = total + counts[i][j];
}
```

Two-Dimensional Array Parameters

When passing a two-dimensional array to a function, you must specify the number of columns *as a constant* when you write the parameter type, so the compiler can pre-calculate the memory addresses of individual elements.

This function computes the total of a given row.

```
const int COLUMNS = 3;
int row_total(int table[][COLUMNS], int row)
{
   int total = 0;
   for (int j = 0; j < COLUMNS; j++)
   {
      total = total + table[row][j];
   }
   return total;
}
```

Two-Dimensional Array Parameter Columns Hardwired

That function works for only arrays of 3 columns.

If you need to process an array
with a different number of columns, like 4,

you would have to write
a different function
that has 4 as the parameter.

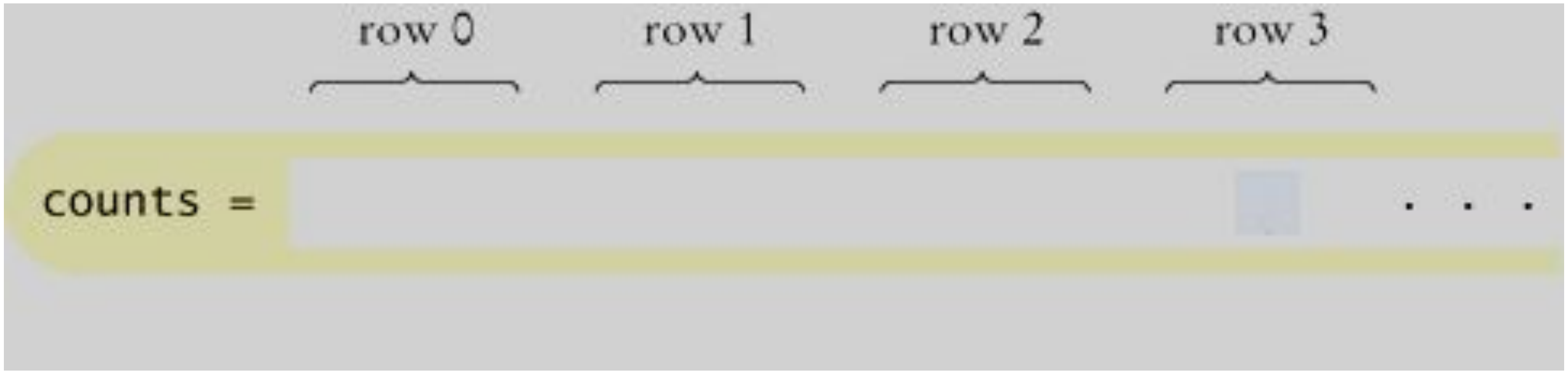
Two-Dimensional Array Storage

What's the reason behind this?

Although the array appears to be two-dimensional, the elements are still stored as a linear sequence.

`counts` is stored as a sequence of rows, each 3 long.
So where is **`counts[3][1]`**?
The offset (calculated by the compiler) from the start of the array is

`3 x` ***`number of columns`*** **`+ 1`**



Two-Dimensional Array Parameters: Rows

The **`row_total`** function did not need to know the number of rows of the array.

If the number of rows is required, pass it in:

```
int column_total(int table[][COLUMNS], int rows, int col)
{
   int total = 0;
   for (int i = 0; i < rows; i++)
   {
      total = total + table[i][col];
   }
   return total;
}
```

Two-Dimensional Array Parameters: Complete Code (1)

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;
const int COLUMNS = 3;
/**
   Computes the total of a row in a table.
   @param table a table with 3 columns
   @param row the row that needs to be totaled
   @return the sum of all elements in the given row
*/
double row_total(int table[][COLUMNS], int row)
{
   int total = 0;
   for (int j = 0; j < COLUMNS; j++)
   {
      total = total + table[row][j];
   }
   return total;
}
```

Two-Dimensional Array Parameters: Complete Code (2)

```
int main()
{
    const int COUNTRIES = 8;
    const int MEDALS = 3;

    string countries[] =
    {"Canada","Italy","Germany","Japan", "Kazakhstan",
   "Russia", "South Korea", "United States"};

   int counts[COUNTRIES][MEDALS] =
    {
       { 0, 3, 0 },
       { 0, 0, 1 },
       { 0, 0, 1 },
       { 1, 0, 0 },
       { 0, 0, 1 },
       { 3, 1, 1 },
       { 0, 1, 0 }
       { 1, 0, 1 }
    };
```

Two-Dimensional Array Parameters: Complete Code (3)

```
   cout << "     Country  Gold  Silver  Bronze   Total"
       << endl;

   // Print countries, counts, and row totals
   for (int i = 0; i < COUNTRIES; i++)
   {
      cout << setw(15) << countries[i];
      // Process the ith row
      for (int j = 0; j < MEDALS; j++)
      {
         cout << setw(8) << counts[i][j];
      }
      int total = row_total(counts, i);
      cout << setw(8) << total << endl;
   }
   return 0;
}
```

Practice It: 2D Array Parameters

Insert the missing statement. The function should return the result of adding the values in the first row of the 2D array received as argument.

```
int add_first_row(int array[][MAX_COLS], int rows,
int cols)
{
   int sum = 0;
   for (int k = 0; k < cols; k++)
   {
      sum = sum + _________________;
   }
   return sum;
}
```

Arrays – Fixed Size is a Drawback

The size of an array *cannot* be changed after it is created.

You have to get the size right – *before* you define an array.

The compiler has to know the size to build it.
and a function must be told about the number
elements and possibly the capacity.

It cannot hold more than it’s initial capacity.

Next we'll discuss vectors, which have variable size and some other programmer-friendly features lacking in arrays.

CHAPTER SUMMARY #1

Use arrays for collecting values.

- Use an array to collect a sequence of values of the same type.
- Individual elements in an array values are accessed by an integer index i, using the notation `values[i]`.
- An array element can be used like any variable.
- An array index must be at least zero and less than the size of the array.
- A bounds error, which occurs if you supply an invalid array index, can corrupt data or cause your program to terminate.
- With a partially filled array, keep a companion variable for the current size.

Be able to use common array algorithms.

- To copy an array, use a loop to copy its elements to a new array.
- When separating elements, don’t place a separator before the first element.
- A linear search inspects elements in sequence until a match is found.
- Before inserting an element, move elements to the end of the array starting with the last one.
- Use a temporary variable when swapping two elements.

CHAPTER SUMMARY #2

Implement functions that process arrays.

- When passing an array to a function, also pass the size of the array.
- Array parameters are always reference parameters.
- A function's return type cannot be an array.
- When a function modifies the size of an array, it needs to tell its caller.
- A function that adds elements to an array needs to know its capacity.

Be able to combine and adapt algorithms for solving a programming problem.

- By combining fundamental algorithms, you can solve complex programming tasks.
- Be familiar with the implementation of fundamental algorithms so that you can adapt them.

CHAPTER SUMMARY #3

Discover algorithms by manipulating physical objects.

- Use a sequence of coins, playing cards, or toys to visualize an array of values.
- You can use paper clips as position markers or counters.

Use two-dimensional arrays for data that is arranged in rows and columns.

- Use a two-dimensional array to store tabular data.
- Individual elements in a two-dimensional array are accessed by using two subscripts, `array[i][j]`.
- A two-dimensional array parameter must have a fixed number of columns.