Common Algorithms – Who Is the Shortest?

Who's the shortest in the line?

Common Algorithms –Minimum

For the minimum, we just reverse the comparison.

```
double smallest = values[0];
for (int i = 1; i < size of values; i++)
{
    if (values[i] < smallest)
    {
        smallest = values[i];
    }
}
```

These algorithms require that the array contain at least one element.

Common Algorithms – Element Separators

When you display the elements of a vector, you usually want to separate them, often with commas or vertical lines, like this:

```
1 | 4 | 9 | 16 | 25
```

Note that there is one fewer separator than there are numbers.

To print five elements,
you need *four* separators.

Common Algorithms – Element Separators

Print the separator before each element *except the initial one* (with index 0):

index
0

```
1 | 4 | 9 | 16 | 25
```

```
for (int i = 0; i < size of values; i++)
{
    if (i > 0)        or if (i>=1)
    {
        cout << " | ";
    }
    cout << values[i];
}
```

Common Algorithms – Linear Search

looking through an array for a specific value

Find the position of a certain value, say 100, in an array:

and want to know position that value is in

```
int pos = 0;
bool found = false;
while (pos < size of values && !found)
{
    if (values[pos] == 100) // looking for 100
    {
        found = true;
    }
    else
    {
        pos++;
    }
}
```

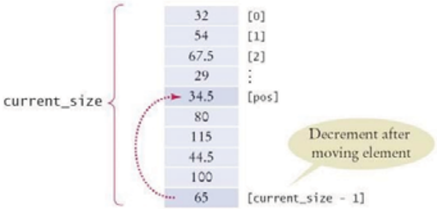
Don't get these tests in the wrong order!

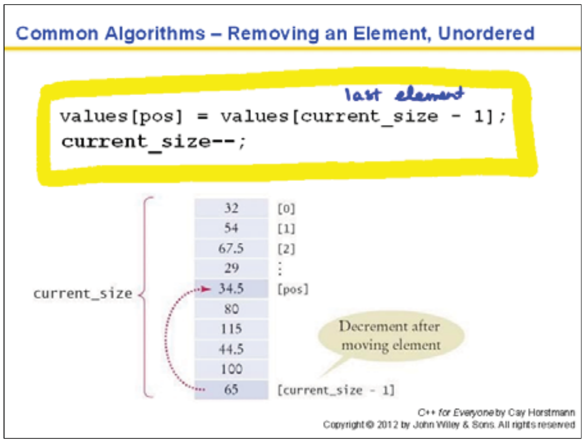
Common Algorithms – Removing an Element, Unordered

Suppose you want to remove the element at index `i`.

If the elements in the vector are not in any particular order, that task is easy to accomplish.

Simply overwrite the element to be removed with the *last* element of the vector, then shrink the size of the vector by removing the value that was copied.



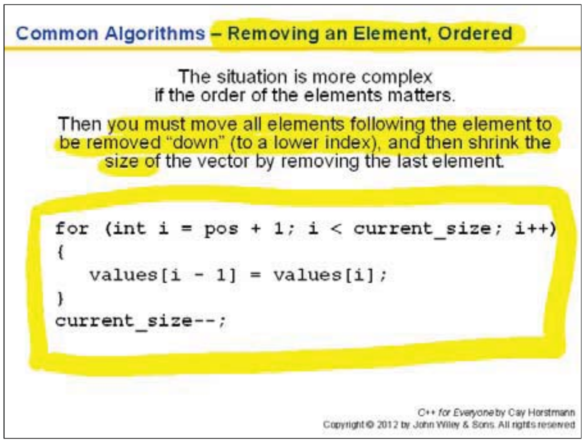


Common Algorithms – Removing an Element, Unordered

last element

```
values[pos] = values[current_size - 1];
current_size--;
```

C++ for Everyone by Cay Horstmann



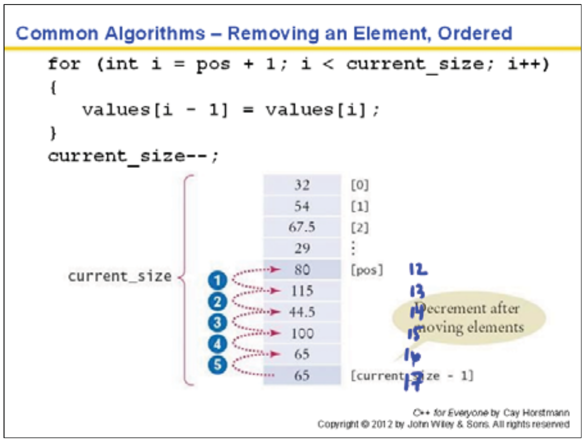
Common Algorithms – Removing an Element, Ordered

The situation is more complex if the order of the elements matters.

Then you must move all elements following the element to be removed "down" (to a lower index), and then shrink the size of the vector by removing the last element.

```
for (int i = pos + 1; i < current_size; i++)
{
   values[i - 1] = values[i];
}
current_size--;
```

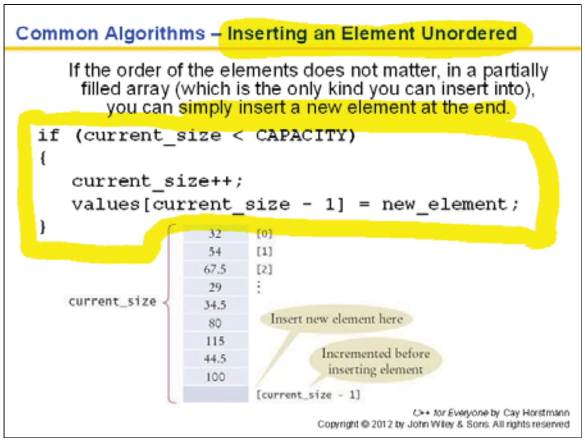
C++ for Everyone by Cay Horstmann



Common Algorithms – Removing an Element, Ordered

```
for (int i = pos + 1; i < current_size; i++)
{
   values[i - 1] = values[i];
}
current_size--;
```

