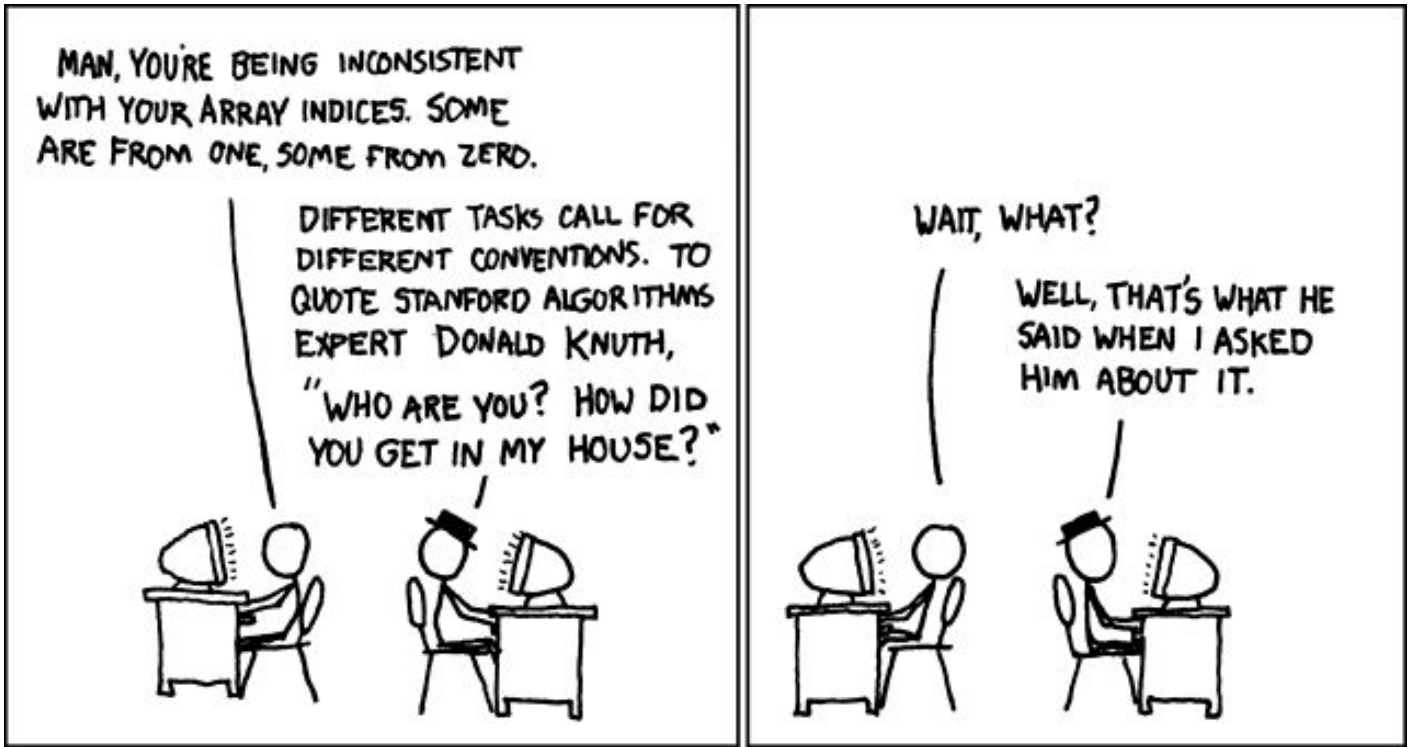
Lecture 17: Arrays



Announcements and reminders

Interview Grading (Project 1)

- **Not optional** (40 pts)
- Sign up and do by Friday 3 March
- Can earn pts even if your codes weren't working! Horrayyyy!

Submissions:

- HW 5 -- due Saturday at 6 PM

Practicum 1 -- Wednesday 5:30 - 7 PM (staggered start, don't be late, nor alarmed)

- Practicum 1 room assignments posted to Piazza:
  - 301, 303 -- ECCR 265
  - 302 -- ECCR 200
  - 304 -- ECCR 1B40
- Practice problems on Moodle -- **DO THEM. They are excellent practice problems for the practicum. That's why we call them "practice problems"**
- Cloud9 okay. 8.5x11" cheat sheet of notes okay. Non-Cloud9 internets = **not** okay

Last time on *Intro Computing…*

- We saw some common algorithms using **loops!**
  - … how to ***traverse*** a string using a loop!
  - … how to count matches in some user input
  - … how to find the first location of something

