2D Arrays

Due this week

- **Project 1**
  - Write solutions in VSCode and paste in Autograder, **Project1-CodeRunner**
  - **Honor Code MCQ**
  - Zip your .cpp files and submit on canvas **Project 1**. Check the due date! **No late submissions!!**
  - **Mandatory Grading Interview through Oct 12**!
  - Follow instructions from write-up
- No Quiz this week
- 3-2-1 Today

2D Arrays

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.

- 2D arrays are built up as an array of 1D arrays!
- Each row is a 1D array

Defining Two-Dimensional Arrays – Initializing

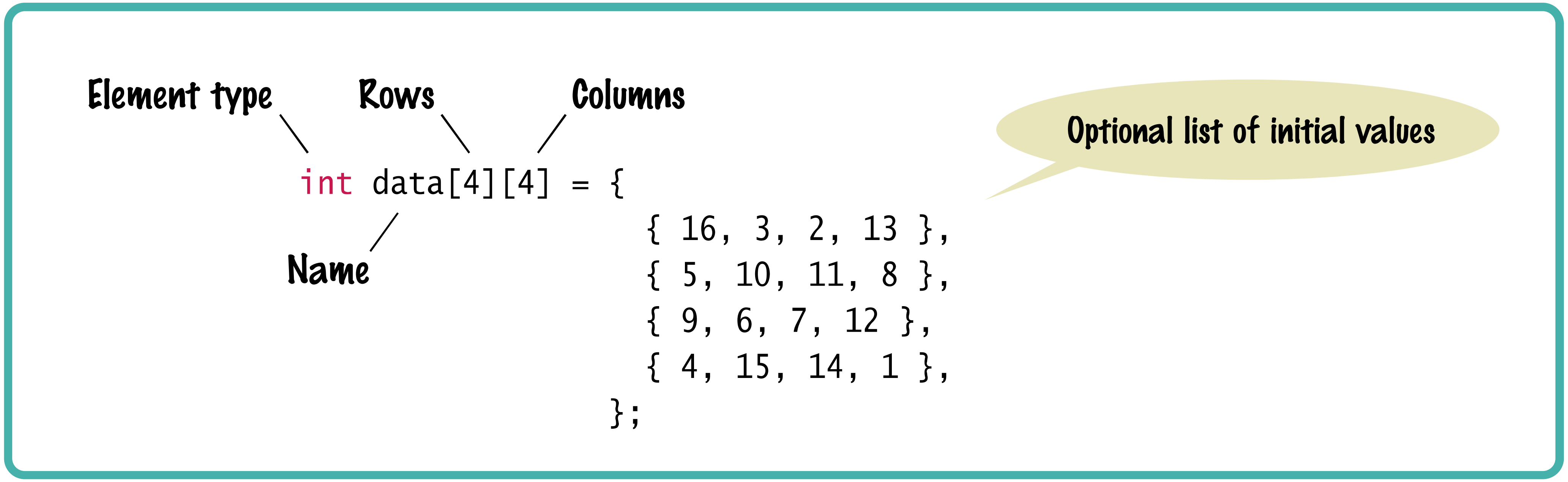
Just as with one-dimensional arrays, you *cannot* change the size of a two-dimensional array once it has been defined.

You can initialize them.

```
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 }
  };
```

Defining 2D arrays

Two-Dimensional Array Definition



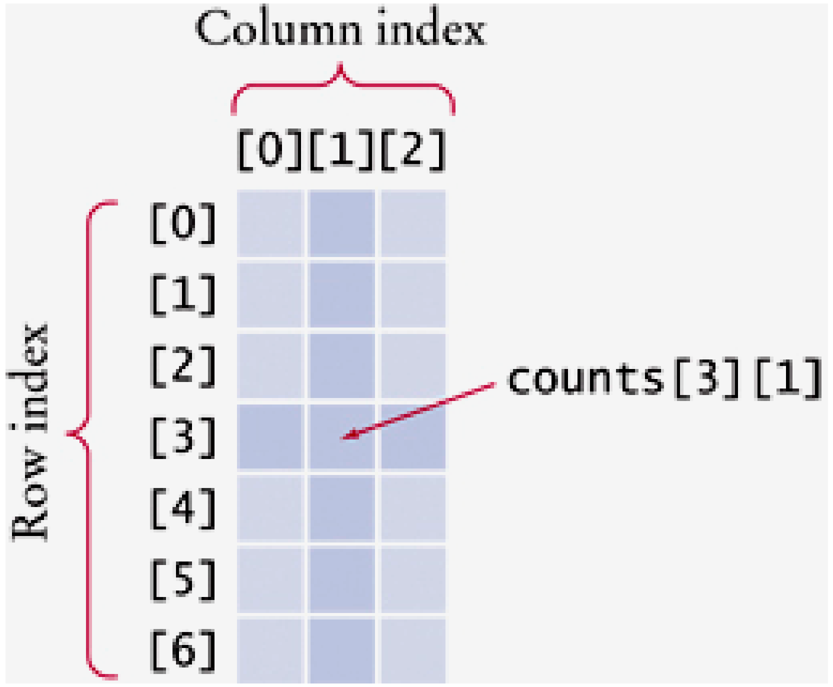
```
int data[4][4] = {
                     { 16, 3, 2, 13 },
                     { 5, 10, 11, 8 },
                     { 9, 6, 7, 12 },
                     { 4, 15, 14, 1 },
                   };
```

