Arrays

Today

- What are arrays?
- Initializing arrays
- Common array algorithms

Arrays

Using Arrays

Think of a sequence of data:

32 54 67.5 29 35 80 115 44.5 100 65

(all of the same type, of course)
(storable as **`double`**s)

Using Arrays

32 54 67.5 29 35 80 115 44.5 100 65

Which is the largest value in this set?

(You must look at every single value to decide.)

Using Arrays

32 54 67.5 29 35 80 115 44.5 100 65

- So you would create a variable for each, of course!

```
double n1, n2, n3, n4, n5, n6, n7, n8, n9, n10;
```

Then what ???

Using Arrays

32 54 67.5 29 35 80 115 44.5 100 65

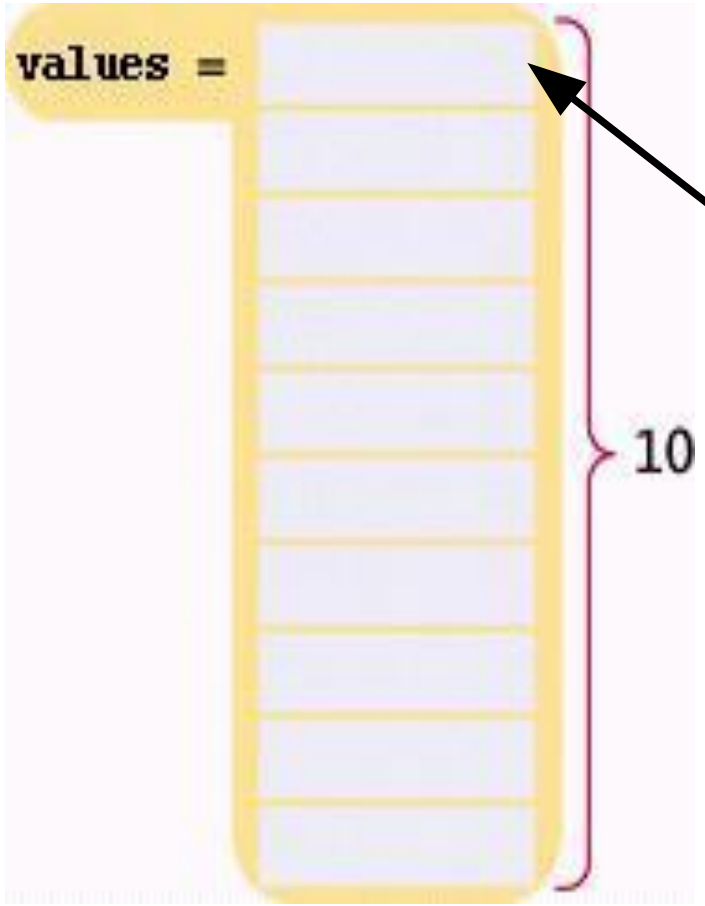
- So you would create a variable for each, of course!

```
double n1, n2, n3, n4, n5, n6, n7, n8, n9, n10;
```

Then what ???

We saw an example in class, on how to find the highest and the lowest values entered as inputs from user. *Recall, how many variables we used*

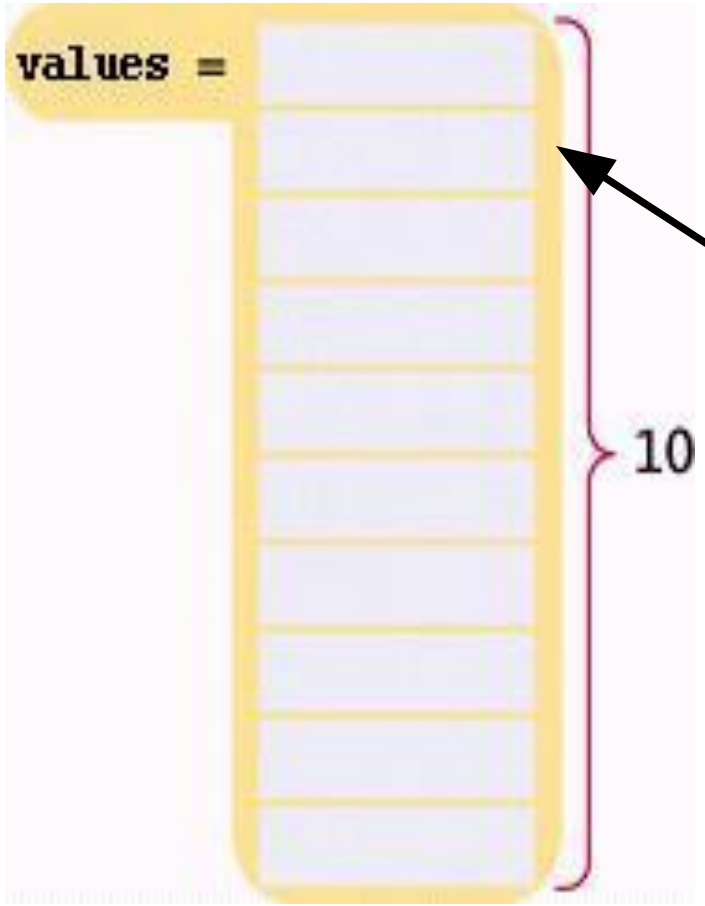
Using Arrays



Arrays - Advantage: You can easily visit each element in an array, checking and updating a variable holding the current maximum.

Hm. Is this the max, so far?

