Chapter Six: Arrays and Vectors II

Slides by Evan Gallagher & Nikolay Kirov

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Chapter Goals

- To become familiar with using vectors to collect values
- To write functions that receive and return vectors
- To be able to use two-dimensional arrays

Copyright © 2012 by John Wiley & Sons. All rights reserved

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.

Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Arrays

An arrangement consisting of *tabular data*:
rows and columns of values

(*no, it's not tax time again*)

is called:

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

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Arrays

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Arrays

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Arrays

It's *my* time!

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Arrays

9.94?

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Arrays

9.98?

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Arrays

Consider this data from the 2010 Olympic skating competitions:

		Gold	Silver	Bronze
Canada	1	0	1	
China1	1	0		
Germany	0	0	1	
Korea1	0	0		
Japan0	1	1		
Russia	0	1	1	
United States		1	1	0

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Arrays

Consider this data from the 2006
Olympic skating competitions:

me!

```
                  Gold    Silver  Bronze
Canada       1     0     1
China1       1     0
Germany      0     0     1
Korea1       0     0
Japan0       1     1
Russia       0     1     1
United States  1     1     0
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Defining Two-Dimensional Arrays

C++ uses an array with *two* subscripts
to store a *two*-dimensional array.

```
const int COUNTRIES = 7;
const int MEDALS = 3;
int counts[COUNTRIES][MEDALS];
```

An array with 7 rows and 3 columns
is suitable for storing our medal count data.

Copyright © 2012 by John Wiley & Sons. All rights reserved

Defining Two-Dimensional Arrays – Unchangeable Size

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

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Defining Two-Dimensional Arrays – Initializing

But you can initialize a 2-D array:

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


Copyright © 2012 by John Wiley & Sons. All rights reserved

Defining Two-Dimensional Arrays

Syntax 6.3 Two-Dimensional Array Definition

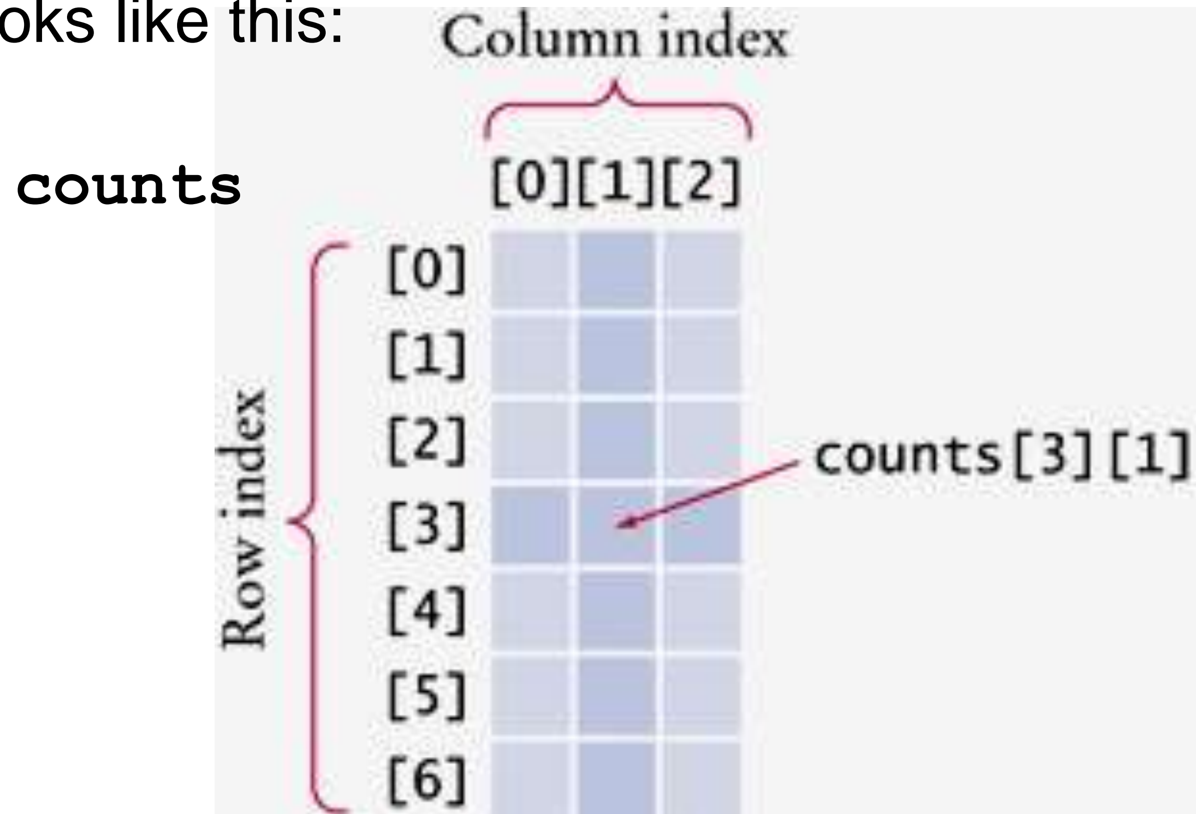

