Arrays Episode III: Revenge of the Arrays

CS 211

Adam Sweeney

October 13, 2017

Wichita State University, EECS

Introduction

- No new syntax today
- Consider how important it is not go out of bounds when using arrays
  - Are there other precautions?
- What other array operations might there be?

Agenda

- Chapter 7.3
- Partially filled arrays
- Searching arrays
- Sorting arrays

Partially-Filled Arrays

Partially-Filled Arrays

- There is no rule that arrays must always be full
- Couple that with the fact that we want to avoid array out-of-bounds access at all costs
- Why not declare very large arrays?

Very Large Arrays

- Advantages
  - Out-of-bounds errors are much harder to achieve
  - Consider a data set that typically consists of a dozen elements, sometimes spiking up to ~50
  - An array of size 100 should easily handle all known scenarios, and quite a few unknown!
  - Lets us "choose" the size of our data set
- Disadvantages
  - Wastes memory
  - Arrays are always fully allocated, so unused array space is wasted space
  - A bit more maintenance to utilize a partially filled array
- Verdict?
  - Generally, a partially-filled array is a great idea
  - Just don't go too overboard on the size

Using Partially-Filled Arrays

- When we were filling arrays, we used the size as our limiting factor
- If we aren't filling an array entirely, relying on the size is pointless
  - We would be examining or doing operations on "dead" space
- Instead, we care about how many elements we plan to use
- However, we still need to track the size, at least initially

An Example

```
int fillArray(int a[], const int size);
double computeAverage(const int a[], const int numUsed);
void showDifference(const int a[], const int numUsed);

int main()
{
    int arr[40];
    int numUsed = fillArray(arr, 40);
    showDifference(arr, numUsed);
    return 0;
}
```

- What we're looking at are the function prototypes
- We initially want to be sure we don't go out of bounds when filling the array
- Once the array is [partially] full, we only care about how many elements were occupied

An Example

- `https://repl.it/MY0H/latest/426347`
- Notes:
  - Only the `fillArray` function is passed the size of the array
  - After that, all the other functions only need to know how many elements are "active"
  - When using a partially-filled array, we don't have to be *as* worried about out-of-bounds errors, but we should instead worry about accessing "bad" elements where there is no meaningful data

Searching Arrays

Searching Arrays

- Typically, when a lot of information is gathered in one place, we will need to search it to find what we need
- Arrays can be searched
- Like a library, we don't just want to know whether something exists, we want to know where it is
    - *Corollary*: Would you ask Google a question and be satisfied if the response was simply "Yes, there is a page that answers that question?"
    - Probably not; the same applies to arrays. We typically want to know where an item is

How Can We Search an Array?

- There a lot of ways to search an array
- We will focus on the most basic search method, which is to start at the first element, and check every element until we find what we're looking for
- `https://repl.it/MYip/latest/426347`
- Notes:
  - The actual searching is done in the function `searchArray`
  - It returns the index where the value is found
  - This is more valuable information than just the value
    - If we know where it is, we can get the value by calling the specific element
    - And we have the added benefit of knowing where in the array it is located
    - This will come in handy for the next section

Sorting an Array

Sorting an Array

- A whole slew of search algorithms become available when an array is sorted
- A sorted array can be more useful, depending on the type of data
  - Ranked data, lists, etc.
- There are literally volumes written on how to sort information
  - A bit more than half of CS 560 is devoted to sorting algorithms
  - The links below demonstrate a variety of sorting algorithms
  - `http://panthema.net/2013/sound-of-sorting/`
  - `https://www.toptal.com/developers/sorting-algorithms/`
  - We will only briefly cover one of the simplest methods for sorting an array

Selection Sort

- The behavior of selection sort depends on how we are sorting
- We will assume that we want to sort in ascending (small to large) order
- When the sort starts, it will look through the entire array to find the minimum value
- When the **index** of the minimum value is known, the values at that index and index 0 are swapped
- We then look through entire array, **minus the first element**, to find the location (index) of the next minimum value
- The value of the element containing the new minimum is swapped with index **1**
- And so on until we have looked at the array *[size - 1]* times
    - There is no need to look at the solitary last element, it will have been sorted by default

Selection Sort Code

- Since this was done in class, the link will be available on Blackboard