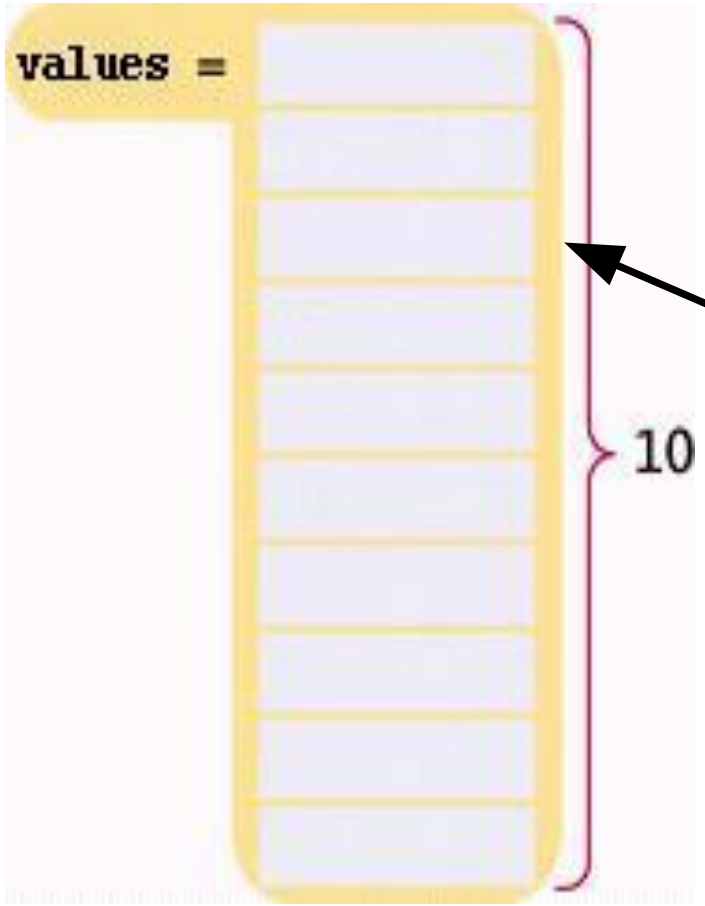
Using Arrays



You can easily visit each element in an array, checking and updating a variable holding the current maximum.

Maybe this one?

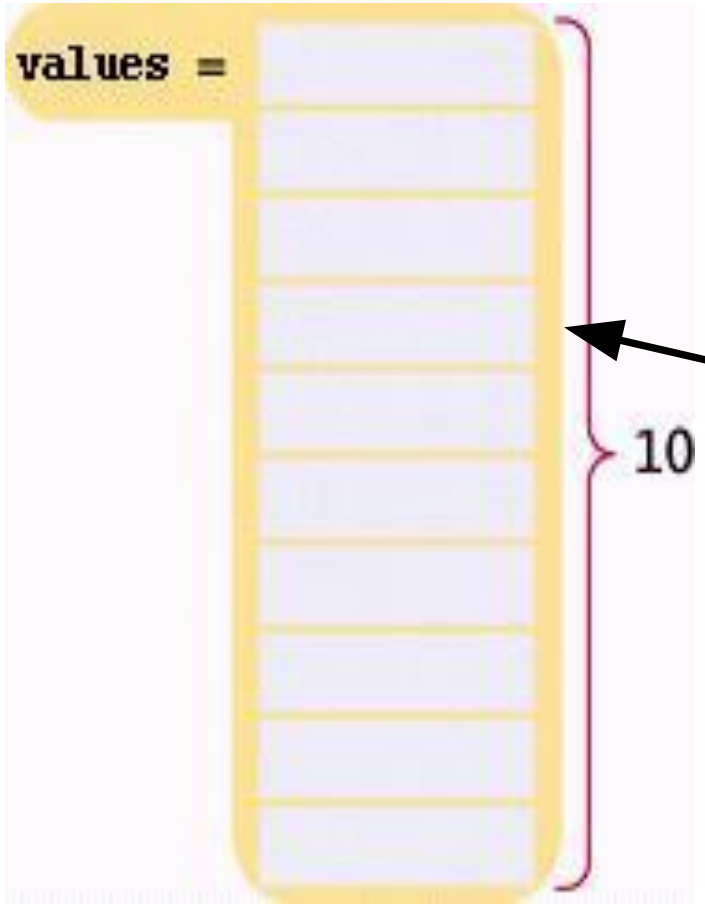
Using Arrays



You can easily visit each element in an array, checking and updating a variable holding the current maximum.

Or this one?

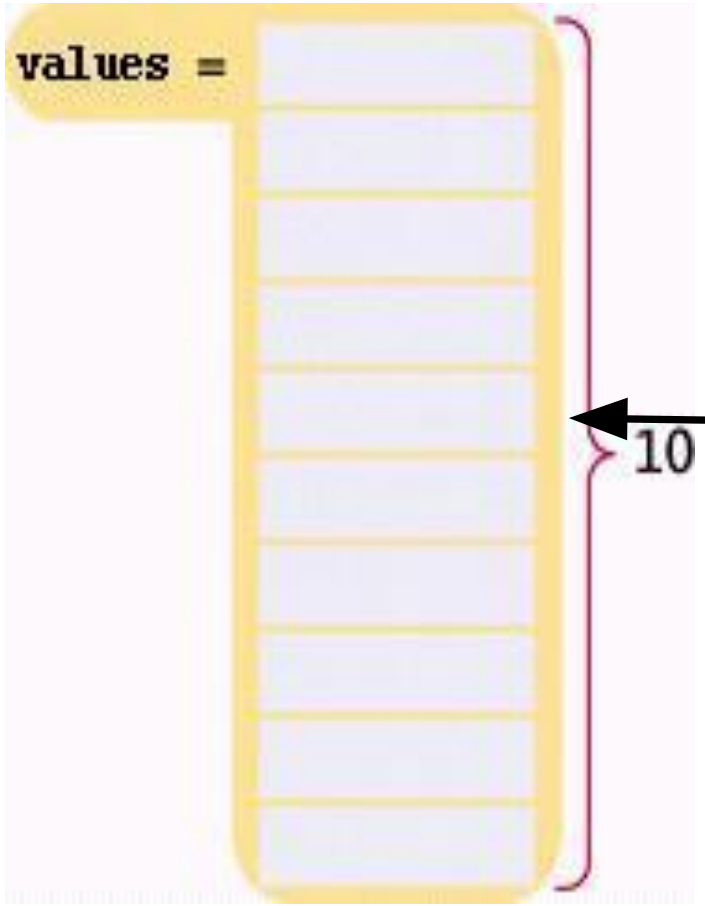
Using Arrays



You can easily visit each element in an array, checking and updating a variable holding the current maximum.

