

Introduction to Data Structures: Arrays

Wagner Truppel
Lecturer, Dept. of Computer Science & Engineering
UC Riverside

wagner@cs.ucr.edu
<http://www.cs.ucr.edu/~wagner>
<http://www.cs.ucr.edu/cs12>

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 1

Announcements

- If you have a question that may be of interest to other students as well, please send it to the course mailing list
- In case you haven't been reading the course mailing list, you should start doing so immediately!

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 2

Today's Topics

- Introduction to Data Structures: **Arrays**
- Searching through an array
 - ◆ Linear & Binary search
 - ◆ First look at Big-O notation
- C++ and statically defined arrays

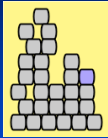
©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 3

Intro to Data Structures

- The “bookstore” example
 - ◆ How do you find the book you’re looking for ? You need to use some kind of *strategy* and...
 - ◆ ... of course, we want the best *strategy* possible, but...
 - ◆ ... what “best” means will depend on how the books are organized (stored)
- In CS, these *strategies* are called **algorithms**

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 4


MessyBookPile, Inc.



- This bookstore keeps its books all in one big pile, with no organization of any kind
- Your algorithm is then:
 - ◆ For each book
 - ★ If it is not the book I want, continue searching
- If the bookstore has **1,000,000 books**, and if it takes **10 secs** for you to check each book, it may take you as long as **4 months** to find the book you want
- Not a very good algorithm, is it ?
- **Note:** search time is directly proportional to number of books: **$O(n)$**

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 5


LongBookShelf, Inc.



- This competitor bookstore keeps its books all in one very long **array, sorted by ISBN**
- One possible algorithm is again **linear search**:
 - ◆ For each book
 - ★ If it is not the book I want, continue searching
- This is the same algorithm used by MessyBookPile, Inc.
- Again... search time is directly proportional to number of books: **$O(n)$**
- Note that the algorithm did **not** make use of the fact that the books are sorted

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 6


LongBookShelf, Inc.



- Another possible algorithm is **binary search**:
 - ◆ Look at the middle book
 - * If it is the book I want, stop
 - * If the book I want comes **before** the middle book, search the **first half** of the array using this same algorithm
 - * If the book I want comes **after** the middle book, search the **second half** of the array using this same algorithm
- This **does** make use of the fact that the books are sorted
- **Note**: recursive algorithm ! (we'll study recursion soon)
- Now... search time is proportional to the **logarithm** of the number of books: $O(\log_2 n)$
- **1,000,000 books, 10 secs/book = ?**

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 7

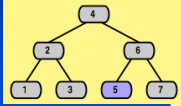
LongBookShelf, Inc.



- Another possible algorithm is **binary search**:
 - ◆ Look at the middle book
 - * If it is the book I want, stop
 - * If the book I want comes **before** the middle book, search the **first half** of the array using this same algorithm
 - * If the book I want comes **after** the middle book, search the **second half** of the array using this same algorithm
- This **does** make use of the fact that the books are sorted
- **Note**: recursive algorithm ! (we'll study recursion soon)
- Now... search time is proportional to the **logarithm** of the number of books: $O(\log_2 n)$
- **1,000,000 books, 10 secs/book = 200 s < 4 min !!!**
 [$2^{20} = 1,048,576$ so $\log_2(1,000,000) \approx 20$]

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 8

BinaryBooks, Inc.



- This bookstore keeps its books organized in a **binary search tree**
- It's suing LongBookShelf, Inc. for they've stolen the idea of **binary search** from BinaryBooks, Inc.
- The algorithm is **binary search**:
 - ◆ Look at a node of the tree
 - * If it contains the book I want, stop
 - * If the book I want comes **before** the book in this node, search this node's **left** child, using this same algorithm
 - * If the book I want comes **after** the book in this node, search this node's **right** child, using this same algorithm

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 9

Bottom line...

