EECS 280 – Lecture 4

Arrays and Pointers

1/19/2022

Kinds of Objects in C++

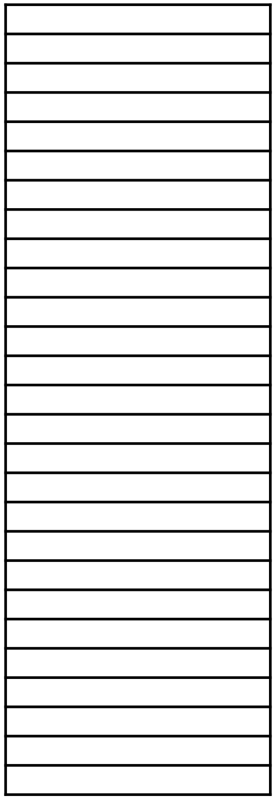
- **Atomic**
  - Also known as **primitive**.
  - `int`, `double`, `char`, etc.
  - Pointer types.

- **Arrays** (homogeneous)
  - A *contiguous* sequence of objects of the same type.

- **Class-type** (heterogeneous)
  - A compound object made up of member subobjects.
  - The members and their types are defined by a **struct** or **class**.

Arrays Intro

Memory



Arrays in C++

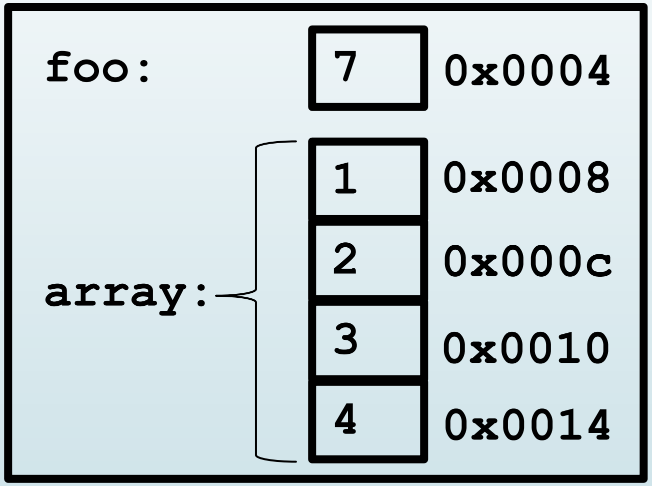
- In C++ an array is a very simple *collection* of objects.
- Arrays…
  - …have a fixed size.
  - …hold elements of all the same type.
  - …have ordered elements.
  - …occupy a *contiguous* chunk of memory.
  - …support constant time random access. (i.e. “indexing”)

Array Decay

- Try to get the value of an array…
  - It suddenly "decays" into **a pointer to its first element**.

```
int foo = 7;
int array[4] = { 1, 2, 3, 4 };
cout << array << endl;
```

Prints “0x0008”!



foo:	7	0x0004
array:	1	0x0008
	2	0x000c
	3	0x0010
	4	0x0014

Array Decay

- The tendency of arrays to turn into pointers has a few consequences…

- You can't assign arrays to each other.

```
int arr1[4] = { 1, 2, 3, 4 };
int arr2[4] = { 5, 6, 7, 8 };
arr2 = arr1; // ERROR: Type mismatch
```

Not trying to get the value. Still an array.

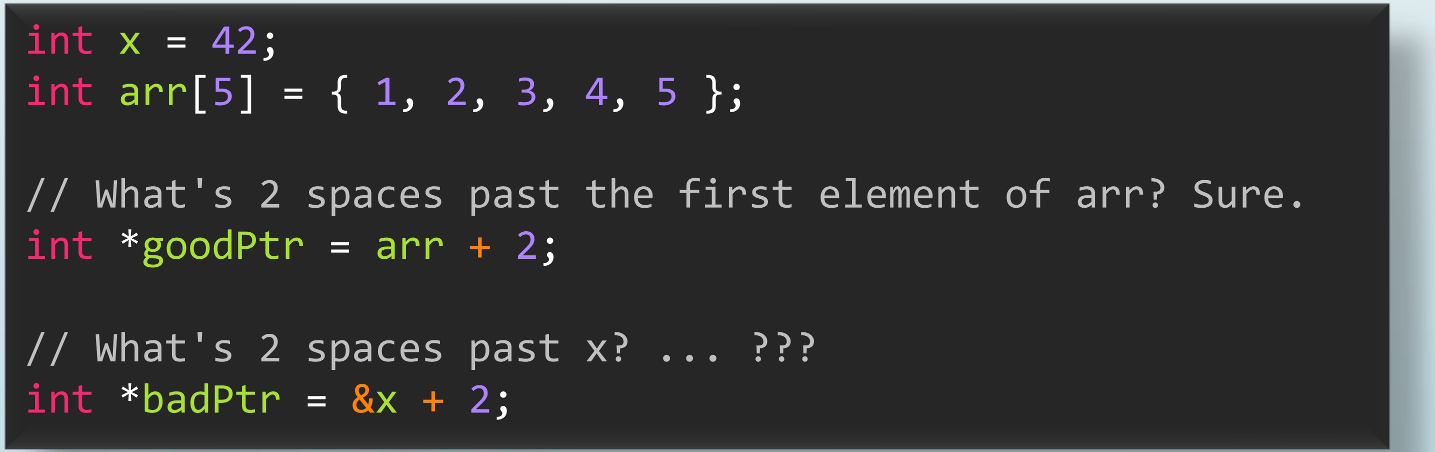
Need to get the value. Turns into a pointer :(.