C++ for Everyone by Cay Horstmann

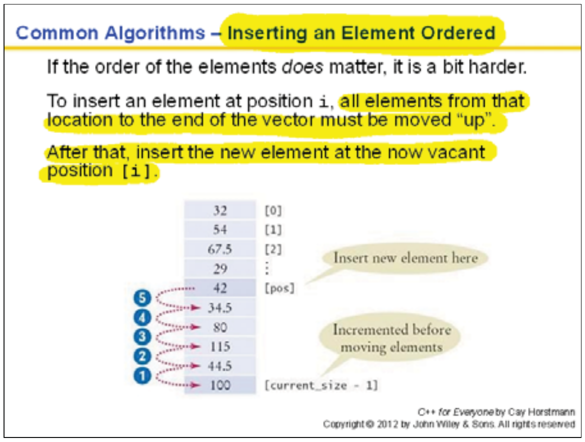


Common Algorithms – Inserting an Element Unordered

If the order of the elements does not matter, in a partially filled array (which is the only kind you can insert into), you can simply insert a new element at the end.

```
if (current_size < CAPACITY)
{
   current_size++;
   values[current_size - 1] = new_element;
}
```

C++ for Everyone by Cay Horstmann



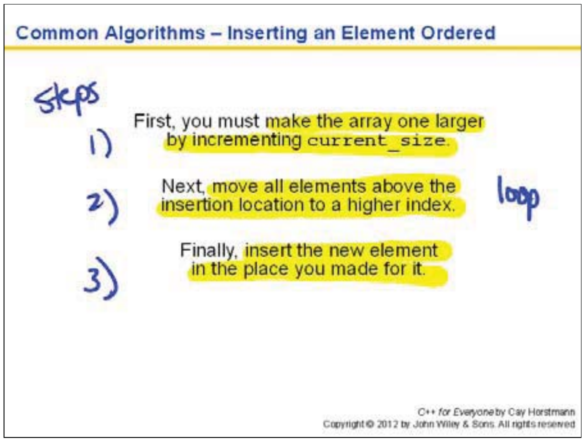
Common Algorithms – Inserting an Element Ordered

If the order of the elements *does* matter, it is a bit harder.

To insert an element at position `i`, all elements from that location to the end of the vector must be moved "up".

After that, insert the new element at the now vacant position `[i]`.

C++ for Everyone by Cay Horstmann



Common Algorithms – Inserting an Element Ordered

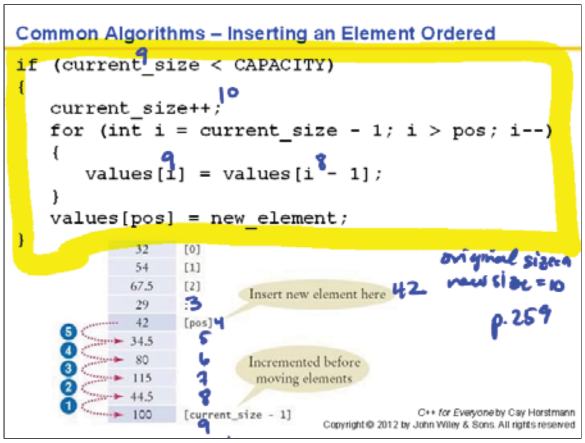
steps

1) First, you must make the array one larger by incrementing `current_size`.

2) Next, move all elements above the insertion location to a higher index. loop

3) Finally, insert the new element in the place you made for it.

C++ for Everyone by Cay Horstmann



Common Algorithms – Inserting an Element Ordered

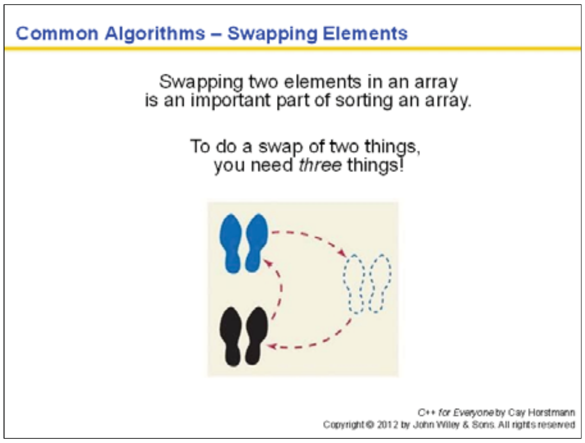
```
if (current_size < CAPACITY)
{
   current_size++;
   for (int i = current_size - 1; i > pos; i--)
   {
      values[i] = values[i - 1];
   }
   values[pos] = new_element;
}
```

9 10 9 8

32 [0]
54 [1]
67.5 [2]
29 3
42 [pos] 4
34.5 5
80 6
115 7
44.5 8
100 [current_size - 1] 9
5
4
3
2
1
Insert new element here 42
Incremented before moving elements

original size=9
new size=10
p.259

C++ for Everyone by Cay Horstmann

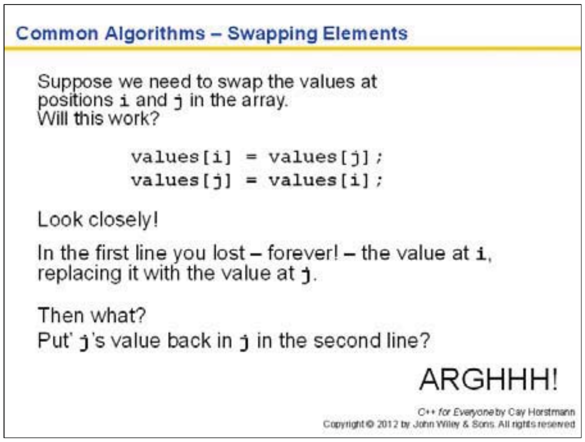


Common Algorithms – Swapping Elements

Swapping two elements in an array is an important part of sorting an array.

To do a swap of two things, you need *three* things!