Chapter 6: Arrays and Vectors

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

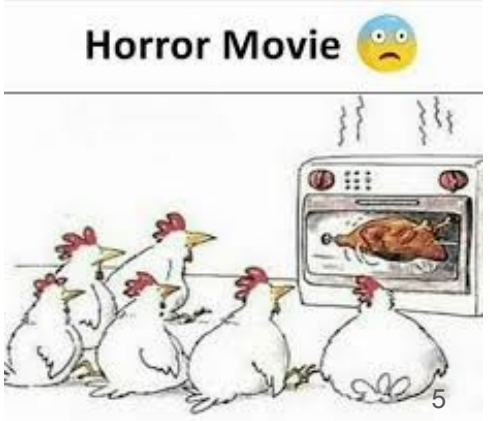
Using arrays

S’pose you have a sequence of data:

32 54 67.5 29 35 80 115 44.5 100 65

… all of which are of the same type (what are they here?)

Question: which data point is the largest in this set?



Using arrays

S'pose you have a sequence of data:

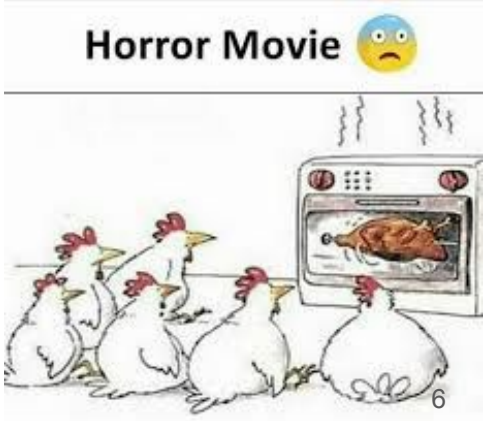
32 54 67.5 29 35 80 115 44.5 100 65

… all of which are of the same type (what are they here?)

Question: which data point is the largest in this set?

You need to create a variable for each one:

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



Using arrays

S'pose you have a sequence of data:

32 54 67.5 29 35 80 115 44.5 100 65

… all of which are of the same type (what are they here?)

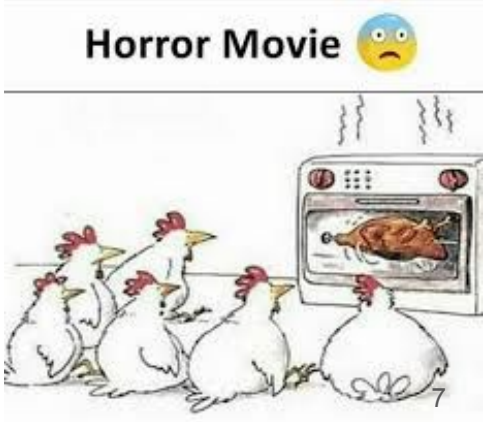
Question: which data point is the largest in this set?

You need to create a variable for each one:

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

… but what if we had a much ***larger*** number of data points?

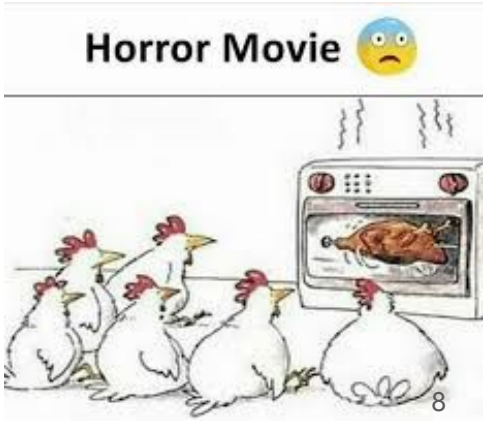
Creating new variables for all of them would be **a horror show**