Or this one?

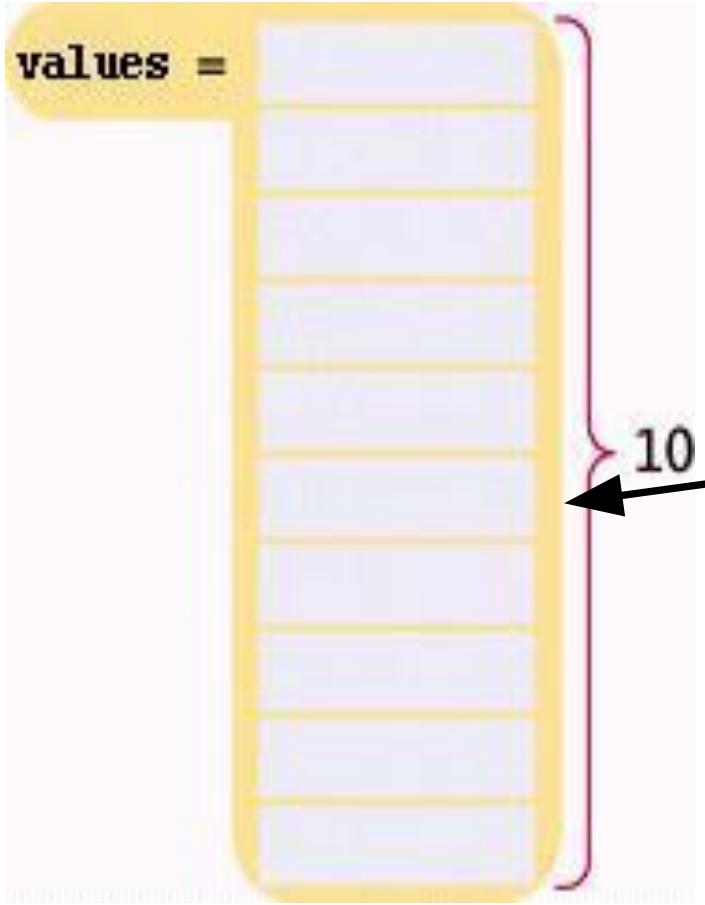
Using Arrays



You can easily visit each element in an array, checking and updating a variable holding the current maximum.

Or this one?

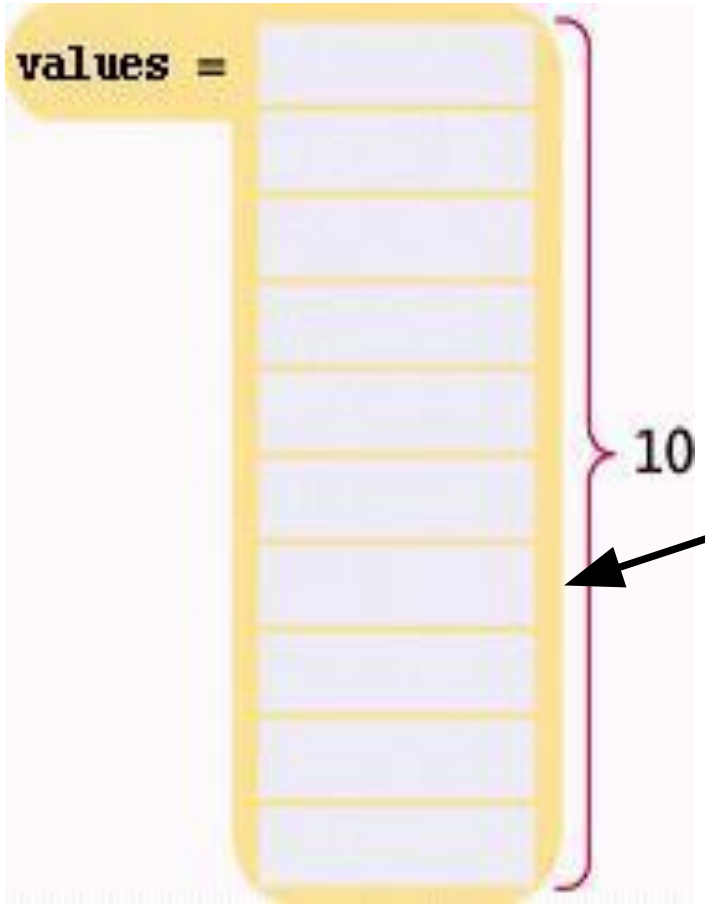
Using Arrays



You can easily visit each element in an array, checking and updating a variable holding the current maximum.

How about here?

