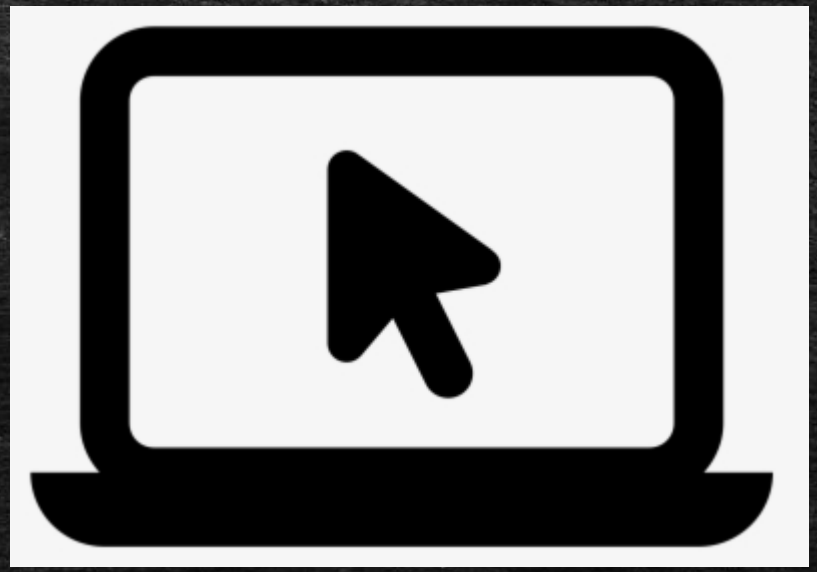
Arrays & Pointers



CS 150 – C++ Programming I
Lecture 19

Introducing Arrays

- A built-in derived type for collecting elements
  - Homogeneous collection (all same type)
  - Contiguous storage & random access (direct to any element)
- Define: `double nums[10]; // array of 10 doubles`
  - Capacity must be constant, known at compile time
- Array definition allocates entire block `int a[5];`
  - Sequential (linear) and contiguous allocation
  - Want element at `a[795]`? `a + 795 * sizeof(e)`
  - Simple arithmetic; very, very fast

Array Initialization & Access

- Arrays can be initialized when defined (*All versions of C++*)
  - `int a[4] = {72, 81, 90, 75};`
  - `double grades[100] = {0};`
  - ~~`int grades[3] = {99, 80, 90, 87};`~~
  - `int grades[] = {99, 80, 75};`
  - C++11 - default initialize: `int grades[5]{}`
- Individual elements are indexed or subscripted
  - Access using `a[subscript]`, `0` to `CAPACITY - 1`
  - When you go out of bounds? Undefined behavior!
  - Never an exception; segmentation fault, trap or overwrite

Array Assignment and Comparison

- There is no array variable as in Java (below)
  - `int[] var = new int[10]; // variable and array`
- The array name acts like a constant pointer or literal
  - Thus, you cannot assign or compare using array names

```
int a1[] = {2, 3, 4}, a2[3];
a2 = a1;              // Illegal
a2[0] = a1[0];        // OK
a2 = {2, 3, 4};       // Illegal
if (a2 == a3)…        // Legal; but stupid
```

Size of an array

- An array does not "carry around" its size, like *vector*
  - There is no `a.length` as with Java arrays
- The compiler knows the size of an array when declared
  - Only when declaration is in-scope at compile time
  - `int a[SIZE]{};`
    `size_t bytes = sizeof(a); // size in bytes`
  - `size_t len = sizeof(a) / sizeof(a[0]);`
- In C++ 11: `end(a) - begin(a)`
- In C++ 14+: `std::size(a)` *// in <vector> or <string>*

Processing Arrays

- In your functions, you'll process the array with loops. But, how does the loop know to stop, if you only pass the address?
  - Method 1: pass the allocated or maximum size
  - Method 2: use a sentinel to mark the end
  - Method 3: calculate a pointer to the end and pass that
  - Method 4: use `begin()` and `end()` iterators
  - Method 5: use a *`for-each`* loop
- Let's use all of them for the next exercise

Your Turn

- Open *arrays.cpp* in your workspace
- Create 5 arrays
  - A 3-element `int`, 4 element `char`, 5-element `long`, 6-element `float` and 7 element `double`
- 1) Print each array, using the array name (what happens)
- 2) Print the contents of each array using 5 different loops
- 3) Read and write outside of an array bounds before printing. What happens to the other arrays?