

# Computer Science 1: Starting Computing CSCI 1300



University of Colorado  
Boulder

# Computer Science 1: Starting Computing CSCI 1300



Dr. Ioana Fleming  
Spring 2019  
Lecture 17



University of Colorado  
Boulder

# Reminders

## Submissions:

- Homework 5: due 2/23 at 6pm
- Homework 6: due 3/4 at 6pm

## Readings:

- Ch. 6 – Arrays
- Ch. 8 – Streams – next week

## Today:

- Global Variables
- Early Alert of Performance



University of Colorado  
Boulder



© traveler1116/iStockphoto.

# Chapter Six: Arrays and Vectors



University of Colorado  
Boulder

Java C++ by Cay Horstmann  
Copyright © 2018 by John Wiley & Sons. All rights reserved

# Topic 2

1. Arrays
2. Common array algorithms
3. Arrays / functions
4. Problem solving: adapting algorithms
5. Problem solving: discovering algorithms
6. 2D arrays
7. Vectors
8. Chapter Summary



# Common Algorithms – Copying

Consider these two arrays:

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

How can we copy the values from **squares** to  
**lucky\_numbers**?

Let's try what seems right and easy...

```
squares = lucky_numbers; ...and wrong!
```

*You cannot assign arrays!*

*The compiler will report a syntax error.*

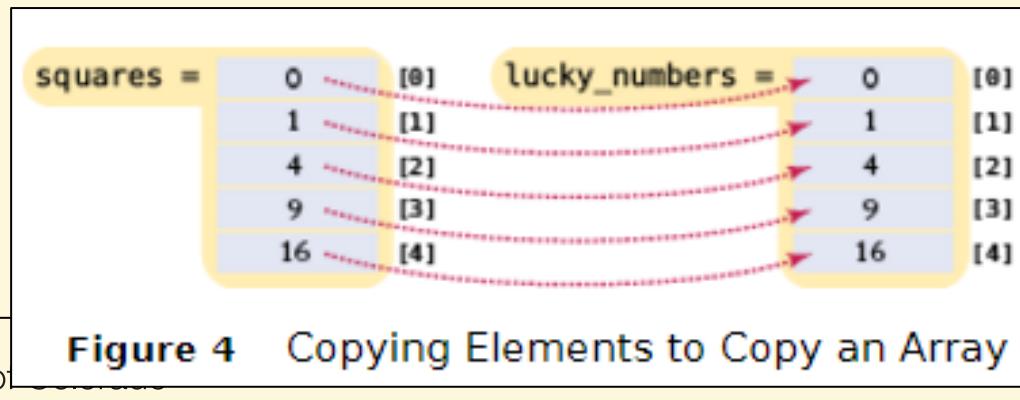


# Common Algorithms – Copying Requires a Loop

```
/* you must copy each element individually using a
loop! */

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

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



**Figure 4** Copying Elements to Copy an Array



# Common Algorithms – Sum and Average Value

You have already seen the algorithm for computing the sum and average of a set of data. The algorithm is the same when the data is stored in an array.

```
double total = 0;  
for (int i = 0; i < size; i++)  
{  
    total = total + values[i];  
}
```

The average is just arithmetic:

```
double average = total / size;
```



# Common Algorithms – Maximum

To compute the largest value in a vector, keep a variable that stores the largest element that you have encountered, and update it when you find a larger one.

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



# Common Algorithms – Minimum

For the minimum, we just reverse the comparison.

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

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



University of Colorado

Boulder

Java C++ by Cay Horstmann

Copyright © 2018 by John Wiley & Sons. All rights reserved

# Common Algorithms – Linear Search

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

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



University of Colorado

Boulder

Java C++ by Cay Horstmann

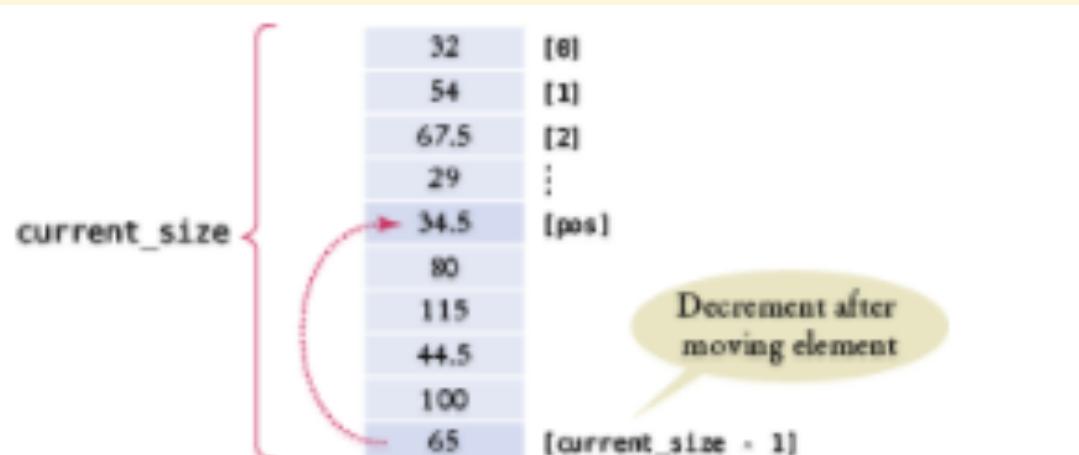
Copyright © 2018 by John Wiley & Sons. All rights reserved

# Common Algorithms – Removing an Element, Unordered

To remove the element at index **i**:

If the elements in the array are not in any particular order, simply overwrite the element to be removed with the *last* element, then shrink the size by 1.

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



University  
Boulder

**Figure 5** Removing an Element in an Unordered Array

# Common Algorithms – Removing an Element, Ordered

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

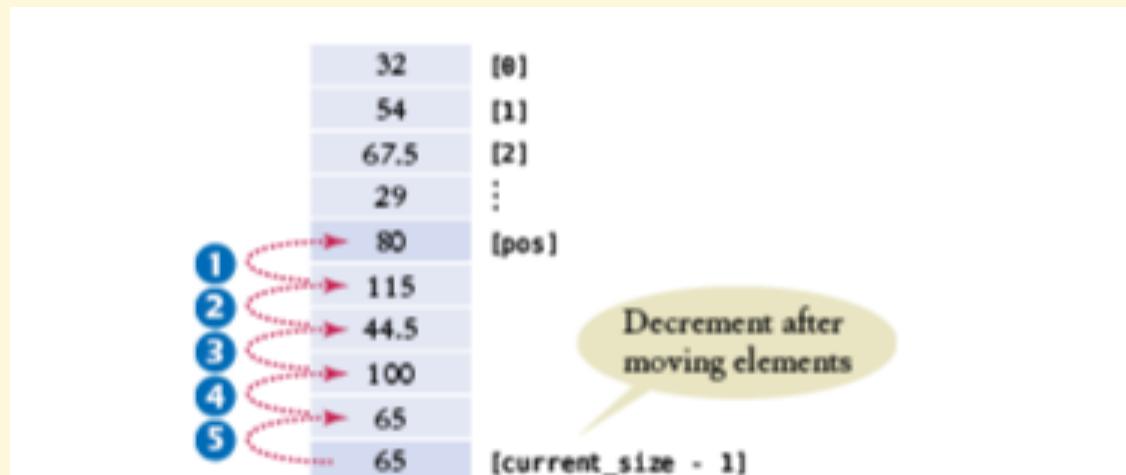
Then you must move all elements following the element to be removed “down” (to a lower index), and then shrink the size of the vector by removing the last element.

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



# Common Algorithms – Removing an Element, Ordered

```
//removing the element at index "pos"
for (int i = pos + 1; i < current_size; i++)
{
    values[i - 1] = values[i];
}
current_size--;
```



**Figure 6** Removing an Element in an Ordered Array



# 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;
}
```