C++ for Everyone by Cay Horstmann



Common Algorithms – Swapping Elements

Suppose we need to swap the values at positions `i` and `j` in the array.
Will this work?

```
values[i] = values[j];
values[j] = values[i];
```

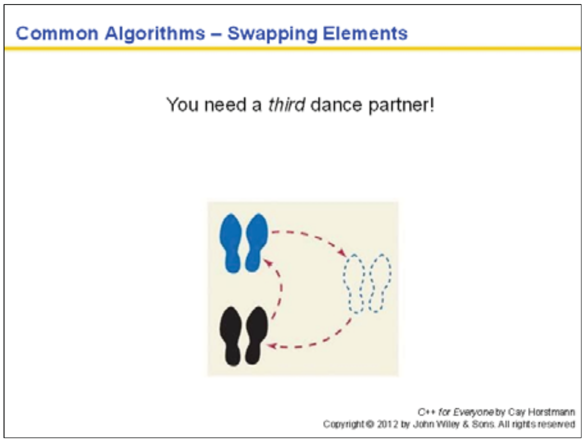
Look closely!

In the first line you lost – forever! – the value at `i`, replacing it with the value at `j`.

Then what?
Put' `j`'s value back in `j` in the second line?

ARGHHH!

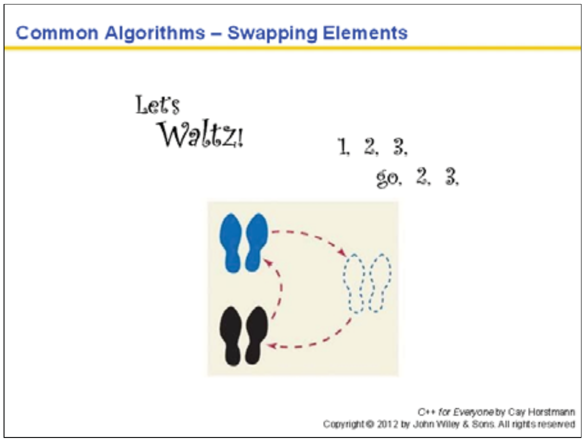
C++ for Everyone by Cay Horstmann



Common Algorithms – Swapping Elements

You need a *third* dance partner!

C++ for Everyone by Cay Horstmann

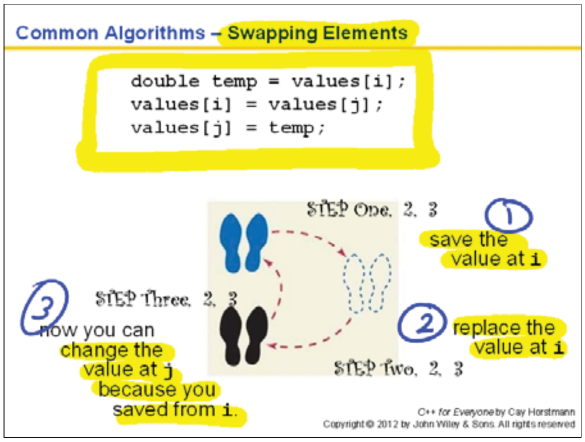


Common Algorithms – Swapping Elements

Let's Waltz!

1, 2, 3,
go, 2, 3,

C++ for Everyone by Cay Horstmann



Common Algorithms – Swapping Elements

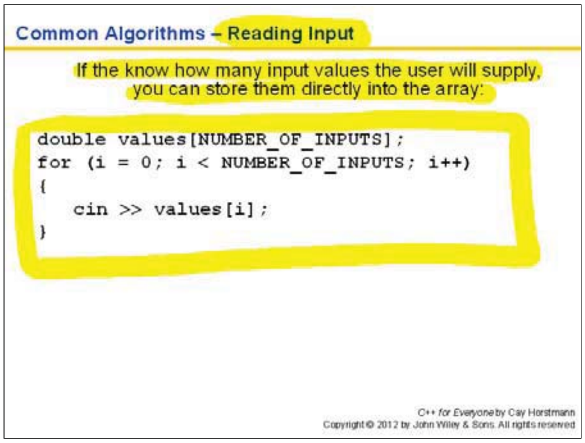
```
double temp = values[i];
values[i] = values[j];
values[j] = temp;
```

STEP One, 2, 3
1 save the value at `i`

STEP Two, 2, 3
2 replace the value at `i`

STEP Three, 2, 3
3 now you can change the value at `j` because you saved from `i`.

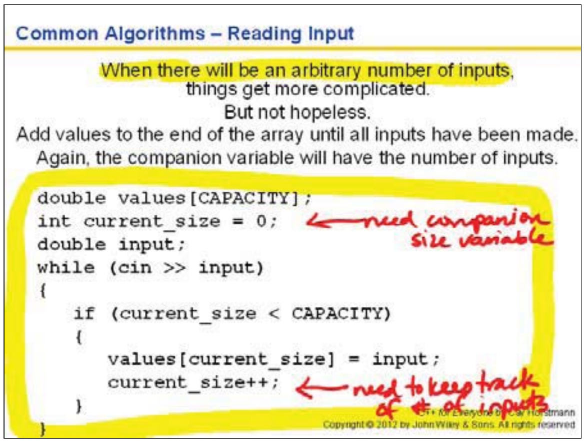
C++ for Everyone by Cay Horstmann



Common Algorithms – Reading Input

If the know how many input values the user will supply, you can store them directly into the array:

```
double values[NUMBER_OF_INPUTS];
for (i = 0; i < NUMBER_OF_INPUTS; i++)
{
   cin >> values[i];
}
```



Common Algorithms – Reading Input

When there will be an arbitrary number of inputs, things get more complicated.

But not hopeless.

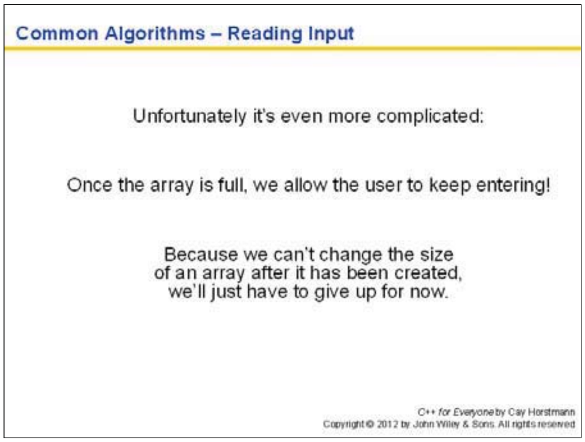
Add values to the end of the array until all inputs have been made.

