

# Introduction to Data Structures: Arrays

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

[wagner@cs.ucr.edu](mailto: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 is 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!
- **Home Programming Project 1** has been out since Sunday
  - ◆ due date is **9 pm Mon Jan 20**

©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

---

---

---

---

---

---

---

---

**Introduction 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...

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

---

---

---

---

---

---

---

---

**Introduction 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...

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

---

---

---

---

---

---

---

---

**Introduction 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)

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

---

---

---

---

---

---

---

---

### MessyBookPile, Inc.

- This bookstore keeps its books all in one big pile, with no organization of any kind
- Your strategy 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 strategy, 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 7

---

---

---

---

---

---

---

---

### LongBookShelf, Inc.

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

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

---

---

---

---

---

---

---

---

### LongBookShelf, Inc.

- Another possible strategy 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 strategy
    - ★ If the book I want comes **after** the middle book, search the **second half** of the array using this same strategy
- This **does** make use of the fact that the books are sorted
- **Note:** recursive strategy ! (more on this next week)
- 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 9

---

---

---

---

---

---

---

---

## LongBookShelf, Inc.

- Another possible strategy 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 strategy
    - If the book I want comes **after** the middle book, search the **second half** of the array using this same strategy
- This **does** make use of the fact that the books are sorted
- **Note**: recursive strategy ! (more on this next week)
- 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 10

---

---

---

---

---

---

---

---

## Bottom line...

- The choice and efficiency of a strategy (called an **algorithm** in CS jargon) depends on how the data it accesses is organized
- Data structures are "containers" for the data your program manipulates, much like built-in data types (int, float, etc), but more powerful
- 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 11

---

---

---

---

---

---

---

---

## 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 12

---

---

---

---

---

---

---

---

## C++ Arrays

- 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 13

---

---

---

---

---

---

---

---

## C++ Arrays

Display 5.2, page 178

Display 5.2: An Array in Memory

int a[6];

Figure ©2002 Pearson Education, Inc. CS 12: Intro. Computer Science II • Lecture 3 14

---

---

---

---

---

---

---

---

## C++ Arrays

- Initialization
  - ◆ At declaration time
 

```
char vowels[5] = { 'a', 'e', 'i', 'o', 'u' };
int even[8] = { 2, 4, 6, 8, 10, 12, 14, 16 };
int odd[] = { 1, 3, 5, 7 }; // auto-sets size to 4
```
  - ◆ Stepping through
- Accessing array elements
 

```
char v = vowels[3]; // sets v to ???
even[0] + 5; // evaluates to ???
```

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 15

---

---

---

---

---

---

---

---

## C++ Arrays

- Initialization
  - ◆ At declaration time
 

```
char vowels[5] = { 'a', 'e', 'i', 'o', 'u' };
int even[8] = { 2, 4, 6, 8, 10, 12, 14, 16 };
int odd[] = { 1, 3, 5, 7 }; // auto-sets size to 4
```
  - ◆ Stepping through
- Accessing array elements
 

```
char v = vowels[3]; // sets v to 'o', not 'i' !!
even[0] + 5; // evaluates to 7
```

  - ◆ Array index **always** starts at **zero**
  - ◆ And **always** ends at (array size - 1)
- Two different uses for brackets [ ]

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

---

---

---

---

---

---

---

---

---

---

## 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 17

---

---

---

---

---

---

---

---

---

---

## C++ Arrays

- 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 complains)

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

---

---

---

---

---

---

---

---

---

---

## 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 19

---

---

---

---

---

---

---

---

## 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
  - ◆ Args passed by ref include size information
  - ◆ Array arguments do **not** include size information

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

---

---

---

---

---

---

---

---

## 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 21

---

---

---

---

---

---

---

---

## 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 the array size f3(vowels, NUM\_VOWELS);

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

---

---

---

---

---

---

---

---

## 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 23

---

---

---

---

---

---

---

---

## Reminders

- If you have not read these yet, make sure to do so:  
**Savitch's chapter 5**  
handout 2: **stack\_frames.pdf**
- You'll need them for home programming project 1
- Check out Savitch's own slides for chapter 5 (available from cs12 page)

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 3 24

---

---

---

---

---

---

---

---