University of Colorado  
Boulder

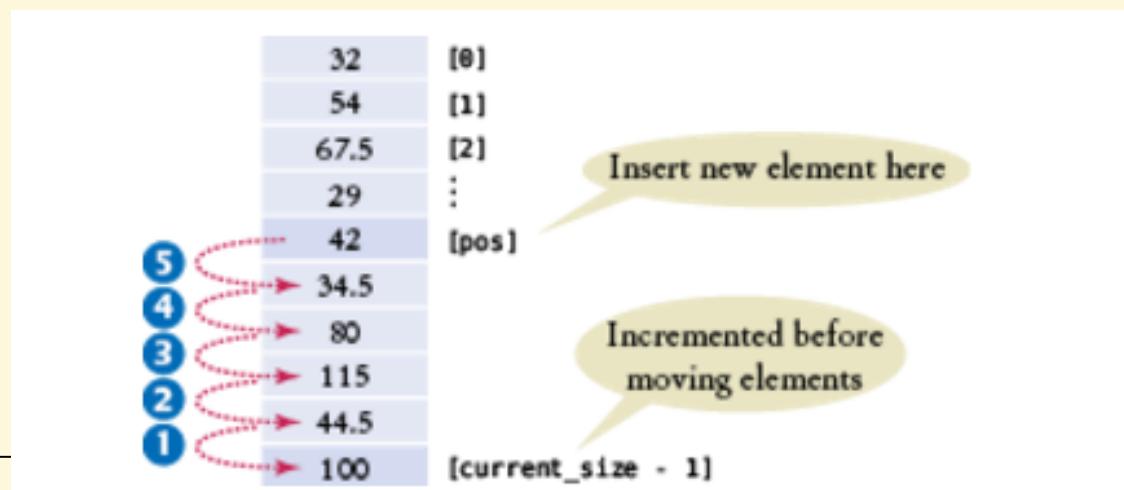
Java C++ by Cay Horstmann  
Copyright © 2018 by John Wiley & Sons. All rights reserved

# 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 “out” to higher indices.

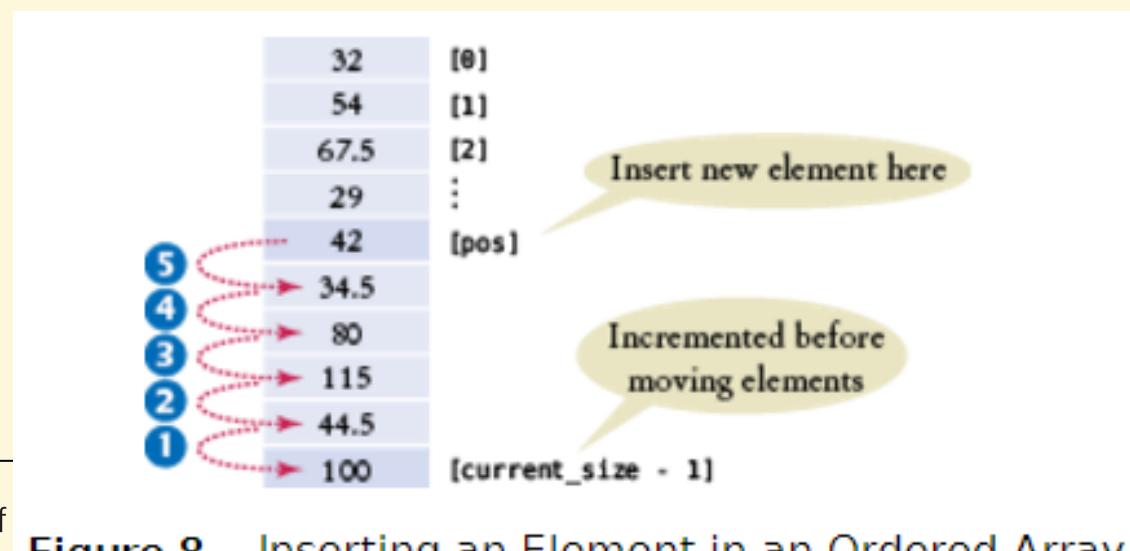
After that, insert the new element at the now vacant position [**i**].