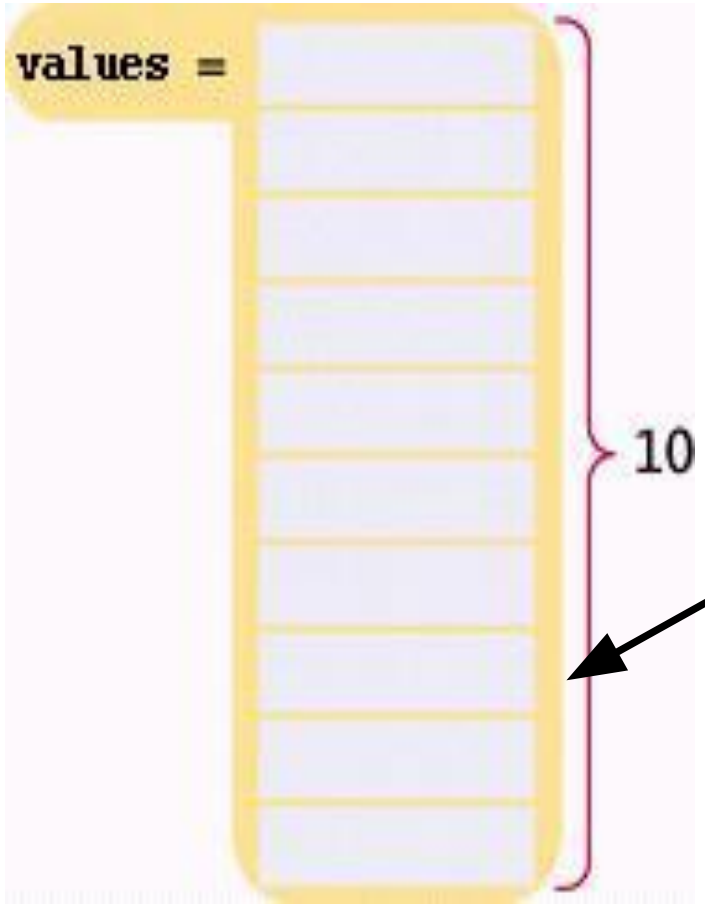
Using Arrays



You can easily visit each element in an array, checking and updating a variable holding the current maximum.

Gotta check here too!

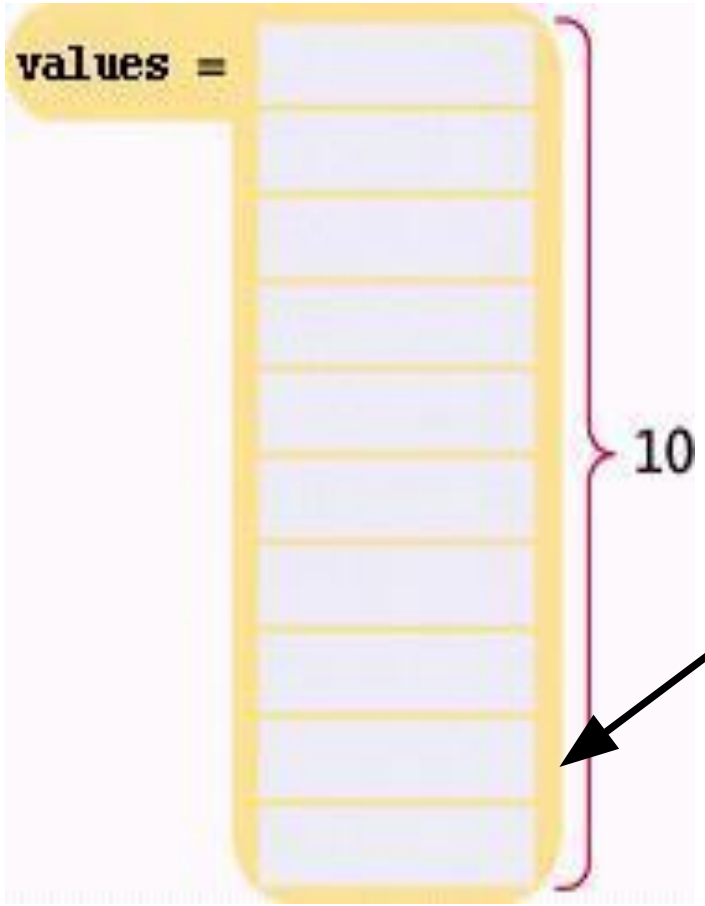
Using Arrays



You can easily visit each element in an array, checking and updating a variable holding the current maximum.

Again, maybe this one?

Using Arrays

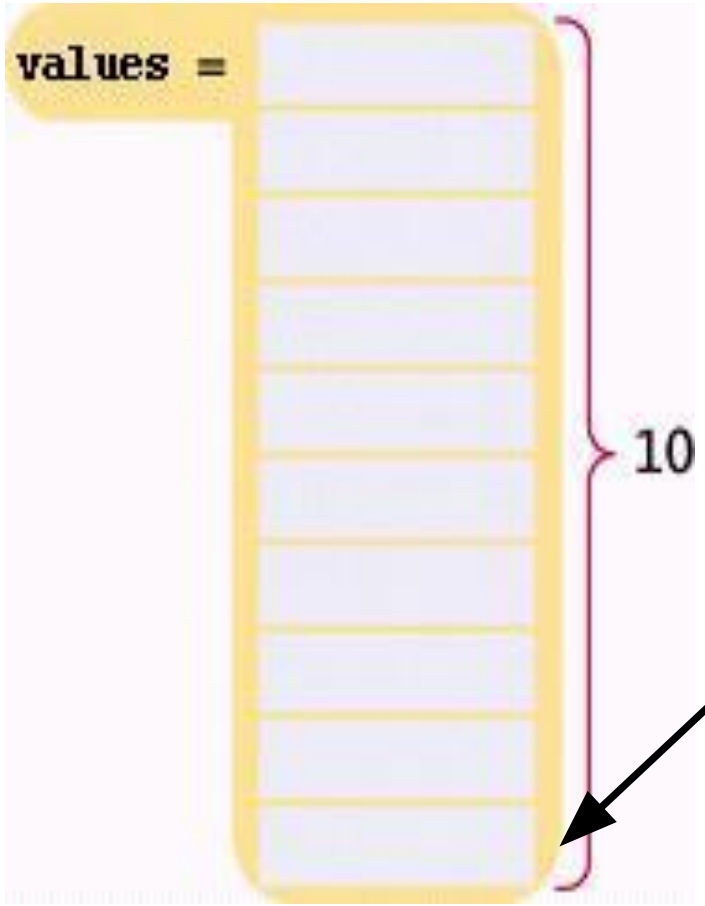


You can easily visit each element in an array, checking and updating a variable holding the current maximum.