Again, the companion variable will have the number of inputs.

```
double values[CAPACITY];
int current_size = 0;
double input;
while (cin >> input)
{
   if (current_size < CAPACITY)
   {
      values[current_size] = input;
      current_size++;
   }
}
```

← need companion size variable

← need to keep track of # of inputs

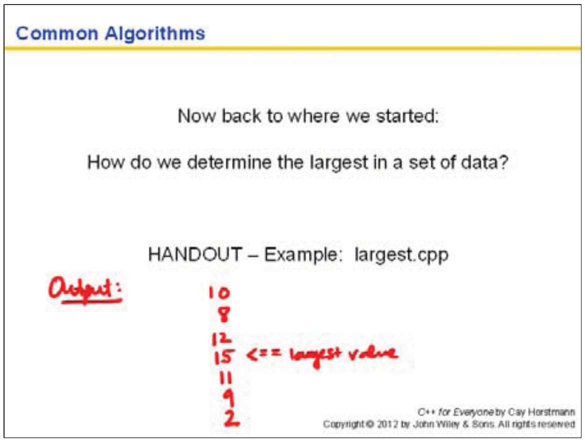


Common Algorithms – Reading Input

Unfortunately it's even more complicated:

Once the array is full, we allow the user to keep entering!

Because we can't change the size of an array after it has been created, we'll just have to give up for now.



Common Algorithms

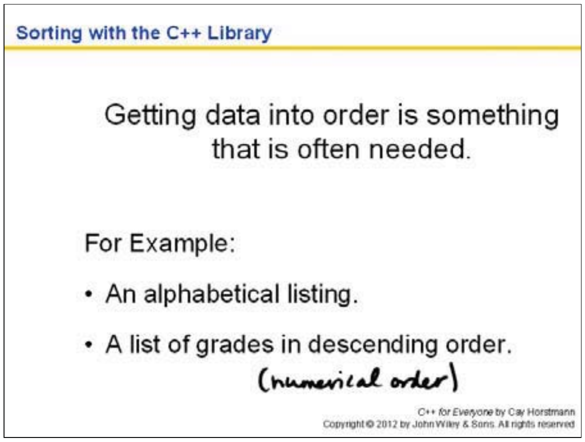
Now back to where we started:

How do we determine the largest in a set of data?

HANDOUT – Example: largest.cpp

Output:

```
10
8
12
15 <== largest value
11
9
2
```

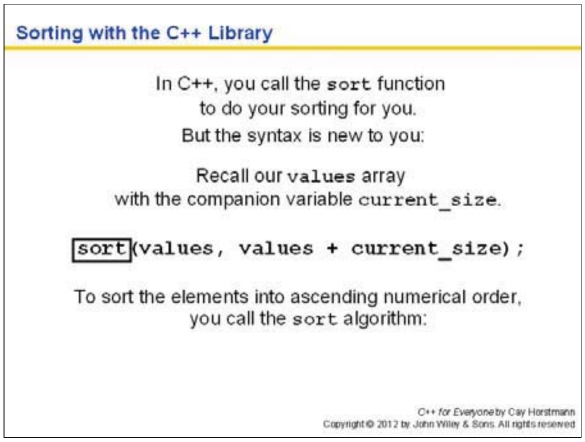


Sorting with the C++ Library

Getting data into order is something that is often needed.

For Example:

- An alphabetical listing.
- A list of grades in descending order. (numerical order)



Sorting with the C++ Library

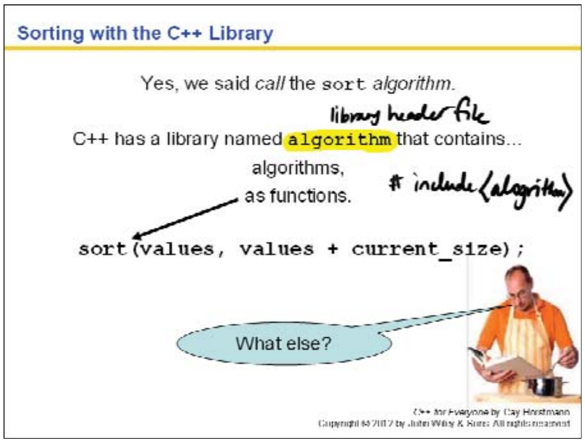
In C++, you call the `sort` function to do your sorting for you.

But the syntax is new to you:

Recall our `values` array with the companion variable `current_size`.

```
sort(values, values + current_size);
```

To sort the elements into ascending numerical order, you call the `sort` algorithm:



Sorting with the C++ Library

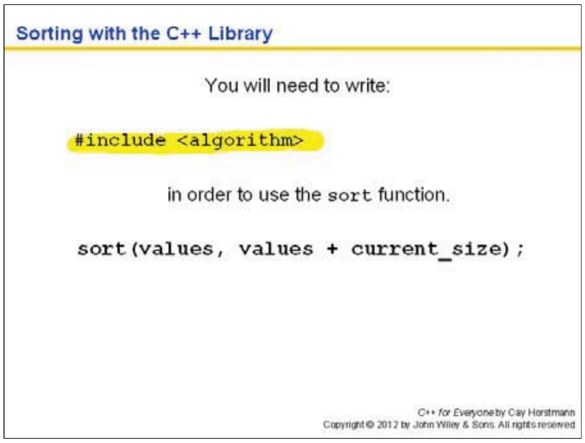
Yes, we said *call* the `sort` *algorithm*.

library header file

C++ has a library named `algorithm` that contains…
algorithms,
as functions.