```
int data[4][4] = {
        { 16, 3, 2, 13 },
        { 5, 10, 11, 8 },
        { 9, 6, 7, 12 },
        { 4, 15, 14, 1 },
    };
```


Copyright © 2012 by John Wiley & Sons. All rights reserved

Defining Two-Dimensional Arrays – Accessing Elements

The Olympic array looks like this:

`counts`

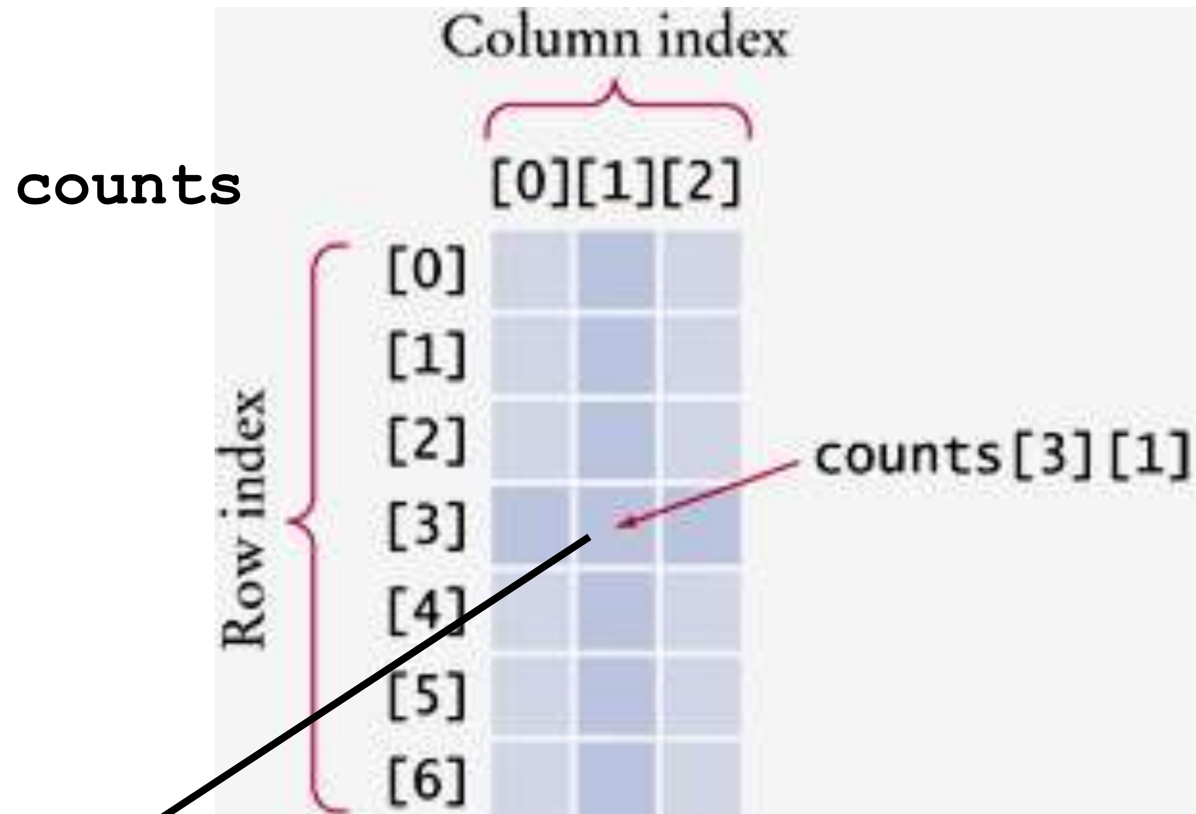


Access to the second element in the fourth row is:

`counts[3][1]`

Copyright © 2012 by John Wiley & Sons. All rights reserved

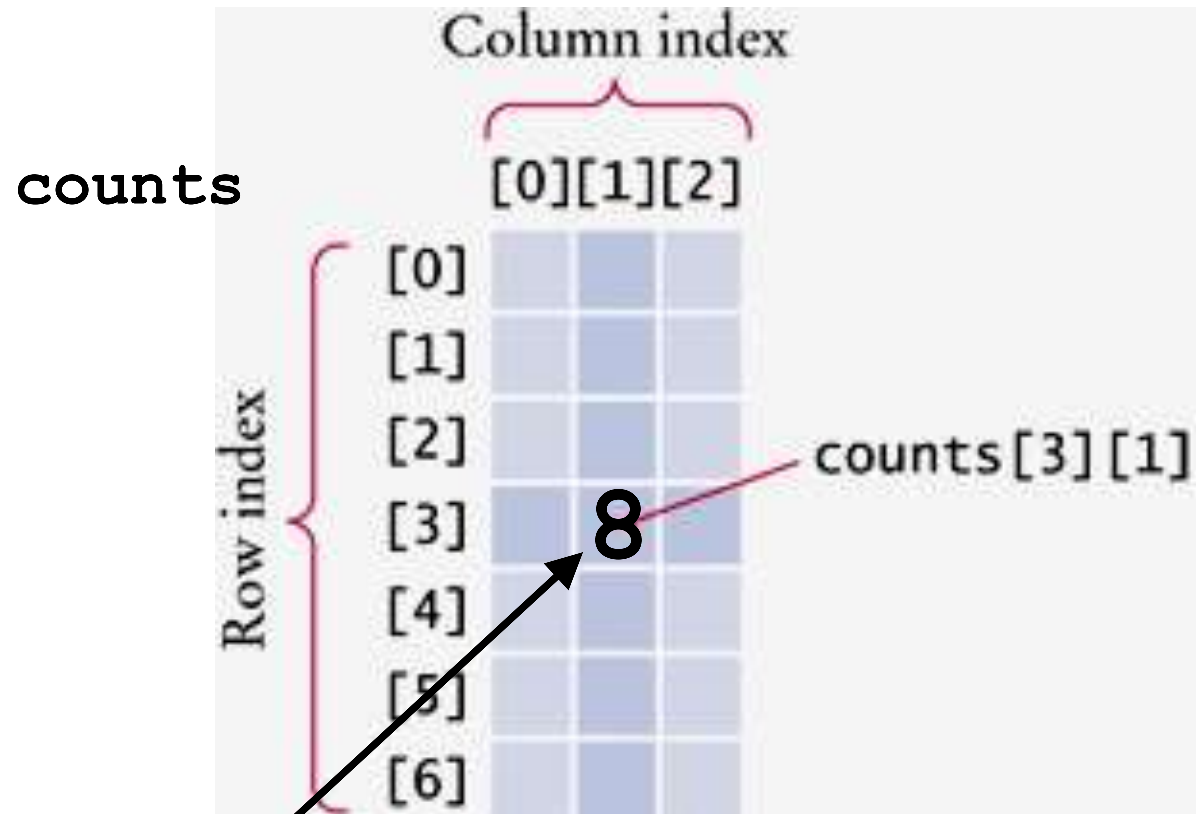
Defining Two-Dimensional Arrays – Accessing Elements




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

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Defining Two-Dimensional Arrays – Accessing Elements




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

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Arrays

I’d like to see the results now, please.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Arrays

Gladly:

```
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;
}
```


Copyright © 2012 by John Wiley & Sons. All rights reserved

Computing Row and Column Totals

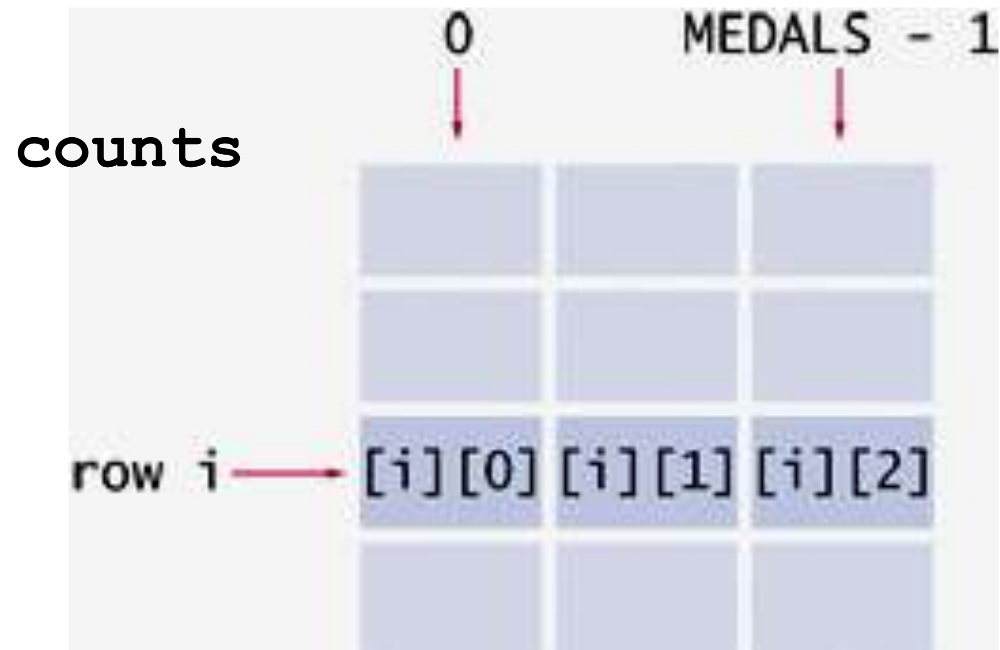
A common task is to compute row or column totals.

In our example,
the row totals give us the total number
of medals won by a particular country.

Copyright © 2012 by John Wiley & Sons. All rights reserved

Computing Row and Column Totals

We must be careful to get the right indices.



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

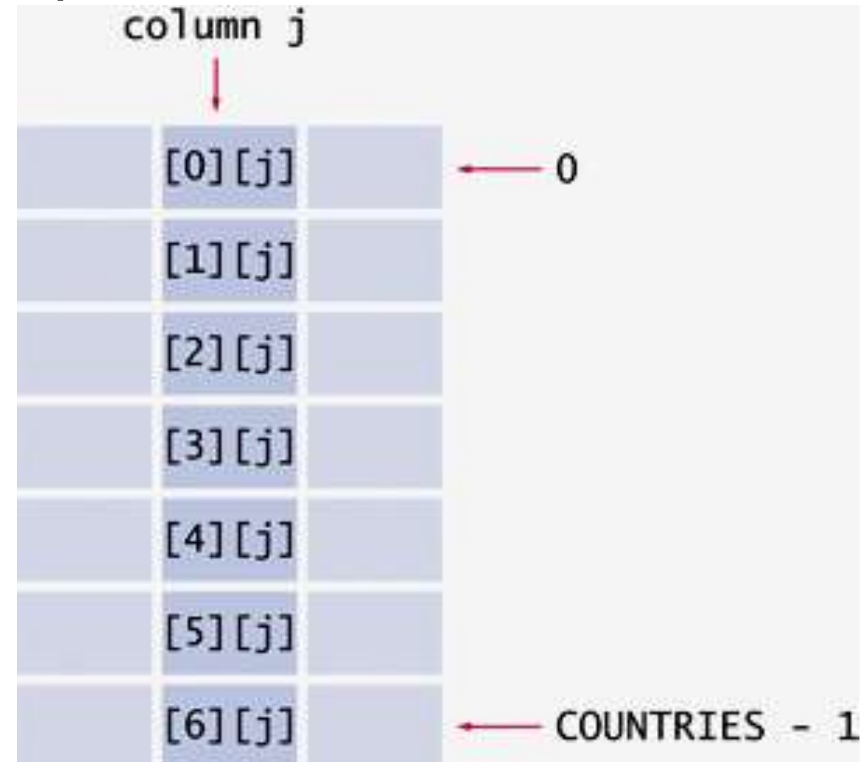
0, 1, … (MEDALS -1)

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Computing Row and Column Totals

How many of each kind of medal (*metal!*) was won by the set of these particular countries?

counts



That would be a column total.

Let `j` be the silver column:

```
int total = 0;
for (int i = 0; i < COUNTRIES; i++)
{
   total = total + counts[i][j];
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

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.

```
table[][COLUMNS]
```


Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Array Parameters

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;
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Array Parameters

```
int row_total(int table[][COLUMNS], int row)
```

In this function, to find the element `table[row][j]`
the compiler generates code
by computing the offset

Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Array Parameters

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.

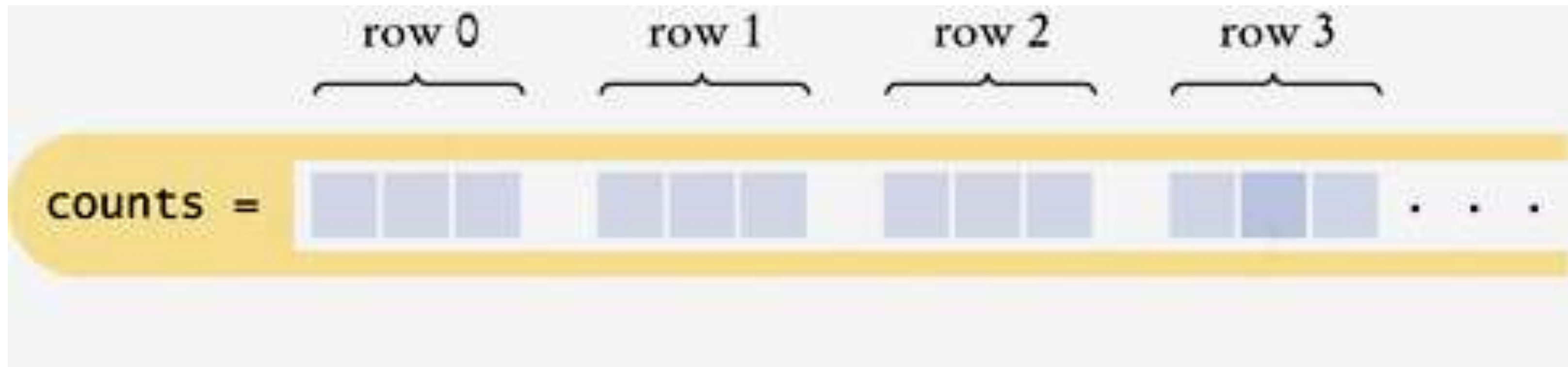
Hm.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Array Parameters

What's the reason behind this?

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



C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

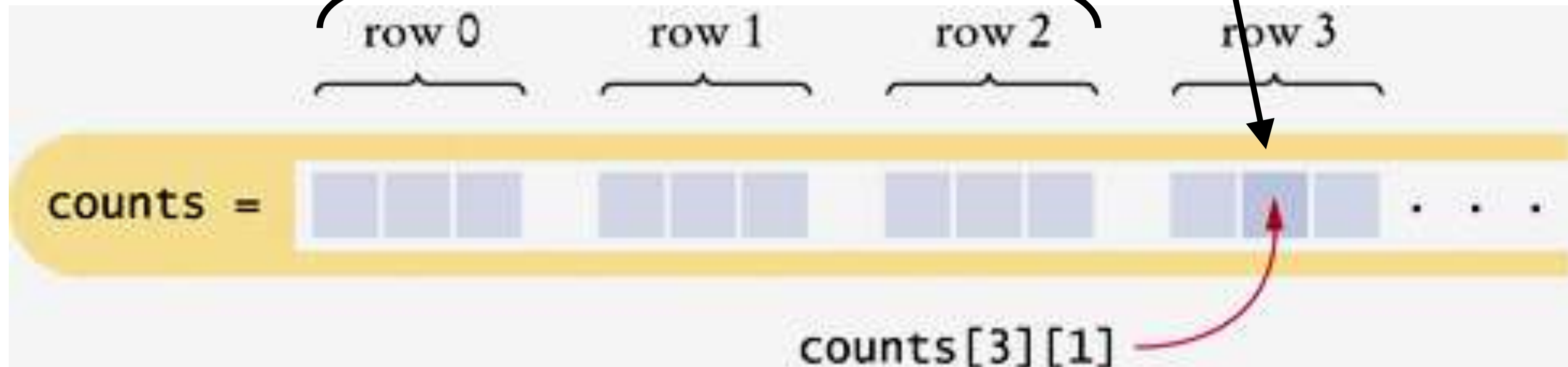
Two-Dimensional Array Parameters

`counts` is stored as a sequence of rows, each 3 long.