This one?

Will this never end?

Using Arrays



You can easily visit each element in an array, checking and updating a variable holding the current maximum.

Or this one? Finally!

Using Arrays

That would have been impossible with ten separate variables!

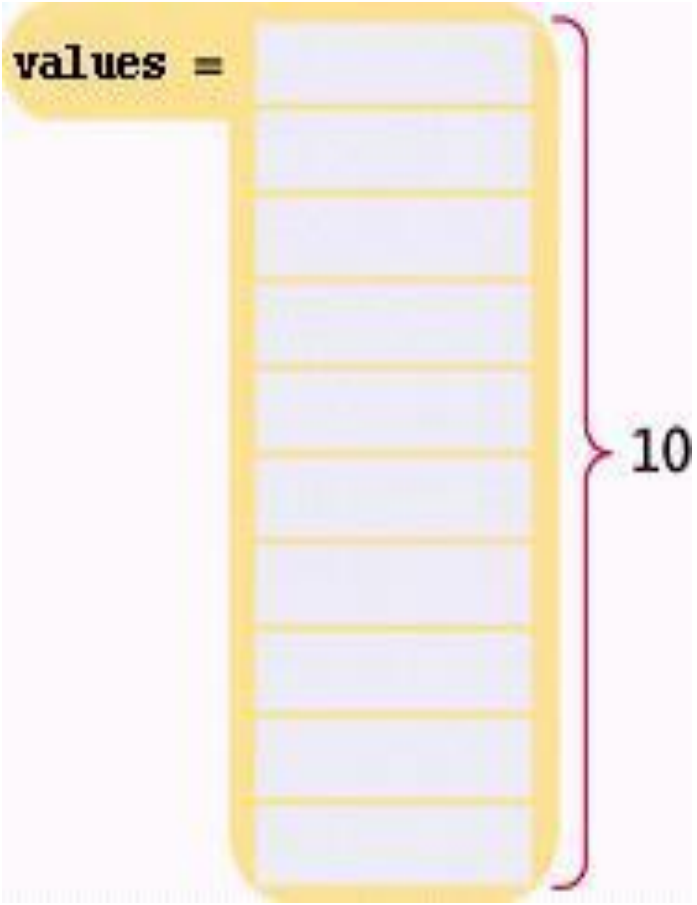
```
double n1, n2, n3, n4, n5, n6, n7, n8, n9, n10;
```

And what if there needed to be more double values in the set?

JUST A LOT OF WORK!

Defining Arrays

An “array of double”



Ten elements of **double** type can be stored under one name as an array.

```
double values[10];
```

type of each element

number of elements – the “size” of the array, must be a constant

Introduction to Arrays

Definition: An array is a collection of data of the same type, referenced as different elements of the same name.

- First "aggregate" data type
  - Means "grouping"
  - *int, float, double, char* are simple data types
- Used for lists of like items
  - Test scores, temperatures, names, etc.
  - Avoids declaring multiple simple variables
  - Can manipulate "list" as one entity

Declaring Arrays

Declare the array allocates memory

```
int score[5];
```

- Declares array of 5 integers named "score"

  Similar to declaring five variables:

  ```
  int score0, score1, score2, score3, score4;
  ```

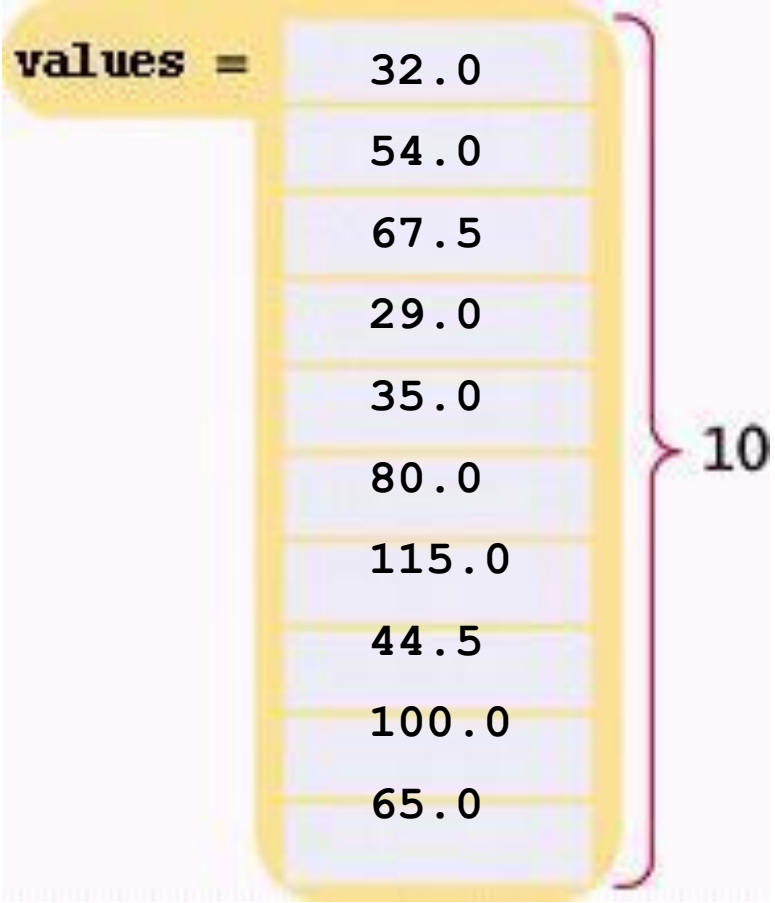
  But you don’t need 5 variable names, just one with arrays!