include <algorithm>

```
sort(values, values + current_size);
```



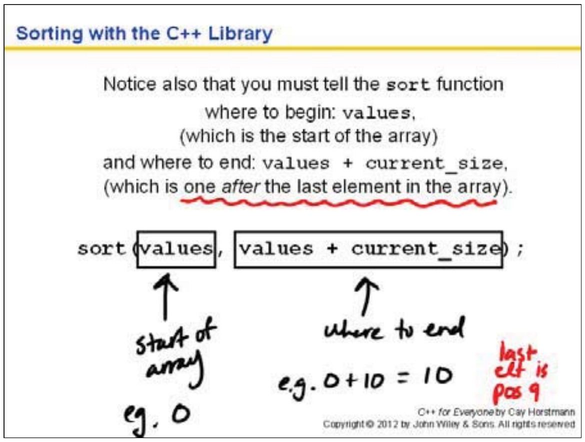
Sorting with the C++ Library

You will need to write:

```
#include <algorithm>
```

in order to use the `sort` function.

```
sort(values, values + current_size);
```



Sorting with the C++ Library

Notice also that you must tell the `sort` function
where to begin: `values`,
(which is the start of the array)
and where to end: `values + current_size`,
(which is one *after* the last element in the array).

```
sort(values, values + current_size);
```

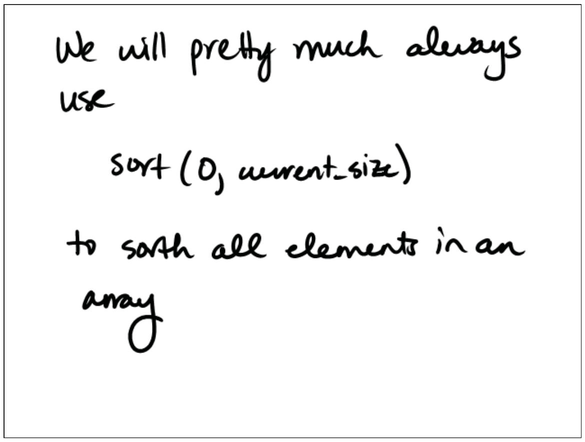
start of array

eg. 0

where to end

e.g. 0+10 = 10

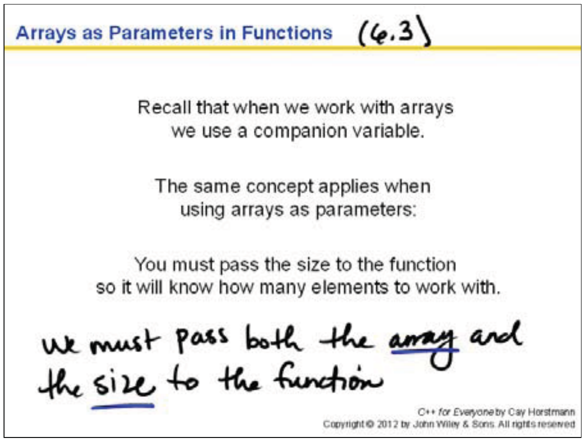
last elt is pos 9



We will pretty much always use

sort (0, current_size)

to sorth all elements in an array



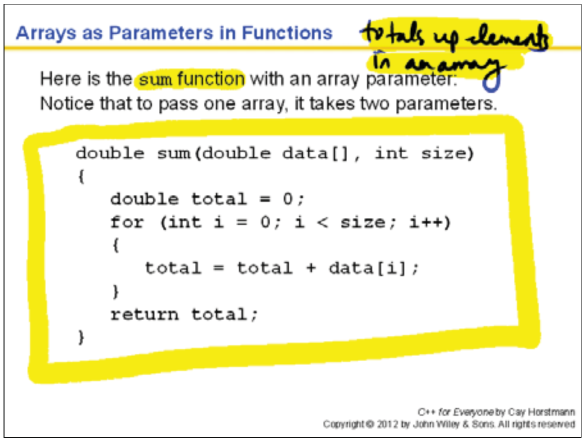
Arrays as Parameters in Functions (6.3)

Recall that when we work with arrays
we use a companion variable.

The same concept applies when
using arrays as parameters:

You must pass the size to the function
so it will know how many elements to work with.

We must pass both the array and the size to the function

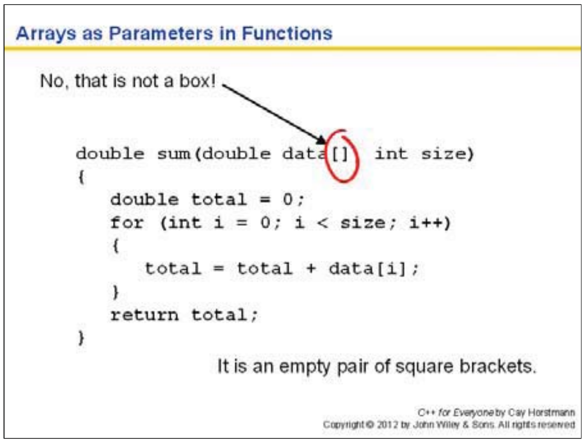


Arrays as Parameters in Functions

totals up elements in an array

Here is the sum function with an array parameter:
Notice that to pass one array, it takes two parameters.

```
double sum(double data[], int size)
{
   double total = 0;
   for (int i = 0; i < size; i++)
   {
      total = total + data[i];
   }
   return total;
}
```

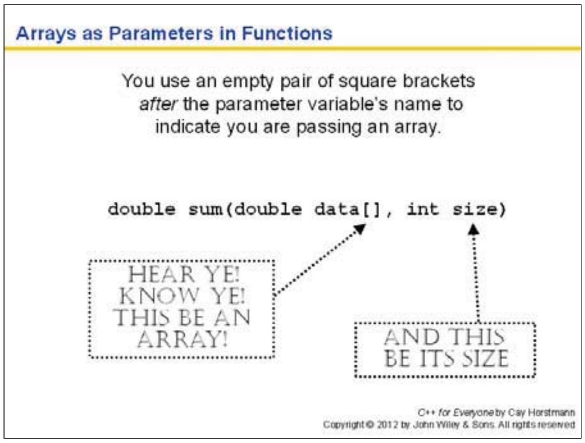


Arrays as Parameters in Functions

No, that is not a box!

```
double sum(double data[], int size)
{
   double total = 0;
   for (int i = 0; i < size; i++)
   {
      total = total + data[i];
   }
   return total;
}
```

It is an empty pair of square brackets.