So where is **`counts[3][1]`**?

The offset from the start of the array is

`3 x number of columns + 1`



C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Array Parameters

```
int row_total(int table[][COLUMNS], int row)
```

Notice the empty square brackets.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

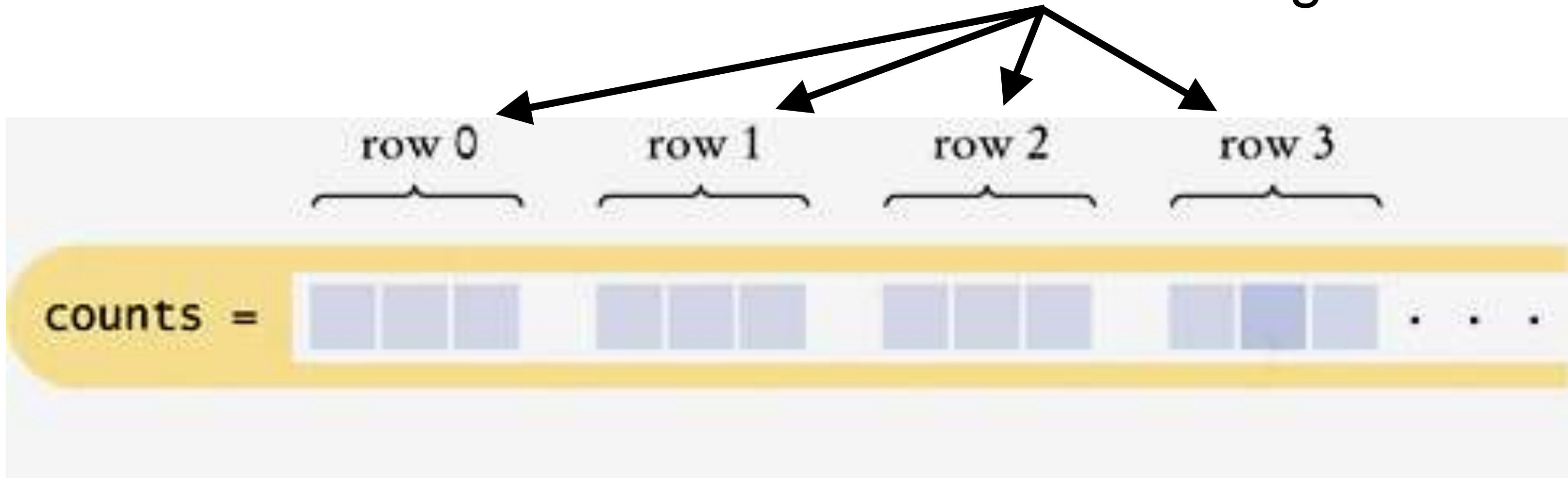
Two-Dimensional Array Parameters

```
int row_total(int table[][COLUMNS], int row)
```

`table[]` looks like a normal 1D array.

It is!

Each element is `COLUMNS` `int`s long.

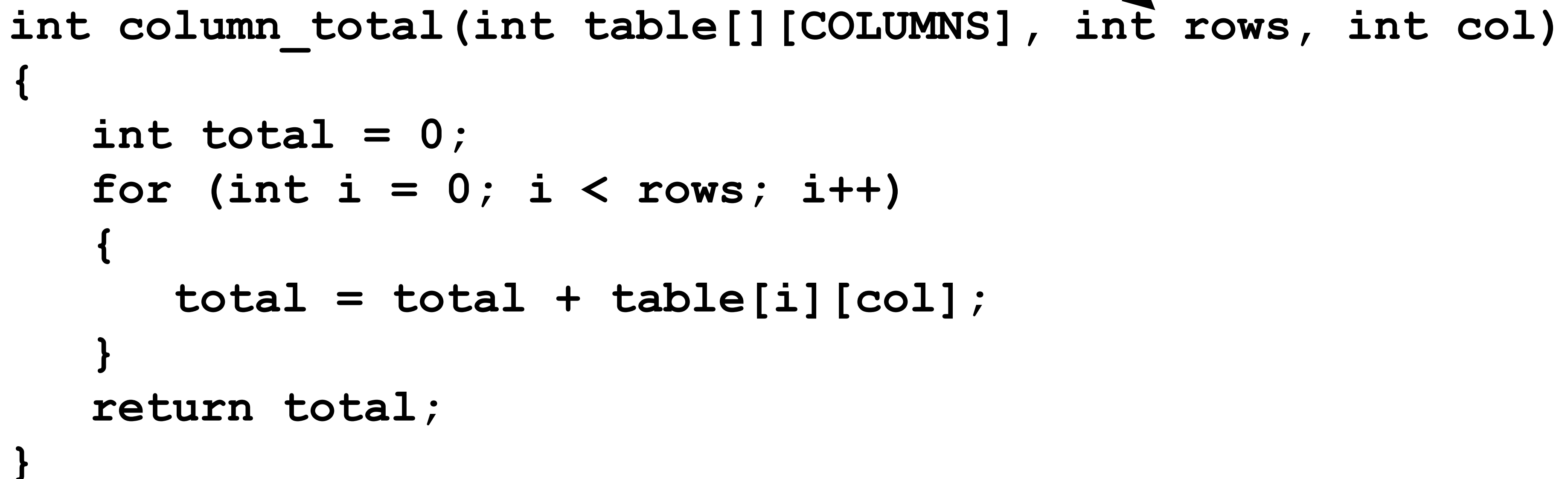


rstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Array Parameters

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:

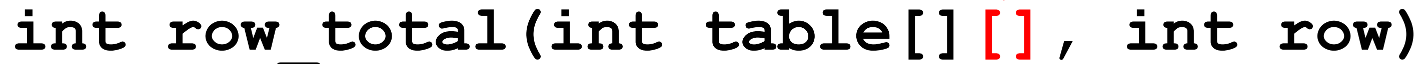


```
int column_total(int table[][COLUMNS], int rows, int col)
{
   int total = 0;
   for (int i = 0; i < rows; i++)
   {
      total = total + table[i][col];
   }
   return total;
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Array Parameters – Common Error

Leaving out the columns value is a very common error.



```
int row_total(int table[][], int row)
...
```

The compiler doesn't know how "long" each row is!

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Array Parameters – Not an Error

Putting a value for the rows is not an error.