Using arrays

Instead, we could pack ***all*** of those 10 data values into ***a single array*** :

values =

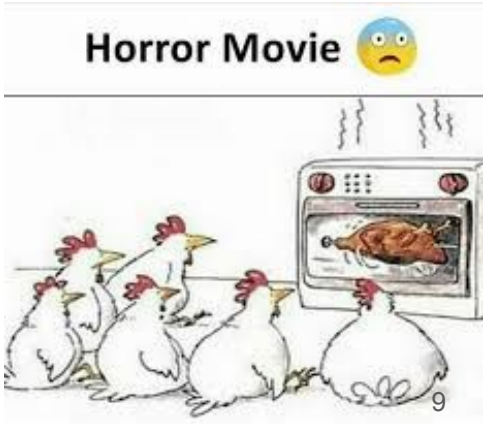


Using arrays

Instead, we could pack ***all*** of those 10 data values into ***a single array*** :

values =

… and then **loop over** all of the values and check which is the maximum so far



Defining arrays

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

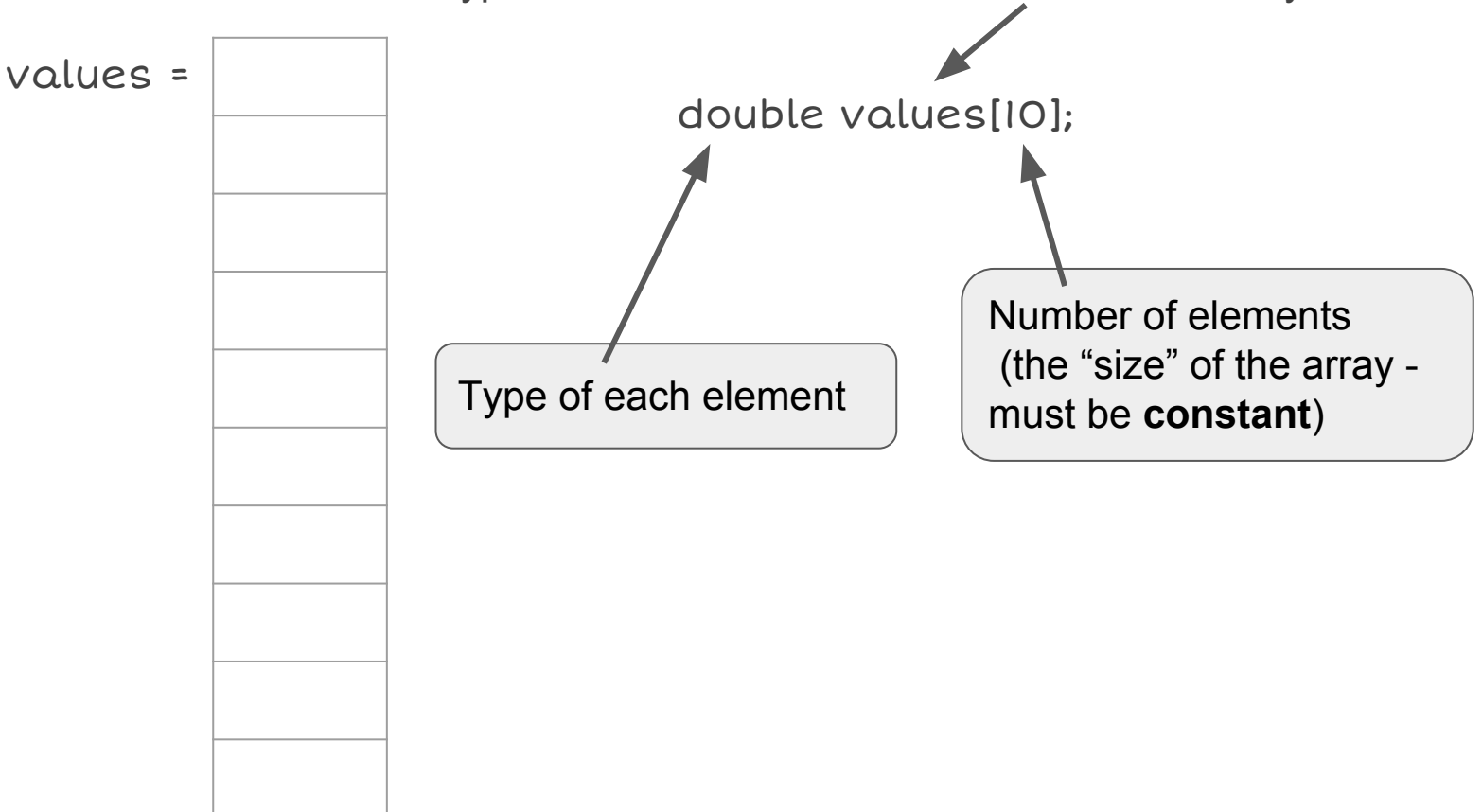

```
double values[10];
```

Type of each element

Number of elements
(the “size” of the array - must be **constant**)

Defining arrays

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



Defining arrays

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

- our first ***aggregate*** data type
  - Aggregate ⇔ Grouped
  - int, float, double, char are all ***simple*** data types (**one** piece of information)

- Used for lists of similar items
  - Examples: test scores, temperatures, names, books, etc…
  - Avoids repetition of declaring multiple simple variables
  - Can manipulate entire “list” as one entity
    - **Example:** If you wanted to curve all scores up by 2 points.