- Individual parts can be called many things:
  - Indexed or subscripted variables
  - "Elements" of the array
  - Value in brackets is called index or subscript
  - Numbered from 0 to (size – 1). Just like strings.

Defining Arrays with Initialization

When you define an array, you can specify the initial values:

```
double values[] = { 32, 54, 67.5, 29, 35,
 80, 115, 44.5, 100, 65 };
```



Array Syntax

Defining an Array

Size must be a constant.

Ok to omit size if initial values are given.

Element type Name Size

```
double values[5] = { 32, 54, 67.5, 29, 34.5 };
```

Optional list of initial values

Use brackets to access an element.

```
values[i] = 0;
```

The index must be ≥ 0 and < the size of the array.

Accessing Arrays

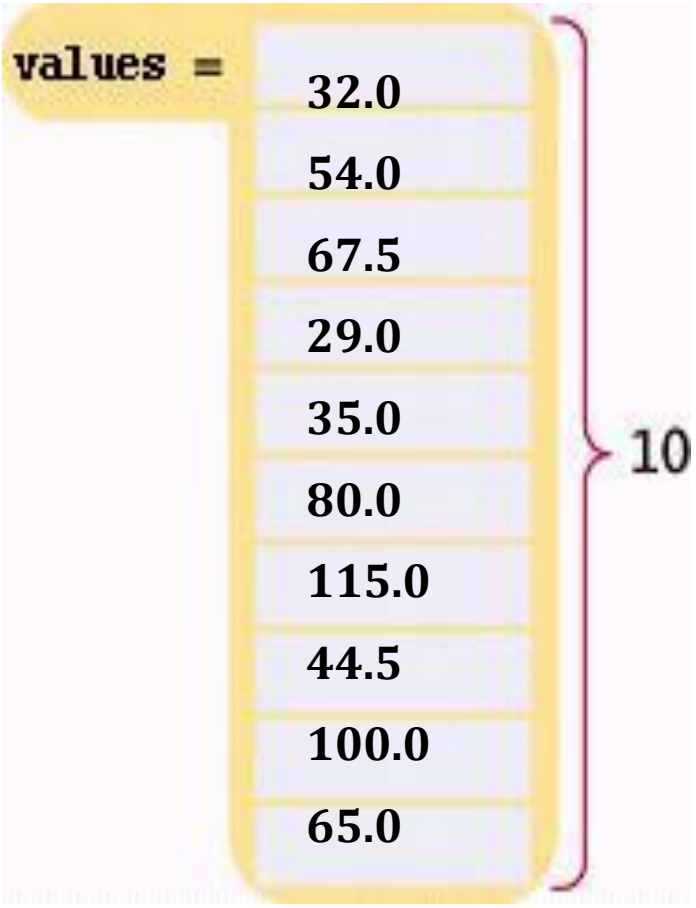
- Access using index/subscript

```
cout << score[3];
```

- Note two uses of brackets:
  - In declaration, specifies SIZE of array
  - Anywhere else, specifies a subscript
- Size, subscript need not be literal

```
int score[MAX_SCORES];
score[n+1] = 99;  -->  If n is 2, identical to: score[3]
```

Accessing an Array Element



The same notation can be used to change the element.

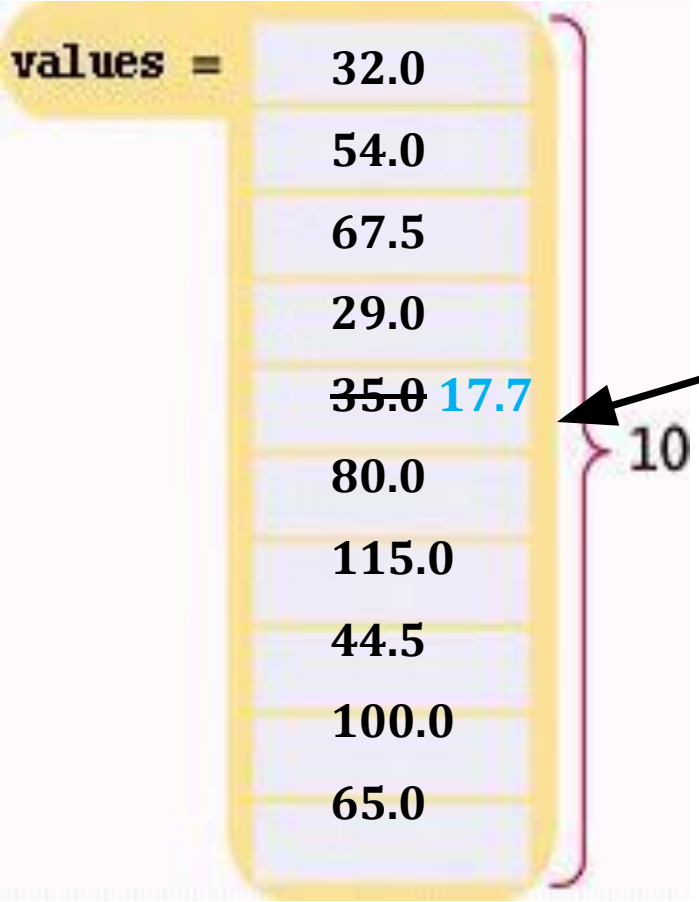
```
double values[10];
...
cout << values[4] << endl;
```

The output will be **35.0**.