```
int row_total(int table[17][COLUMNS], int row)
...
```

The compiler just ignores whatever you place there.

Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Array Parameters – Not an Error

Putting a value for the rows is not an error.

...

The compiler just ignores whatever you place there.

...

Never mind

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Array Parameters

Here is the complete program for medal and column counts.

ch06/medals.cpp

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

const int COLUMNS = 3;
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Array Parameters

ch06/medals.cpp

```
/**
   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;
}
```


Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Array Parameters

ch06/medals.cpp

```
int main()
{
   const int COUNTRIES = 7;
   const int MEDALS = 3;

   string countries[] =
      {
         "Canada",
         "China",
         ″Germany",
         ″Korea",
         ″Japan",
         ″Russia",
         "United States"
      };
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Array Parameters

ch06/medals.cpp

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


Copyright © 2012 by John Wiley & Sons. All rights reserved

Two-Dimensional Array Parameters

ch06/medals.cpp

```
   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;
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Arrays – One 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.

Copyright © 2012 by John Wiley & Sons. All rights reserved

Arrays – One Drawback

Wouldn't it be good if there were
something that never filled up?

Copyright © 2012 by John Wiley & Sons. All rights reserved

Vectors

A ***vector***

is not fixed in size when it is created

and

it does not have the limitation
of needing an auxiliary variable

AND

you can keep putting things into it

forever!

Well, conceptually forever.
(There's only so much RAM.)

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Defining Vectors

When you define a vector, you
must specify the type of the elements.

```
vector<double> data;
```

Note that the element type is enclosed in angle brackets.

`data` can contain *only* **`double`**s

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Defining Vectors

By default, a vector is empty when created.

```
vector<double> data; // data is empty
```


Copyright © 2012 by John Wiley & Sons. All rights reserved

Defining Vectors

You can specify the initial size.

You still must specify the type of the elements.

For example, here is a definition of a vector of **`double`**s whose initial size is **`10`**.

```
vector<double> data(10);
```

This is very close to the **`data`** *array* we used earlier.

Copyright © 2012 by John Wiley & Sons. All rights reserved

Defining Vectors

Syntax 6.2 Defining a Vector

Element type

Size

```
vector<double> data(10);
```

Name

If you omit the size and the parentheses, the vector has size 0.

Use brackets to access an element.

```
data[i] = 0;
```

The index must be ≥ 0 and < `data.size()`.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Defining Vectors

Table 2 Defining Vectors

Code	Description
`vector<int> numbers(10);`	A vector of ten integers.
`vector<string> names(3);`	A vector of three strings.
`vector<double> values;`	A vector of size `0`.
`vector<double> values();`	**Error:** Does not define a vector.

`vector<int> numbers;`
`for (int i = 1; i <= 10; i++)`
`{`
`   numbers.push_back(i);`
`}`
	A vector of ten integers, filled with 1, 2, 3, ..., 10.

`vector<int> numbers(10);`
`for (int i = 0; i < numbers.size(); i++)`
`{`
`   numbers[i] = i + 1;`
`}`
	Another way of defining a vector of ten integers and filling it with 1, 2, 3, ..., 10.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Accessing Elements in Vectors

You access the elements in a vector
the same way as in an array, using an index.

```
vector<double> values(10);
//display the forth element
cout << values[3] << end;
```

HOWEVER...

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Accessing Elements in Vectors

It is an error to access a element that is not there in a vector.

```
vector<double> values;
//display the forth element
cout << values[3] << end;
```

Copyright © 2012 by John Wiley & Sons. All rights reserved

`push_back`

So how do you put values into a vector?

You push 'em—

—in the back!

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back and pop_back

The method ***push_back*** is used to put a value into a vector:

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

`push_back` and `pop_back`

```
values.push_back( 32 );
```

adds the value **`32.0`** to the vector named **`values`**.

The vector increases its size by 1.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

`pop_back`

And how do you take them out?

You pop ’em!

—from the back!

Copyright © 2012 by John Wiley & Sons. All rights reserved

`push_back` and `pop_back`

The method ***`pop_back`*** removes

the last value placed into the vector with **`push_back`**.

```
values.pop_back();
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back and **pop_back**

```
values.pop_back();
```

removes the last value from the vector named **values**

and the vector decreases its size by 1.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

`push_back` Adds an Element

```
vector<double> values;
```

```
values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back Adds an Element


```
vector<double> values;
```

```
values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

values is an empty vector.
Its size is 0.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

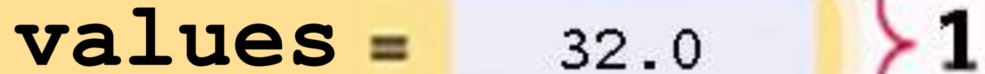
push_back Adds an Element

values } 0

```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back Adds an Element




```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

32 is placed into the vector.
Its size is now 1.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back Adds an Element

values = 32.0 } 1

```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back Adds an Element

```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

54 is placed into the vector.
It now contains the elements
32.0 and **54.0**,
and its size is **2**.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back Adds an Element

values = 32.0 54.0 } 2

```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

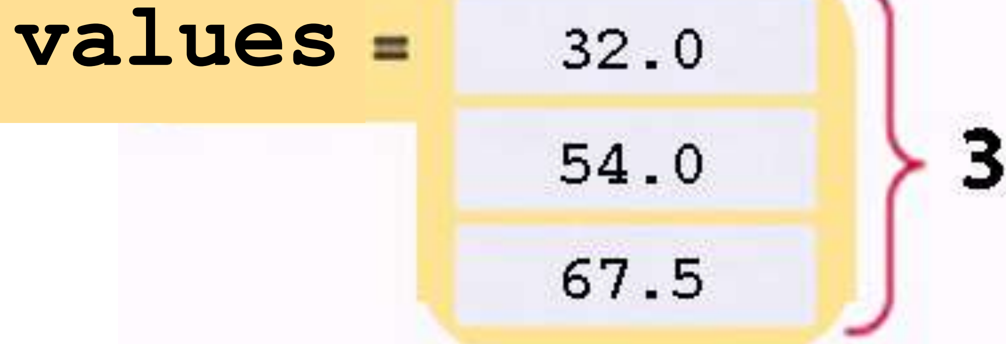
`push_back` Adds an Element