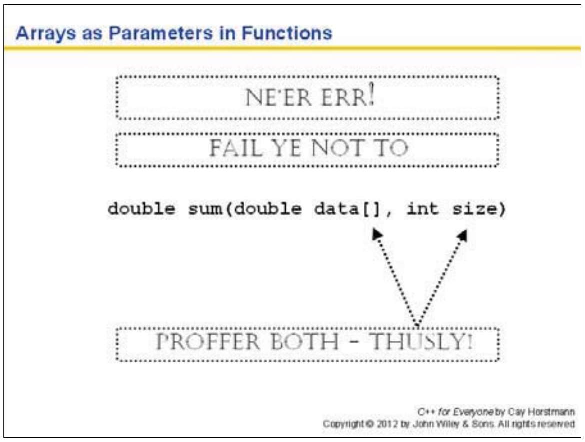
Arrays as Parameters in Functions

You use an empty pair of square brackets *after* the parameter variable's name to indicate you are passing an array.

```
double sum(double data[], int size)
```

HEAR YE! KNOW YE! THIS BE AN ARRAY!

AND THIS BE ITS SIZE



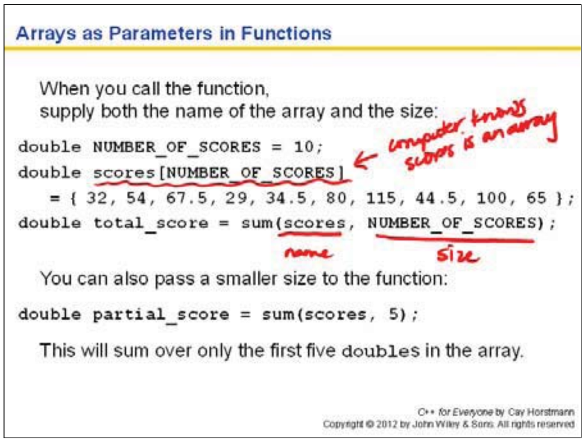
Arrays as Parameters in Functions

NE'ER ERR!

FAIL YE NOT TO

```
double sum(double data[], int size)
```

PROFFER BOTH – THUSLY!



Arrays as Parameters in Functions

When you call the function,
supply both the name of the array and the size:

```
double NUMBER_OF_SCORES = 10;
double scores[NUMBER_OF_SCORES]
   = { 32, 54, 67.5, 29, 34.5, 80, 115, 44.5, 100, 65 };
double total_score = sum(scores, NUMBER_OF_SCORES);
```

← computer knows scores is an array

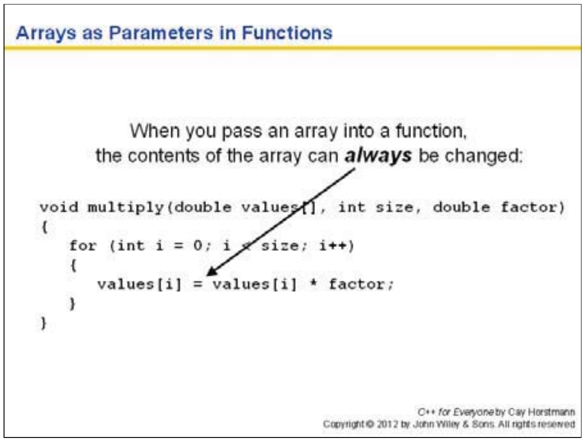
name

size

You can also pass a smaller size to the function:

```
double partial_score = sum(scores, 5);
```

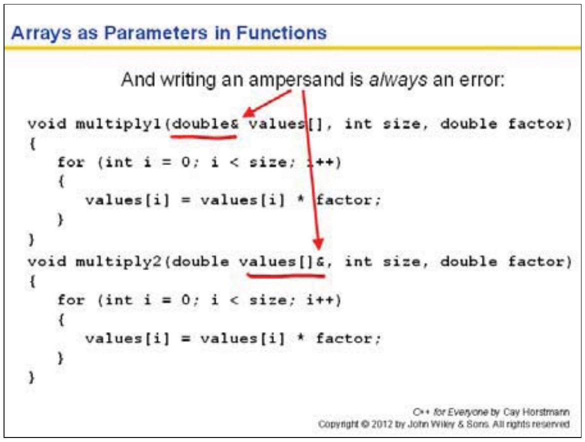
This will sum over only the first five `doubles` in the array.



Arrays as Parameters in Functions

When you pass an array into a function,
the contents of the array can ***always*** be changed:

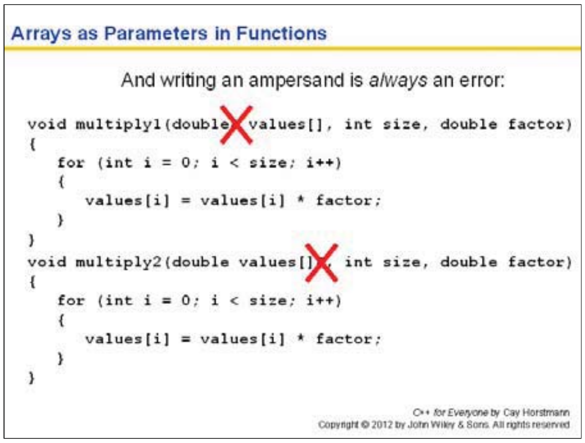
```
void multiply(double values[], int size, double factor)
{
   for (int i = 0; i < size; i++)
   {
      values[i] = values[i] * factor;
   }
}
```



Arrays as Parameters in Functions

And writing an ampersand is *always* an error:

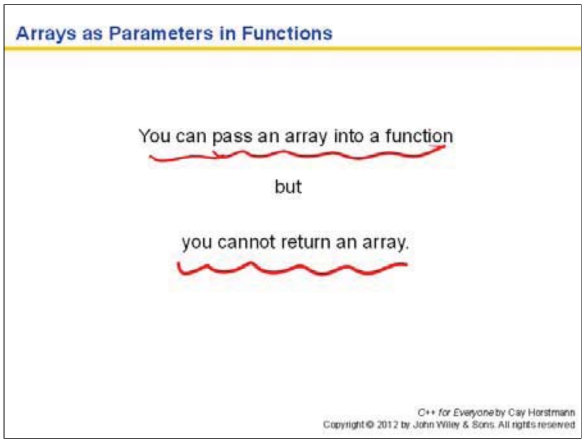
```
void multiply1(double& values[], int size, double factor)
{
   for (int i = 0; i < size; i++)
   {
      values[i] = values[i] * factor;
   }
}
void multiply2(double values[]&, int size, double factor)
{
   for (int i = 0; i < size; i++)
   {
      values[i] = values[i] * factor;
   }
}
```