Update and Access an Array Element

To access the element at index 4 using this notation: **`values[4]`**

4 is the *index*.




```
values[4] = 17.7;
cout << values[4] << endl;
```

The output will be **17.7**.

Accessing an Array Element

That is, the legal elements for the **values** array are:

values[0], the ***first*** element

values[1], the second element

values[2], the third element

values[3], the fourth element

values[4], the fifth element

...

values[9], the tenth ***and last legal*** element
recall: **double values[10];**

The index must be **>= 0** and **<= 9**.

0, 1, 2, 3, 4, 5, 6, 7, 8, 9 is … 10 numbers.

Array Usage

- Powerful storage mechanism
- Can issue commands like:
  - "Do this to i^{th} indexed variable”, where i is computed by program
  - "Display all elements of array score"
  - "Fill elements of array score from user input"
  - "Find highest value in array score"
  - "Find lowest value in array score”
- Disadvantages: size MUST BE KNOWN at declaration

Demo

Stock_prices.cpp

Common Array Algorithms

Recap: Array

Defining an Array

Size must be a constant.

Ok to omit size if initial values are given.

Element type Name Size

```
double values[5] = { 32, 54, 67.5, 29, 34.5 };
```

Optional list of initial values

Use brackets to access an element.

```
values[i] = 0;
```

The index must be ≥ 0 and < the size of the array.

Common Algorithms

- Filling an array
- Copying One array to another
- Sum, Average, Maximum and Minimum
- Swapping Elements
- Linear Search

Common Algorithms – Filling

- This loop fills an array with zeros:

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

- To fill an array with squares (0, 1, 4, 9, 16, ...).

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

Common Algorithms – Copying

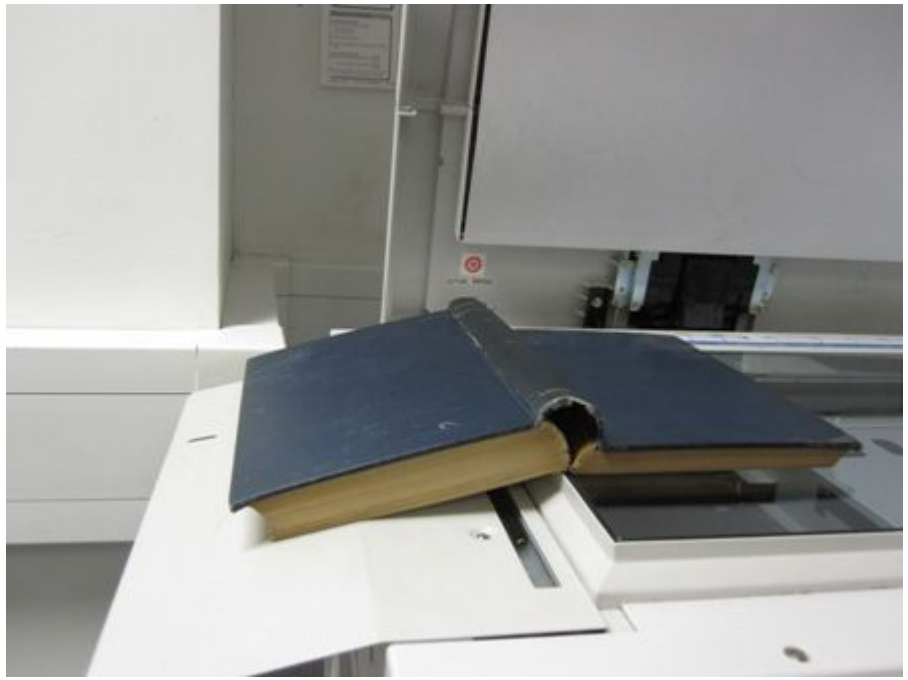
- Consider copying

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

- Can I do this?

```
squares_copy = squares; // Wont work
```

- Probably need a loop, to copy element by element *(similar to copying a book page by page)*.



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];
}
```

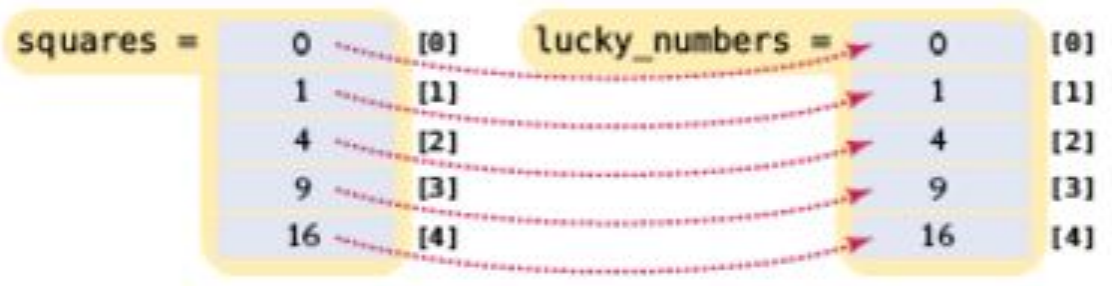


Figure 4 Copying Elements to Copy an Array

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.

Common Algorithms – Swapping Elements

Suppose we need to swap two integer values a & b

```
a = b;
b = a;
```

- Look closely! In the first line you lost – forever! – the value of a, replacing it with the value of b.
- Then what?
- Need some temporary location to save one of them? Yes!

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;
```