- Data structures are "containers" for the data your program manipulates, much like built-in data types (int, float, etc), but more powerful
- The choice and efficiency of an algorithm depends on how the data it accesses is organized
- Likewise, how you're going to store your data depends on what you want to do with it.
- All data structures have a natural set of operations that can be applied to it
 - ◆ Data access (retrieving, setting, removing)
 - ◆ Searching
- Each data structure implements these operations differently and, thus, have different efficiencies
- Big-O notation "measures" that efficiency
 - ◆ $O(n)$ is **slower** than $O(\log_2 n)$

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 10

C++ Arrays

- Representation in memory
- Static declaration
- Initialization
- Accessing array elements
- Stepping through an array
- Traps
- Arrays and functions

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 11

C++ Arrays

- Arrays are an example of a data structure
- Contiguous chunks of memory
- All chunks must be of the same data type (int, float, or whatever), called the **array base type**
- Each chunk occupies the same number of bytes in memory
- Static declaration:


```
int books[1000000]; // not 1,000,000 !
float dailyTemp[366];
Student cs12[100]; // Student is a custom
                  // defined struct or class
```
- Better to use named constants:


```
int books[MAX_NUM_BOOKS];
```

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 12

C++ Arrays

- Stepping through


```

                // Initializing...
                const int ARRAY_SIZE = 8;
                int even[ARRAY_SIZE];
                for (int k = 0; k < ARRAY_SIZE; k++)
                { even[k] = 2*(k + 1); }

                // Accessing...
                const int NUM_VOWELS = 5;
                char vowels[NUM_VOWELS] = { 'a', 'e', 'i', 'o', 'u' };
                for (int i = 0; i < NUM_VOWELS; i++)
                { cout << vowels[i] << endl; }
            
```

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 16

C++ Array Traps

- Index **always** starts with **0** and **always** ends with **(array size - 1)**
- C++ does **not** complain if you go past the end of the array... **major** source of bugs !
- Java is different in that respect: the Java compiler does complain

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 17

C++ Arrays and functions

- You can pass array elements as function arguments
 - ◆ by value


```
void f1(char v); // takes a char by value
```
 - ◆ by reference


```
void f2(char &v); // takes a char by reference
```

```

for (int i = 0; i < NUM_VOWELS; i++)
{
    f1(vowels[i]); // uses vowels[i], but cannot change it
    f2(vowels[i]); // uses vowels[i], and may change it
}
    
```

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 18

C++ Arrays and functions

- You can also pass the entire array as an argument to a function


```
void f3(char a[], int size); // takes an array argument
f3(vowels, NUM_VOWELS); // no [] here !!
```
- Array arguments behave *almost* like arguments passed by ref
 - ◆ Arguments passed by reference include size information
 - ◆ Array arguments do **not** include size information

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 19

C++ Arrays and functions

- Arguments passed by ref include size information


```
void f(char &c); // takes a char by reference
f(vowels[3]);
```

 - ◆ f receives the **address** of **vowels[0]** (*not* of **vowels[3]** !!!)
 - ◆ f is able to compute the address of **vowels[3]** since f knows how much memory the array base type occupies
 - ◆ Address of **vowels[3]** = address of **vowels[0]** + 3*size(char)

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 20

C++ Arrays and functions

- Array arguments do **not** include size information


```
void f(char a[]); // takes an array argument
f(vowels);
```

 - ◆ f receives the **address** of **vowels[0]**
 - ◆ f knows how much memory the array base type occupies, but that's it... it doesn't know where the array ends
 - ◆ Need to pass another argument:


```
void f3(char a[], int size); // also takes array size
f3(vowels, NUM_VOWELS);
```

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 21

C++ Arrays and functions

- Functions cannot return arrays
- You'll learn how to "return" arrays when we talk about pointers

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 22

Reminders

- If you have not read these yet, make sure to do so:
Savitch's sections
10.1, 10.2, 10.3
handout 2: **stack_frames.pdf**
- Check out Savitch's own slides for chapter 10 (available from cs12 page)

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 23