```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

`67.5` is placed into the vector.
It now contains the elements
`32.0`, `54.0` and `67.5`,
and its size is `3`.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back Adds an Element




```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back Adds an Element

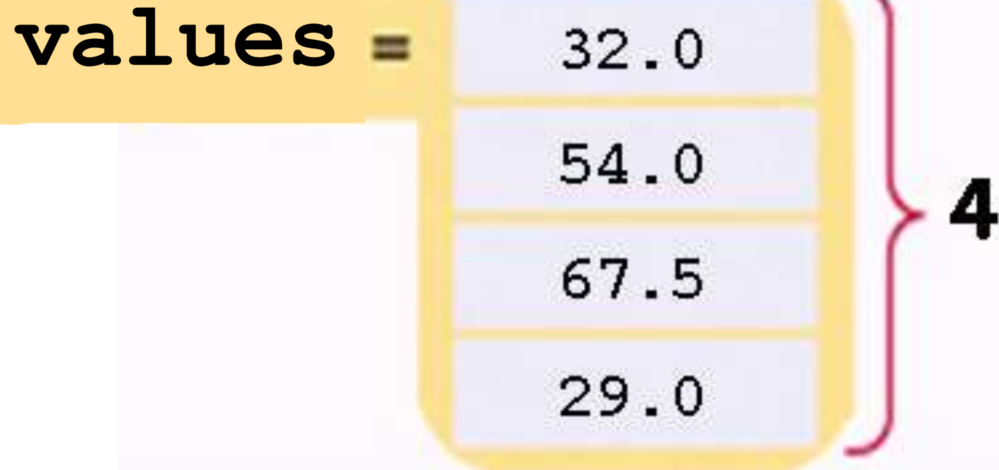
```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

values = 32.0, 54.0, 67.5, 29.0 } 4

29 is placed into the vector.
It now contains the elements `32.0`, `54.0`, `67.5` and `29.0`, and its size is `4`.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back Adds an Element




```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back Adds an Element

values = 32.0, 54.0, 67.5, 29.0, 65.0 (size 5)

```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

65 is placed into the vector.
It now contains the elements
`32.0`, `54.0`, `67.5`, `29.0` and `65.0`,
and its size is `5`.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

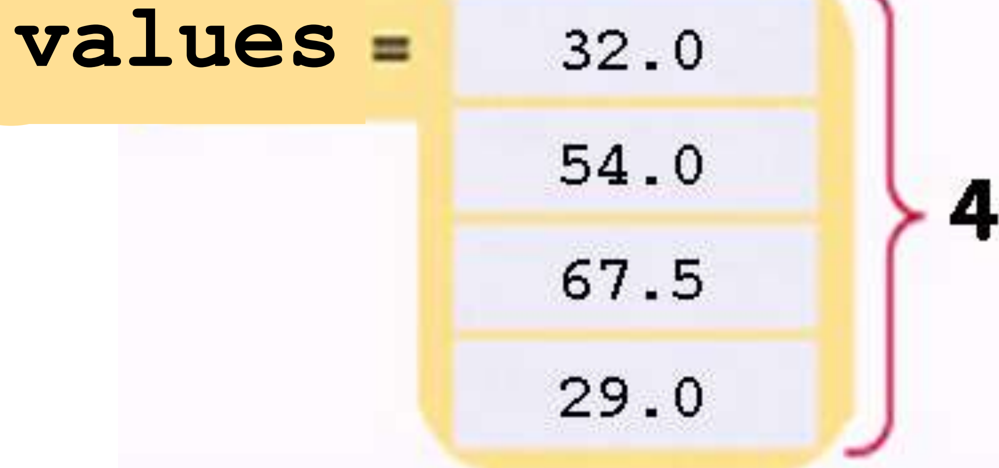
Removing the Last Element with pop_back

values = 32.0, 54.0, 67.5, 29.0, 65.0 } 5

```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Removing the Last Element with pop_back




```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

65 is no longer in the vector.
It now contains only the elements **32.0**, **54.0**, **67.5** and **29.0**,
and its size is **4**.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back and pop_back

You can use **push_back** to put user input into a vector:

```
double input;
while (cin >> input)
{
   values.push_back(input);
}
```


Copyright © 2012 by John Wiley & Sons. All rights reserved

`push_back` Adds an Element

```
vector<double> values;
```

```
double input;
while (cin >> input)
{
   values.push_back(input);
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back Adds an Element

values
0

```
vector<double> values;

double input;
while (cin >> input)
{
    values.push_back(input);
}
```

We are starting again with an empty vector. Its size is 0.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back Adds an Element

values }0

```
vector<double> values;

double input;
while (cin >> input)
{
   values.push_back(input);
}
```

--- The user types 32

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

`push_back` Adds an Element

values
0

```
vector<double> values;

double input;
while (cin >> input)
{
    values.push_back(input);
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back Adds an Element

```
vector<double> values;

double input;
while (cin >> input)
{
    values.push_back(input);
}
```

32 is placed into the vector.
Its size is now 1.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back Adds an Element

values = 32.0 } 1

```
vector<double> values;

double input;
while (cin >> input)
{
    values.push_back(input);
}
```

--- The user types **54**

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

`push_back` Adds an Element

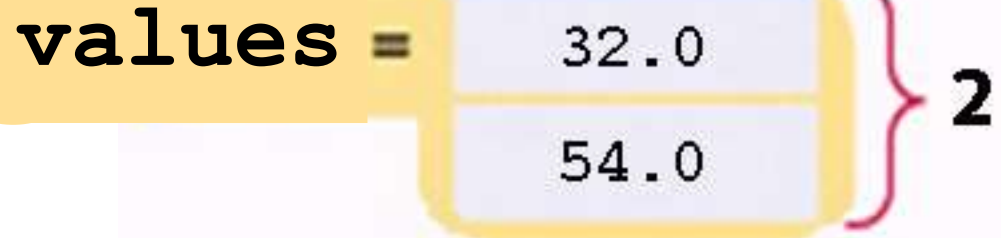
values = 32.0 | 54.0 } 2

```
vector<double> values;

double input;
while (cin >> input)
{
    values.push_back(input);
}
```

`54` is placed into the vector.
Its size is now 2.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back Adds an Element




```
vector<double> values;

double input;
while (cin >> input)
{
   values.push_back(input);
}
```

--- The user types 67.5

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

`push_back` Adds an Element

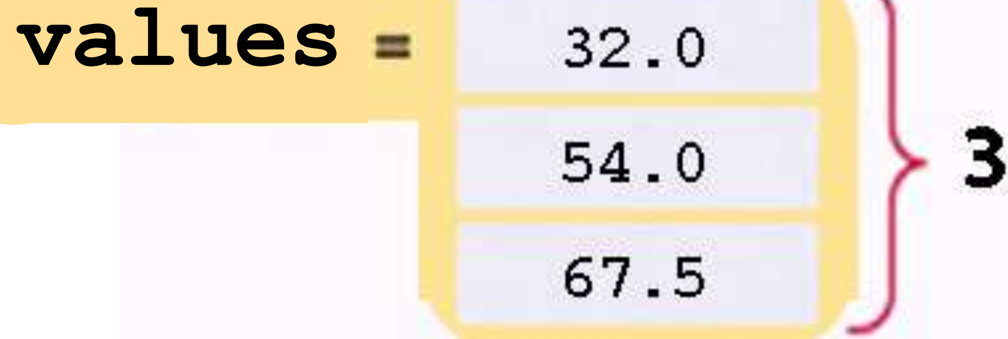
values = 32.0 | 54.0 | 67.5 } 3

```
vector<double> values;

double input;
while (cin >> input)
{
    values.push_back(input);
}
```

`67.4` is placed into the vector.
Its size is now 3.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back Adds an Element




```
vector<double> values;

double input;
while (cin >> input)
{
    values.push_back(input);
}
```

--- The user types 29

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

push_back Adds an Element


```
vector<double> values;

double input;
while (cin >> input)
{
    values.push_back(input);
}
```

29 is placed into the vector.
Its size is now 4.