Arrays as Parameters in Functions

And writing an ampersand is *always* an error:

```
void multiply1(double& values[], int size, double factor)
{
   for (int i = 0; i < size; i++)
   {
      values[i] = values[i] * factor;
   }
}
void multiply2(double values[]&, int size, double factor)
{
   for (int i = 0; i < size; i++)
   {
      values[i] = values[i] * factor;
   }
}
```

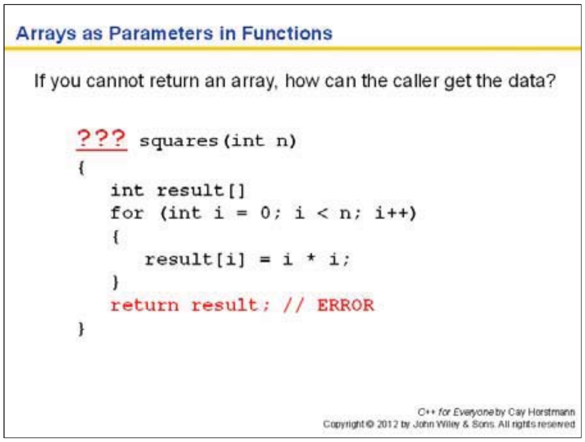


Arrays as Parameters in Functions

You can pass an array into a function

but

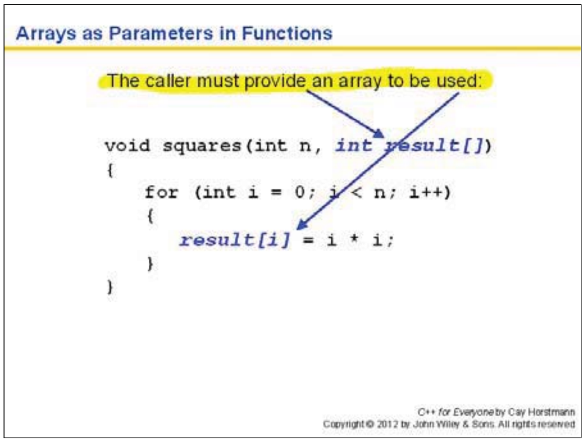
you cannot return an array.



Arrays as Parameters in Functions

If you cannot return an array, how can the caller get the data?

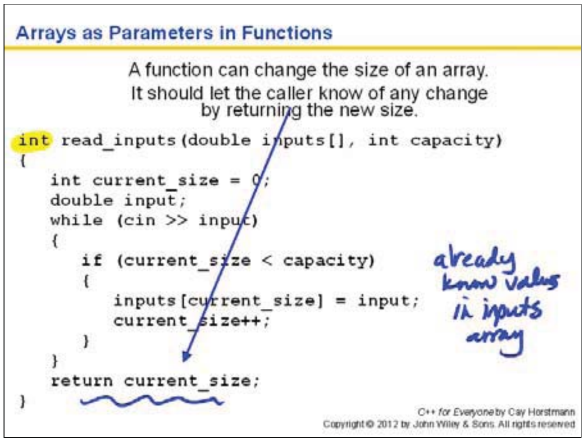
```
??? squares(int n)
{
   int result[]
   for (int i = 0; i < n; i++)
   {
      result[i] = i * i;
   }
   return result; // ERROR
}
```



Arrays as Parameters in Functions

The caller must provide an array to be used:

```
void squares(int n, int result[])
{
   for (int i = 0; i < n; i++)
   {
      result[i] = i * i;
   }
}
```

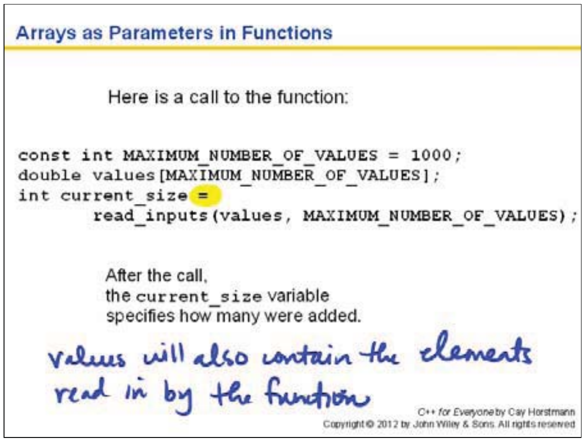


Arrays as Parameters in Functions

A function can change the size of an array.
It should let the caller know of any change
by returning the new size.

```
int read_inputs(double inputs[], int capacity)
{
   int current_size = 0;
   double input;
   while (cin >> input)
   {
      if (current_size < capacity)
      {
         inputs[current_size] = input;
         current_size++;
      }
   }
   return current_size;
}
```

already know valus in inputs array



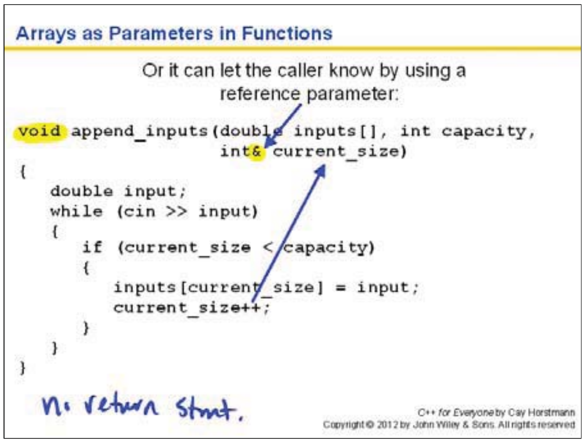
Arrays as Parameters in Functions

Here is a call to the function:

```
const int MAXIMUM_NUMBER_OF_VALUES = 1000;
double values[MAXIMUM_NUMBER_OF_VALUES];
int current_size =
      read_inputs(values, MAXIMUM_NUMBER_OF_VALUES);
```