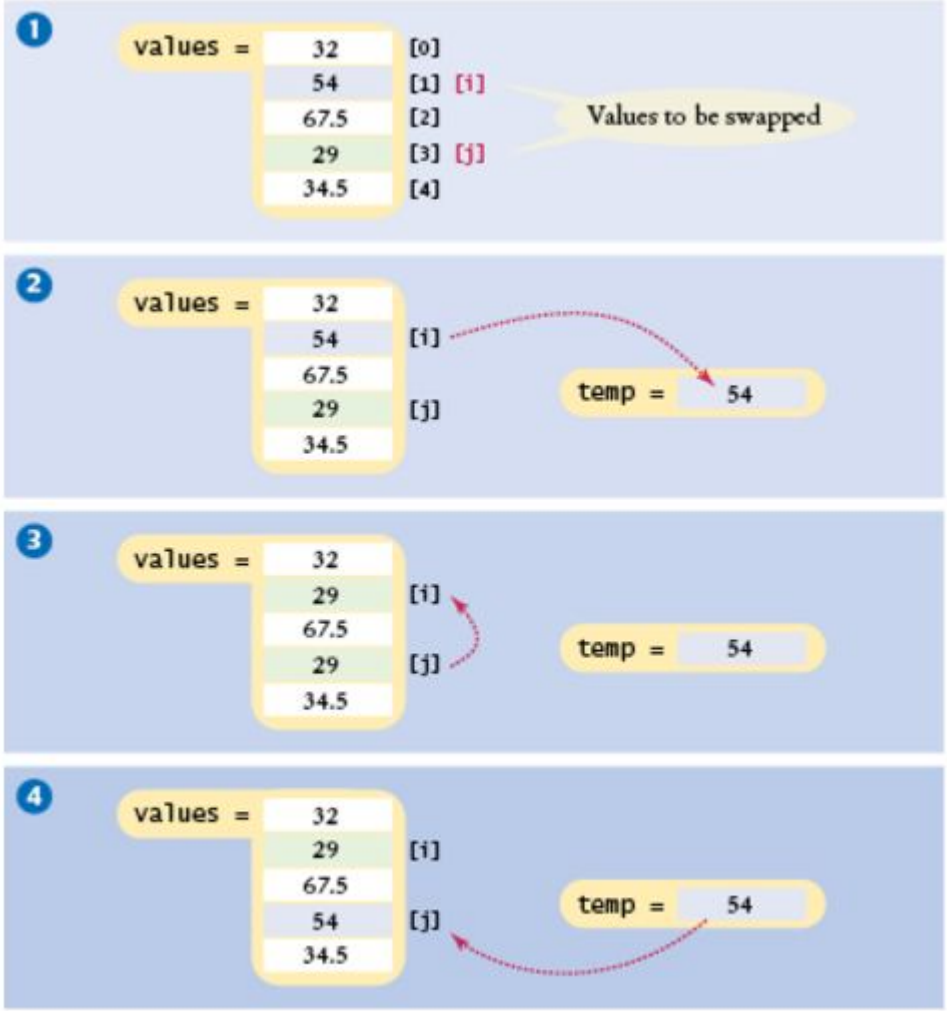
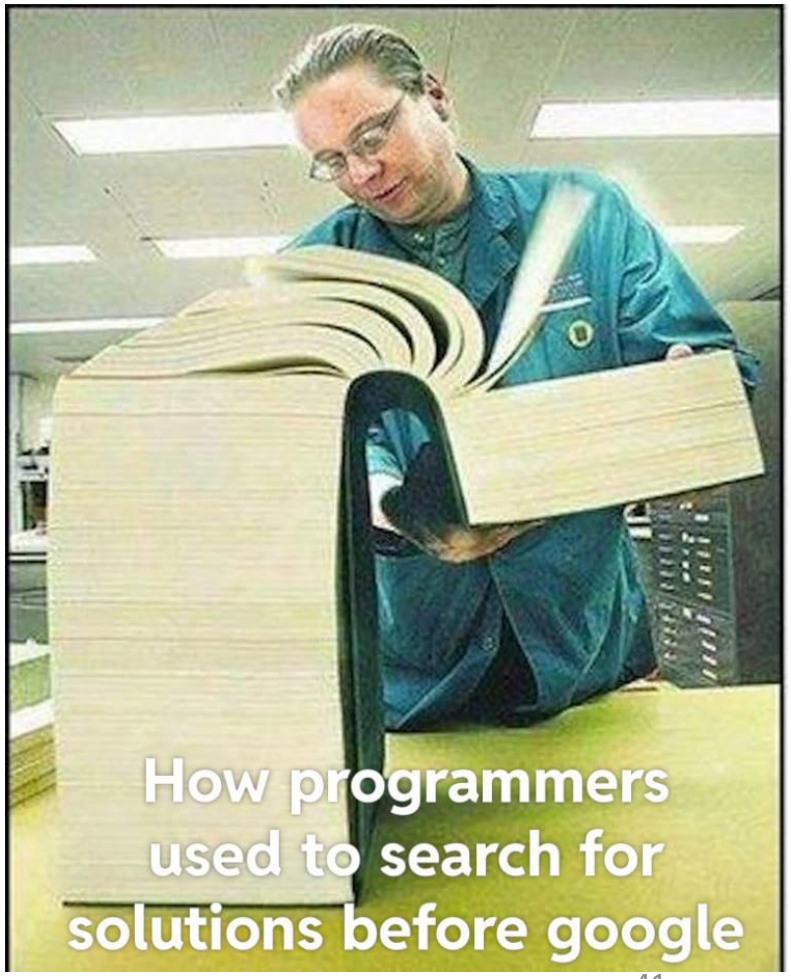


Figure 9 Swapping Array Elements

Common Algorithms – Linear Search

- Given an element, is this element present in an array?
- Is 25 present in {2, 56, 23, 25, 9, 10} ?
- How about 24?

- Well, probably need to look into the array, element by element and just find it!



Common Algorithms – Linear Search

- Let's see the logic to find 100.0 in the array
- You probably need to do the two things
  - Keep looking one by one, until you found 100.0!
  - If you don't, let's say "Not found!"

- Solution
  - Loops!
  - One condition to check at every step of the loop