Copyright © 2012 by John Wiley & Sons. All rights reserved

Using Vectors – `size_of`

How do you visit every element in an vector?

Recall arrays.

Copyright © 2012 by John Wiley & Sons. All rights reserved

Using Vectors – `size_of`

With arrays, to display every element, it would be:

?

```
for (int i = 0; i < 10; i++)
{
   cout << values[i] << endl;
}
```

But with vectors, we don't know about that **10**!

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Using Vectors – `size_of`

Vectors have the **`size`** member function which returns the current size of a vector.

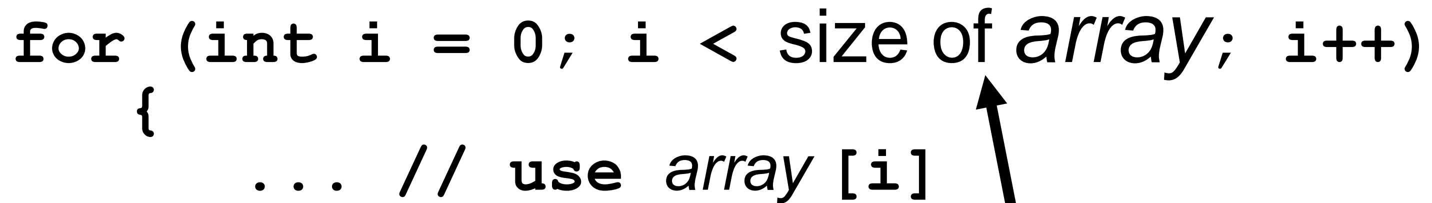
The vector always knows how many are in it and you can always ask it to give you that quantity by calling the **`size`** method:

```
for (int i = 0; i < values.size(); i++)
{
   cout << values[i] << endl;
}
```


Copyright © 2012 by John Wiley & Sons. All rights reserved

Using Vectors – `size_of`

Recall all those array algorithms you learned?



To make them work with vectors, you still use a **`for`** statement,

but instead of looping until size of *array*,

```
for (int i = 0; i < vector.size(); i++)
   {
      ... // use vector[i]
```


Copyright © 2012 by John Wiley & Sons. All rights reserved

Vectors As Parameters In Functions

You know that

functions

are the way to go for code reuse
and solving sub-problems
and many other good things…

so…

Copyright © 2012 by John Wiley & Sons. All rights reserved

Vectors As Parameters In Functions

How can you pass vectors as parameters?

You use vectors as function parameters in exactly the same way as any parameters.

Copyright © 2012 by John Wiley & Sons. All rights reserved

Vectors Parameters – Without Changing the Values

For example, the following function computes the sum of a vector of floating-point numbers:

```
double sum(vector<double> values)
{
    double total = 0;
    for (int i = 0; i < values.size(); i++)
    {
        total = total + values[i];
    }
    return total;
}
```

This function *visits* the vector elements, but it does *not change* them.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Vectors Parameters – Changing the Values

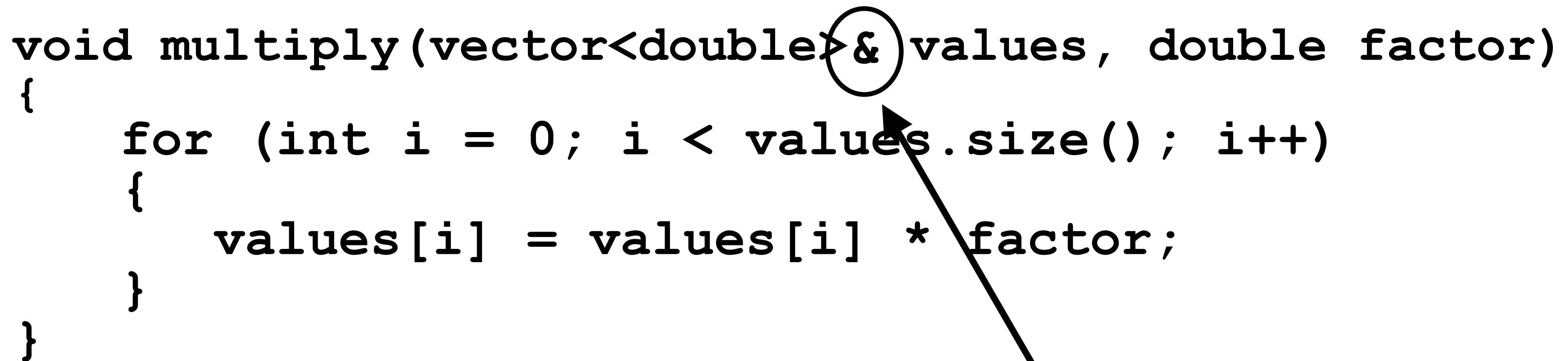
Sometimes the function *should change* the values stored in the vector:

```
void multiply(vector<double>& values, double factor)
{
   for (int i = 0; i < values.size(); i++)
   {
      values[i] = values[i] * factor;
   }
}
```


Copyright © 2012 by John Wiley & Sons. All rights reserved

Vectors Parameters – Changing the Values

Sometimes the function *should change* the values stored in the vector:



```
void multiply(vector<double>& values, double factor)
{
    for (int i = 0; i < values.size(); i++)
    {
        values[i] = values[i] * factor;
    }
}
```

Note that the vector is passed *by reference*, just like any other parameter you want to change.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Vectors Returned from Functions

Sometimes the function should ***return*** a vector.

Vectors are no different from any other values in this regard.

Simply build up the result in the function and return it:

```
vector<int> squares(int n)
{
    vector<int> result;
    for (int i = 0; i < n; i++)
    {
        result.push_back(i * i);
    }
    return result;
}
```

The function returns the squares from 0^2 up to $(n - 1)^2$ by returning a vector.

Copyright © 2012 by John Wiley & Sons. All rights reserved

Vectors and Arrays as Parameters in Functions

Vectors as parameters are easy.

Arrays are not *quite* so easy.

(vectors… vectors…)

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Algorithms – Copying, Arrays Cannot Be Assigned

Suppose you have two arrays

```
int squares[5] = { 0, 1, 4, 9, 16 };
int lucky_numbers[5];
```

The following assignment is an error:

```
lucky_numbers = squares; // Error
```

You must use a loop to copy all elements:

```
for (int i = 0; i < 5; i++)
{
    lucky_numbers[i] = squares[i];
}
```


Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Algorithms – Copying, Vectors Can Be Assigned

Vectors do not suffer from this limitation.

Consider this example:

```
vector<int> squares;
for (int i = 0; i < 5; i++)
{
   squares.push_back(i * i);
}
vector<int> lucky_numbers;
            // Initially empty
lucky_numbers = squares;
         // Now lucky_numbers contains
         // the same elements as squares
```


Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Algorithms – Copying, Vectors Can Be Assigned

You can assign a vector to another vector.