Declaring arrays

Declare the array → **allocates memory**

```
int scores[5];
```

- Declares `scores` to be an array containing 5 integers
- Similar to declaring 5 variables:
  - `int scores[0], scores[1], scores[2], scores[3], scores[4]`

- Individual parts can be called many things:
  - **Indexed** or **subscripted** variables
  - **Elements of the array**
  - Value in brackets is called the **index** or **subscript**
    - Runs from 0 to (size-1)

Declaring arrays

When you declare an array, you can also initialize it (just like simple variables!)

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

values =

32.0
54.0
67.5
29.0
35.0
80.0
115.0
44.5
100.0
65.0

Array syntax

Defining an array:

Type of each element

Name

Size
(# elements)

(Can omit size argument if initial values are given)

Optional list of initial values

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

```
values[i] = 0.0;
```

Use brackets to access an element

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

Accessing arrays

Access using index/subscript

```
cout << scores[3];
```

Note two uses of brackets:

- In **declaration**, specifies the **size** of the array
- Anywhere else, specifies a subscript (particular element)

Size, subscript need not be literal:

```
int scores[NUM_STUDENTS];
...
cin >> scores[i+1];
```

← if `i = 2`, identical to `cin >> scores[3]`, for example.

Accessing arrays

To access the element at index 4, use notation: `values[4]` (4 is the ***index***)

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

values =

32.0
54.0
67.5
29.0
35.0
80.0
115.0
44.5
100.0
65.0

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

→ the output will be 35.0

Accessing arrays

The same notation can be used to **modify the contents of an array:**

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

values =

32.0
54.0
67.5
29.0
35.0
80.0
115.0
44.5
100.0
65.0

```
values[4] = 17.7;
```

→ the value at index 4 will now be 17.7

Accessing arrays

The same notation can be used to **modify the contents of an array:**

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

values =

32.0
54.0
67.5
29.0
17.7
80.0
115.0
44.5
100.0
65.0

```
values[4] = 17.7;

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

→ the value at index 4 will now be 17.7

→ **what will the output be now??**

Accessing arrays

The legal elements for the values array are:

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

`values[1]`, the second element

`values[2]`, the third element

…

`values[8]`, the ninth element

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

`values[i]`, the *i*+1th element -- The index i must be ≥ 0 and ≤ 9

0 1 2 3 4 5 6 7 8 9 ← that's 10 numbers!

Array usage

Powerful storage mechanism

Can do useful things like…

- *Do **this** to the i^{th} element*, where *i* is computed by the program
- *Display all elements of the array* `scores`
- *Fill elements of the array* `scores` *from user keyboard input*
- *Find the highest value in the array* `scores` ← **Example: largest.cpp**
- *Find the median value in the array* `scores`

Disadvantage: size **must be known** when you declare the array

Had to do either: `double scores[] = { 23, 59.5, 100, ... };`

or `double scores[200];`

Array usage

Disadvantage: size **must be known** when you declare the array

Had to do either: `double scores[] = { 23, 59.5, 100, ... };`

or `double scores[200];`

One way around this: Define array with ***capacity*** as large as you think you might possibly need!

```
const int CAPACITY = 10;
double scores[CAPACITY];
int current_size = 0; int input;
while (cin >> input) {
  if (current_size < CAPACITY) {
    scores[current_size] = input;
    current_size++;
  }
}
```

Some quick notes about global variables

So far, we have dealt with only **local variables** -- variables that are known *only* to the function that created them.

Definition: A **global variable** is a variable defined outside the scope of any function.

```
int x;

int addTwo(int num) {
    return num + 2;
}

int main() {
    x = 5;
    int y = addTwo(x);
    return 0;
}
```

Some quick notes about global variables

Upshot: Global variables are sometimes useful, but almost always difficult to predict and debug.

For HW 6: Do not use global variables. Code quickly becomes spaghetti.

What just happened?

- We saw how to store groups of similar data using **arrays!**
  - … how to declare and define arrays!
  - … how to access data elements within an array!
  - … how to change values in an array!