Pointer Arithmetic

Pointer Arithmetic

- How does pointer arithmetic work?
  - `int *ptr`; The compiler knows how big an `int` is. (4 bytes[1])
  - `ptr + x` computes the address `x` `int`s forward in memory
  - Operators: `+`, `-`, `+=`, `-=`, `++`, `--`

- Warning! Pointer arithmetic only makes sense in arrays!
  - Arrays are guaranteed to be **contiguous** memory.



```
int x = 42;
int arr[5] = { 1, 2, 3, 4, 5 };

// What's 2 spaces past the first element of arr? Sure.
int *goodPtr = arr + 2;

// What's 2 spaces past x? ... ???
int *badPtr = &x + 2;
```

[1] Depends on the platform.

Array Indexing

- **Indexing** is a shorthand for **pointer arithmetic** followed by a **dereference**.

  `ptr[i]` is defined as `*(ptr+i)`

- Generally used with arrays:

```
int arr[4] = { 1, 2, 3, 4 };
cout << arr[3] << endl;
cout << *(arr + 3) << endl;
```

Equivalent

arr turns into a pointer, gets offset by 3, then dereferenced

Pointer Comparisons

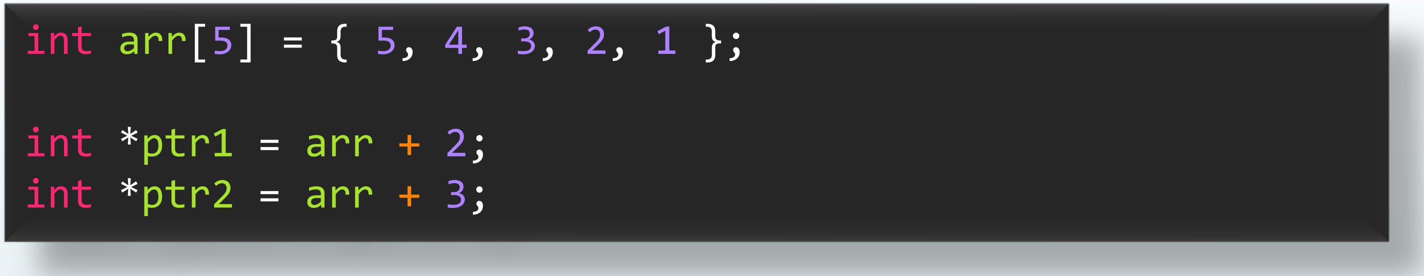
- We can also use comparison operators with pointers.

`<, <=, >, >=, ==, !=`

- These just compare the address values numerically.

Exercise: Pointer Comparison

- Given an array and some pointers…



```
int arr[5] = { 5, 4, 3, 2, 1 };

int *ptr1 = arr + 2;
int *ptr2 = arr + 3;
```

- Are the following expressions true or false?
  - `ptr1 == ptr2`
  - `ptr1 == ptr2 – 1`
  - `ptr1 < ptr2`
  - `*ptr1 < *ptr2`
  - `ptr1 < arr + 5`

Don’t do this. Ever.

- We know this equivalence:

```
arr[i] = *(arr+i)
```

- Let’s try something…

```
 arr[i]
*(arr+i)
*(i+arr)
 i[arr]
```

- Yeah. That actually works.

Ken Thompson

Turing Award Recipient

Created the Unix Operating System

Frances Allen

Turing Award Recipient

Pioneering work in Optimizing Compilers

Break Time!

You might be interested in the announcement below from **Renew CS:**

Renew CS is offering a free mentoring program with the goal of improving the success of women and non-binary students in computer science (Engr 101, EECS 183, 203, and 280). We offer weekly help sessions (both in-person and remote) as well as monthly special topic sessions run by undergraduate students.

For more information on Renew CS see
https://tinyurl.com/35vy26dt.
See a video created by two of our mentors at
https://tinyurl.com/bdtb5kmw.
To express interest in receiving mentoring, fill out the form at https://forms.gle/WmQTJ9Prk2W9S1Cq7.

Break Time!

We’ll start again in one minute.

Traversal by Index

```
int const SIZE = 5;
int arr[SIZE] = { 1, 2, 3, 4, 5 };
```

- Traversal by Index
  - Keep track of an integer **index** variable.
  - To get an element, use the index as an **offset** from the beginning of the array.

Index starts at offset of 0.

Continue until index too large.

```
for (int i = 0; i < SIZE; ++i) {
  cout << *(arr + i) << endl;
  cout << arr[i] << endl;
}
```

Increment index.

Use subscript to access element at index.

Traversal by Pointer

```
int const SIZE = 5;
int arr[SIZE] = { 1, 2, 3, 4, 5 };
```

- Traversal by Pointer
  - Walk a **pointer** across the array elements.
  - When you want an element, just dereference the pointer!

Notice that "end" is really "one past the end"

Continue until pointer at end.

```
int *end = arr + SIZE;
for (int *ptr = arr; ptr < end; ++ptr) {

    cout << *ptr << endl;
}
```

Pointer starts at beginning of the array.

Increment pointer.

Dereference pointer to current element.

Functions and Array Parameters

Exercise: Array Functions

- Find the file “`L04.3_maxValue`” on Lobster.

  lobster.eecs.umich.edu

- Write the code for `maxValue`.
- Use the visualization to check your answer.

```
int maxValue(int arr[], int len) {
  // WRITE YOUR CODE HERE!
  // Use a loop and indexing.
}

int main() {
  int arr[4] = {1, 2, 3, 4};
  int m = maxValue(arr, 4);
  cout << m << endl;
}
```