Of course they have to hold the same *type* to do this.

```
vector<int> squares;
for (int i = 0; i < 5; i++)
{
   squares.push_back(i * i);
}
vector<int> lucky_numbers;
                // Initially empty
lucky_numbers = squares;
        // Now lucky_numbers contains
        // the same elements as squares
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Algorithms – Finding Matches

Suppose we want all the values in a vector that are greater than a certain value, say 100, in a vector.

Store them in another vector:

```
vector<double> matches;
for (int i = 0; i < values.size(); i++)
{
   if (values[i] > 100)
   {
      matches.push_back(values[i]);
   }
}
```


Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Algorithms – Removing an Element, Unordered

If you know the position of an element you want to remove from a vector in which the elements are not in any order, as you did in an array,

overwrite the element at that position with the last element in the vector,

then be sure to remove the last element, which also makes the vector smaller.

```
int last_pos = values.size() - 1;
   // Take the position of the last element
values[pos] = values[last_pos];
   // Replace element at pos with last element
values.pop_back();
   // Delete last element to make vector
   // one smaller
```


Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Algorithms – Removing an Element, Ordered

If you know the position of an element you want to remove from a vector in which the elements *are* in some order, as you did in an array,

move all the elements after that position,

then remove the last element to reduce the size.

```
for (int i = pos + 1; i < values.size(); i++)
{
   values[i - 1] = values[i];
}
data.pop_back();
```


Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Algorithms – Inserting an Element, Unordered

When you need to insert an element into a vector whose elements are not in any order…

…oh, this is going to be so easy:

```
values.push_back(new_element);
```


Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Algorithms – Inserting an Element, Ordered

However when the elements in a vector are in some order, it's a bit more complicated, just like it was in the array version.

Of course you must know the position, say **`pos`**, where you will insert the new element.

As in the array version,
you need to move all the elements "up".

```
for (int i = last_pos; i > pos; i--)
{
    values[i] = values[i - 1];
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Algorithms – Inserting an Element, Ordered

You can't do that!

In a vector you cannot assign to the position after the last one!

You cannot assign to any position bigger than

`values() - 1`.

OH DEAR!!!

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Algorithms – Inserting an Element, Ordered

Somehow you need to make
the vector one bigger

before you do the moving.

Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Algorithms – Inserting an Element, Ordered

Be clever.

If you **`push_back`** the last element:

```
int last_pos = values.size() - 1;
values.push_back(values[last_pos]);
```

…*but, but*…

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Algorithms – Inserting an Element, Ordered

Yes, it will be in the vector twice,

but why care?

```
int last_pos = values.size() - 1;
values.push_back(values[last_pos]);
```

You will overwrite it by doing the moving.

Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Algorithms – Inserting an Element, Ordered

And, more importantly,

the vector is now one larger after the **`push_back`**.

Congratulations, it's to safe go ahead and start moving.

```
int last_pos = values.size() - 1;
values.push_back(values[last_pos]);
for (int i = last_pos; i > pos; i--)
{
   values[i] = values[i - 1];
}
values[pos] = new_element;
```

And don't forget to insert the new element.

That's what you've been trying to do all along!

Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Algorithms – Inserting an Element, Ordered

Ah.

```
int last_pos = values.size() - 1;
values.push_back(values[last_pos]);
for (int i = last_pos; i > pos; i--)
{
   values[i] = values[i - 1];
}
values[pos] = new_element;
```

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Common Algorithms – Inserting an Element, Ordered

But don't be too clever,

if the position to insert the new element
is after the last element…

…oh, this is going to be so easy,
don't do any moving, just put it there:

```
values.push_back(new_element);
```

Inserting into an ordered vector means
inserting into the *middle* of the vector!

Copyright © 2012 by John Wiley & Sons. All rights reserved

Sorting with the C++ Library

Recall that you call the **`sort`** function to do your sorting for you.

This can be used on vectors also.

The syntax for vectors is even more unusual than arrays:

```
sort(values.begin(), values.end());
```

Go ahead and use it as you like.
But don't forget to **`#include <algorithm>`**

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Arrays or Vectors? That Is the Question

Should you use arrays or vectors?

(you know you want to say vectors…)

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

Arrays or Vectors? That Is the Question

For most programming tasks,
vectors are easier to use than arrays.

(say vectors, say vectors…)

Copyright © 2012 by John Wiley & Sons. All rights reserved

Arrays or Vectors? That Is the Question

Vectors can grow and shrink.

(grow, shrink - *think:* vectors…)

Copyright © 2012 by John Wiley & Sons. All rights reserved

Arrays or Vectors? That Is the Question

Even if a vector always stays the same size,
it is convenient that a vector remembers its size.

No chance of missing auxiliaries.

Vectors are smarter then arrays!

(size matters and vectors know their own - vectors…)

Copyright © 2012 by John Wiley & Sons. All rights reserved

Arrays or Vectors? That Is the Question

For a beginner, the sole advantage of an array is the initialization syntax.

(syntax, shmyntax – it's easy too with vectors…)

Copyright © 2012 by John Wiley & Sons. All rights reserved

Arrays or Vectors? That Is the Question

Advanced programmers sometimes prefer arrays because they are a bit more efficient.

Moreover, you need to know how to use arrays if you work with older programs

(only a bit? and *olde*r? why not be current by using vectors…)

Copyright © 2012 by John Wiley & Sons. All rights reserved

Prefer Vectors over Arrays

So:

Prefer Vectors over Arrays

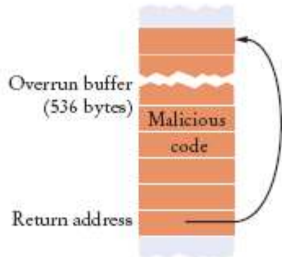
(it's so nice when the moral of the story is: vectors**!!!**)

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

CHAPTER SUMMARY

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.

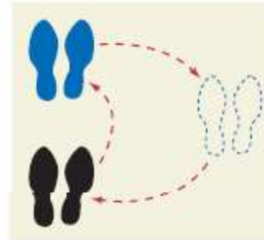


Copyright © 2012 by John Wiley & Sons. All rights reserved

CHAPTER SUMMARY

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.



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.

Copyright © 2012 by John Wiley & Sons. All rights reserved

CHAPTER SUMMARY

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

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

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.

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved

CHAPTER SUMMARY

Use vectors for managing collections whose size can change.

- A vector stores a sequence of values whose size can change.
- Use the `size` member function to obtain the current size of a vector.
- Use the `push_back` member function to add more elements to a vector. Use `pop_back` to reduce the size.
- Vectors can occur as function arguments and return values.
- Use a reference parameter to modify the contents of a vector.
- A function can return a vector.

Copyright © 2012 by John Wiley & Sons. All rights reserved

End Arrays and Vectors II

C++ for Everyone by Cay Horstmann
Copyright © 2012 by John Wiley & Sons. All rights reserved