Two-Dimensional Arrays – Accessing Elements

```
// copy to num what is currently
// stored in the array at [3][1]

int num = counts[3][1];


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



CODE

Print All Elements in a 2D Array

```
[0][0], [0][1], [0][2]
// Process the 1st row:
for (int j = 0; j < MEDALS; j++)
{
  cout << setw(8) << counts[0][j];
}
```

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

Print All Elements in a 2D Array

```
[0][0], [0][1], [0][2]
[1][0], [1][1], [1][2]
[2][0], [2][1], [2][2]
for (int j = 0; j < MEDALS; j++)
{
  cout << setw(8) << counts[i][j];
}
```

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

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;
}
```

In order to print each element, we need two for loops:

- one to loop over all rows,
- and another to loop over all columns.

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];
}
```

Multidimensional Array Parameters

- Similar to one-dimensional array
    - 1st dimension size not given (Provided as second parameter)
    - 2nd dimension size IS given
- Example:

```
void DisplayPage(const char p[][100], int sizeDimension1)
{
	for (int index1=0; index1<sizeDimension1; index1++)
	{
		for (int index2=0; index2 < 100; index2++)
			cout << p[index1][index2];
		cout << endl;
	}
}
```

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;
}
```

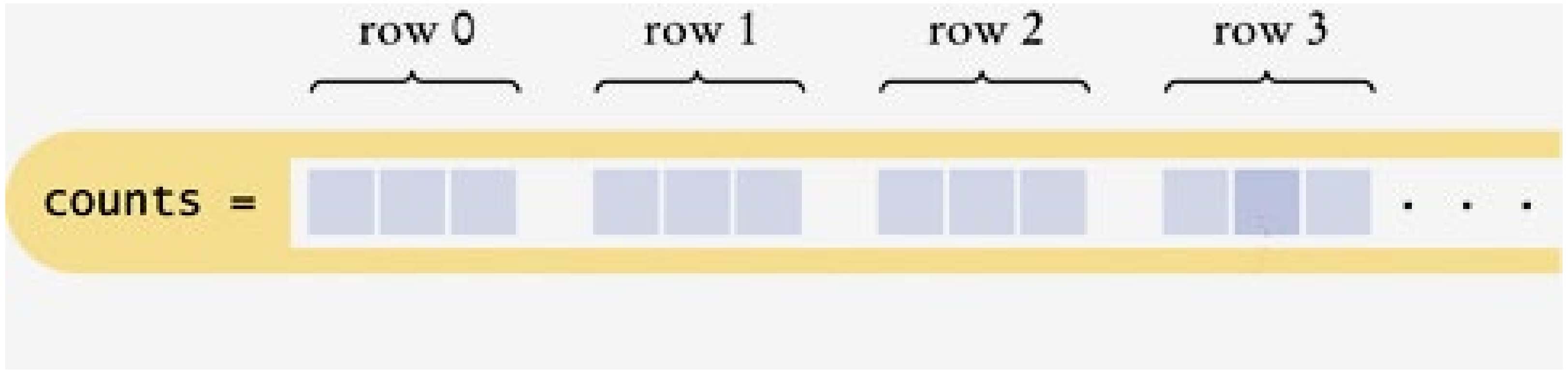
Two-Dimensional Array Parameter Columns Hardwired

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;
}
```

Arrays -- fixed size can be 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 needs to know the size in order to build the array, and functions need to be told number of elements in array, and possibly its capacity (and arrays can’t hold more than their initial capacity)

- Later, we’ll talk about vectors, which can have variable size and some other nice flexible features that arrays don’t have.

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;
}
```