**Figure 8** Inserting an Element in an Ordered Array

# Inserting an Element Ordered: Code

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



University of  
Boulder

**Figure 8** Inserting an Element in an Ordered Array

# 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?

We end up with 2 copies of the  $[j]$  value, and have lost the  $[i]$



University of Colorado

Boulder

Copyright © 2018 by John Wiley & Sons. All rights reserved

# Code for Swapping Array Elements

```
// save the first element in  
// a temporary variable  
// before overwriting the 1st  
  
double temp = values[i];  
values[i] = values[j];  
values[j] = temp;
```

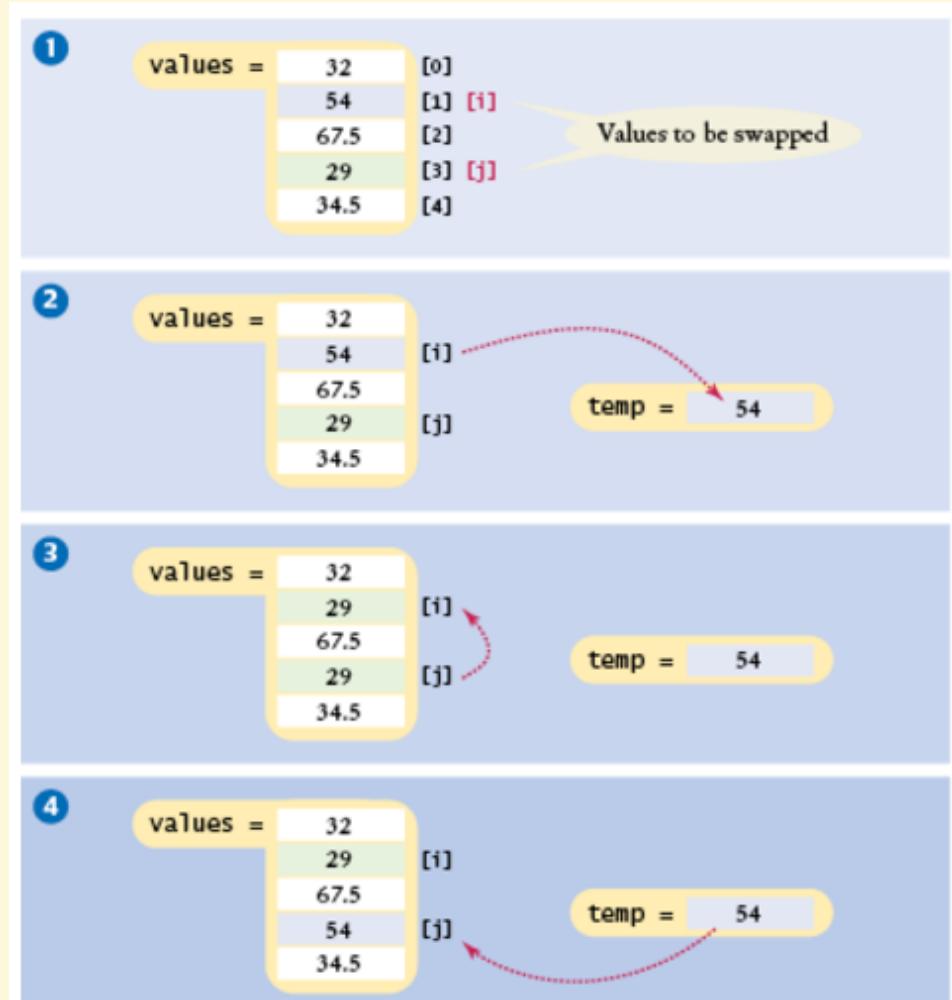


Figure 9 Swapping Array Elements



# Topic 3

1. Arrays
2. Common array algorithms
3. Arrays / functions
4. Problem solving: adapting algorithms
5. Problem solving: discovering algorithms
6. 2D arrays
7. Vectors
8. Chapter Summary



# Arrays as Parameters in Functions

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.



University of Colorado

Java C++ by Cay Horstmann

Copyright © 2018 by John Wiley & Sons. All rights reserved

# Array as function argument

- What does the computer know about an array?
  - The base type
  - The address of the first indexed variable
  - The number of indexed variables
- What does a function know about an array argument?
  - The base type
  - The address of the first indexed variable



# Entire Arrays as Arguments

- Formal parameter can be entire array
  - Argument then passed in function call is array name
  - Called "array parameter"
- Send size of array as well
  - Typically done as second parameter
  - Simple int type formal parameter



# Array Parameter Function Example

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;
}
```



University of Colorado  
Boulder

Java C++ by Cay Horstmann  
Copyright © 2018 by John Wiley & Sons. All rights reserved

# Entire Array as Argument Example

- In some main() function definition,  
consider this calls:

```
int score[5], numberOfScores = 5;  
fillup(score, numberOfScores);
```

- 1<sup>st</sup> argument is entire array
- 2<sup>nd</sup> argument is integer value
- Note no brackets in array argument!



# Array Parameters

- May seem strange
  - No brackets in array argument
  - Must send size separately
- One nice property:
  - Can use SAME function to fill any size array!
  - Exemplifies "re-use" properties of functions
  - Example:

```
int score[5], time[10];  
fillUp(score, 5);  
fillUp(time, 10);
```



# Array as Argument: How?

- What's really passed?
- Think of array as 3 "pieces"
  - Address of first indexed variable (`arrName[0]`)
  - Array base type (`int` or `double` or `float` or ...)
  - Size of array
- Only 1<sup>st</sup> piece is passed!
  - Just the beginning address of array (the 1<sup>st</sup> element)
  - Knowing the type helps us retrieve the (2<sup>nd</sup> – last) elements



# Array Parameters in Functions Require [ ] in the Header

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)
```



University of Colorado  
Boulder

Java C++ by Cay Horstmann  
Copyright © 2018 by John Wiley & Sons. All rights reserved

# Array Function Call Does NOT Use the Brackets!

When you call the function, supply both the name of the array and the size, BUT NO SQUARE BRACKETS!!

```
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);
```

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.



# Array Parameters Always are Reference Parameters

When you pass an array into a function, the contents of the array can **always** be changed. An array name is actually a reference, that is, a memory address:

```
//function to scale all elements in array by a factor
void multiply(double values[], int size, double factor)
{
    for (int i = 0; i < size; i++)
    {
        values[i] = values[i] * factor;
    }
}
```



# Arrays as Parameters but No Array Returns

You can pass an array into a function but  
you cannot return an array.

However, the function can modify an input array, so the function definition must include the result array in the parentheses if one is desired.



University of Colorado

Java C++ by Cay Horstmann

Copyright © 2018 by John Wiley & Sons. All rights reserved

# Arrays as Parameters and Return Value

If a function can change the size of an array,  
it should let the caller know the new size by returning it:

```
int read_inputs(double inputs[], int capacity)
{
    //returns the # of elements read, as int
    int current_size = 0;
    double input;
    while (cin >> input)
    {
        if (current_size < capacity)
        {
            inputs[current_size] = input;
            current_size++;
        }
    }
    return current_size;
}
```



University of Colorado

Boulder

Java C++ by Cay Horstmann

Copyright © 2018 by John Wiley & Sons. All rights reserved

# Array Parameters in Functions: Calling the Function

Here is a call to the `read_inputs` function:

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

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



# Function to Fill or Append to an Array

Or it can let the caller know by passing and returning the current size:

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

*Note this function has the added benefit of either filling an empty array or appending to a partially-filled array.*

---



# Array Functions Example

The following 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.



University of Colorado

Boulder

Copyright © 2018 by John Wiley & Sons. All rights reserved

# Array Functions Example Code

- score\_v2.cpp



# Constant Array Parameters

- When a function doesn't modify an array parameter, it is considered good style to add the `const` reserved word, like this:

```
double sum(const double values[], int size)
```

- The `const` reserved word helps the reader of the code, making it clear that the function keeps the array elements unchanged.
- If the implementation of the function tries to modify the array, the compiler issues a warning.



# The const Parameter Modifier

- Recall: array parameter actually passes address of 1<sup>st</sup> element
- Function can then modify array!
  - Often desirable, sometimes not!
- Protect array contents from modification
  - Use "const" modifier before array parameter
    - *Called "constant array parameter"*
    - *Tells compiler to "not allow" modifications*



# Example – function definition

```
void addarray(int size,           // IN size of arrays
              const float A[], // IN input array
              const float B[], // IN input array
              float C[])       // OUT result array

// Takes two arrays of the same size as input parameters
// and outputs an array whose elements are the sum of the
// corresponding elements in the two input arrays.

{
    int i;
    for (i = 0; i < size; i++)
        C[i] = A[i] + B[i];

} // End of function addarray
```



# Example – function call

The function addarray could be used as follows:

In main():

```
int one[50], two[50], three[50];
    //
    //
addarray(50, one, two, three);

// but also:
addarray(20, one, two, three);

// it will only do the addition on the first 20 elements
of each array
```