After the call,
the `current_size` variable
specifies how many were added.

values will also contain the elements read in by the function

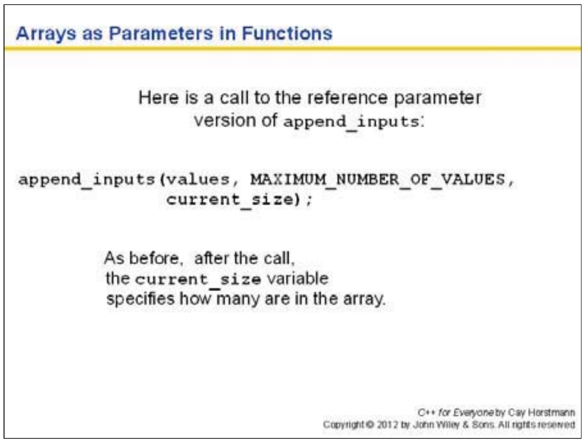


Arrays as Parameters in Functions

Or it can let the caller know by using a reference parameter:

```
void append_inputs(double inputs[], int capacity,
                   int& current_size)
{
   double input;
   while (cin >> input)
   {
      if (current_size < capacity)
      {
         inputs[current_size] = input;
         current_size++;
      }
   }
}
```

no return stmt.

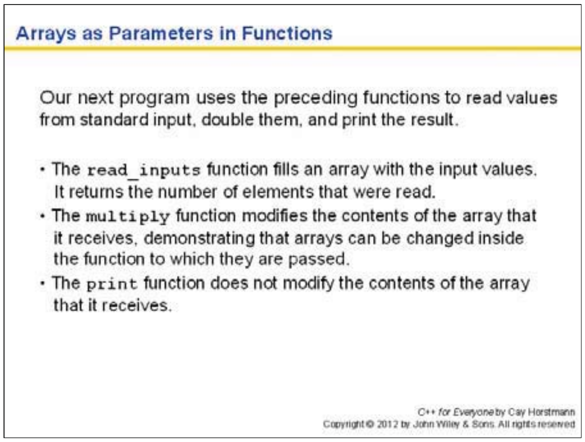


Arrays as Parameters in Functions

Here is a call to the reference parameter version of `append_inputs`:

```
append_inputs(values, MAXIMUM_NUMBER_OF_VALUES,
              current_size);
```

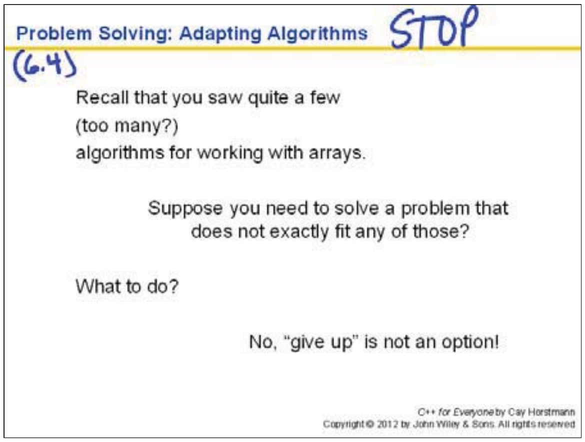
As before, after the call, the `current_size` variable specifies how many are in the array.



Arrays as Parameters in Functions

Our next program uses the preceding functions to read values from standard input, double them, and print the result.

- The `read_inputs` function fills an array with the input values. It returns the number of elements that were read.
- The `multiply` function modifies the contents of the array that it receives, demonstrating that arrays can be changed inside the function to which they are passed.
- The `print` function does not modify the contents of the array that it receives.



Problem Solving: Adapting Algorithms

STOP

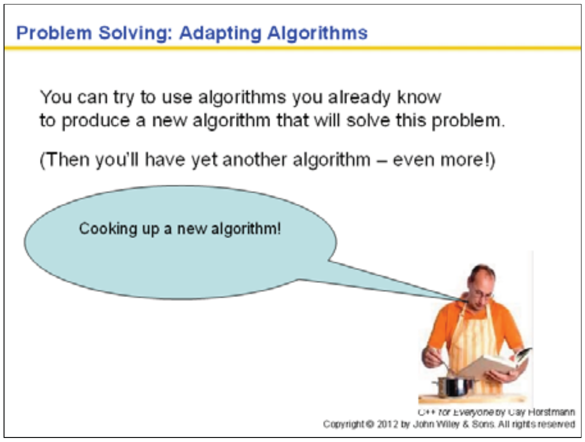
(6.4)

Recall that you saw quite a few
(too many?)
algorithms for working with arrays.

Suppose you need to solve a problem that does not exactly fit any of those?

What to do?

No, "give up" is not an option!

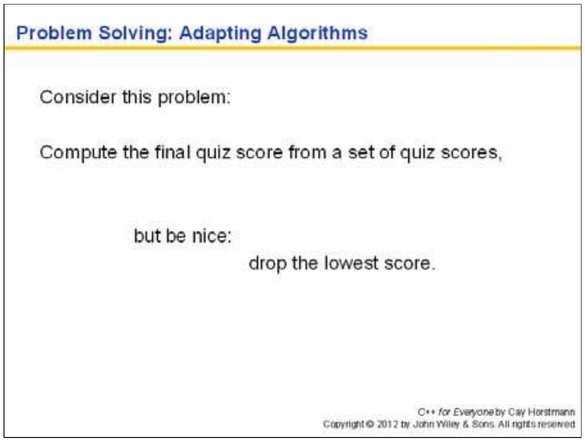


Problem Solving: Adapting Algorithms

You can try to use algorithms you already know to produce a new algorithm that will solve this problem.

(Then you'll have yet another algorithm – even more!)

Cooking up a new algorithm!



Problem Solving: Adapting Algorithms

Consider this problem:

Compute the final quiz score from a set of quiz scores,

but be nice:

drop the lowest score.