

# CS 12: Intro. to Computer Science II

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>

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 1

---

---

---

---

---

---

---

---

## This Week's Topics

- **Very brief** review of CS 10
  - ◆ Variables
  - ◆ Operators: Arithmetic, Logical
  - ◆ Automatic type conversion
  - ◆ Flow of control
  - ◆ L-values and R-values
  - ◆ Functions
    - \* Call-by-value and call-by-reference
    - \* Function overloading
- Introduction to Data Structures: **Arrays**
- Searching through an array
  - ◆ Linear and Binary searches
  - ◆ First look at Big-Oh notation

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 2

---

---

---

---

---

---

---

---

## Review of CS 10

- Variables
- Operators: Arithmetic, Logical
- Automatic type conversion
- Flow of control
- L-values and R-values
- Functions
  - ◆ Call-by-value
  - ◆ Call-by-reference
  - ◆ Function overloading

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 3

---

---

---

---

---

---

---

---

## Variables

- Variables are “containers” for values
- Must be declared before they can be used
- Are type-specific: an **int** variable is not the same as a **float** variable, for example
- Values are stored in memory cells, occupying a machine dependent amount of bytes.
  - ◆ Eg: typically, an **int** takes **4 bytes**
- Examples:  
int n = 3;  
float radius = 0.54;

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 4

---

---

---

---

---

---

---

---

## Operators I: Arithmetic

- Let you manipulate your variables
- Arithmetic operators
  - ◆ things like +, -, \*, /, ++, --
  - ◆  $(a + 3)*(b++)$
  - ◆ **n++** ⇨ use **n**, then increment **n** by 1
  - ◆ **++n** ⇨ increment **n** by 1, then use **n**
  - ◆ Say that **a = b = 1**. What are the values of
    - $(a + 3)*(b++)$  and **b** ?
    - $(a + 3)*(++b)$  and **b** ?

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 5

---

---

---

---

---

---

---

---

## Operators I: Arithmetic

- Watch out for integer division !
  - ◆ 3 / 6 evaluates to **ZERO**
  - ◆ 3.0 / 6.0 evaluates to 0.5
  - ◆ 3 / 6.0 evaluates to 0.5 ⇨ auto conversion
  - ◆ 3.0 / 6 evaluates to 0.5 ⇨ auto conversion
- Same thing applies to integer *variables* (not just literals like 3 and 6)

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 6

---

---

---

---

---

---

---

---

## Operators II: Logical

- Logical operators
  - things like `==`, `!=`, `>`, `<`, `<=`, `>=`, `&&`, `||`
  - modern compilers let you write **and** for `&&` and **or** for `||`
  - Evaluate to a **bool** (**true** or **false**)
  - `3 > 5` evaluates to **false**
  - `3 != 5` evaluates to **true**
  - `x == 5` result depends on the value of `x`
  - `&&` (also **and**) is the *AND* operator
    - `(3 > 5) and (3 != 5)` results in **false**
  - `||` (also **or**) is the *OR* operator
    - `(3 > 5) or (3 != 5)` results in **true**

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 7

---

---

---

---

---

---

---

---

## Operators II: Logical

- Truth tables:

A	B	A and B	A or B
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 8

---

---

---

---

---

---

---

---

## Operators II: Logical

- Short-circuit evaluation
  - When evaluating **a and b and c and d**, where **a, b, c, d** are boolean expressions, there's no need to continue evaluating when a **false** is encountered
  - Eg: if **c** is **false**, then only **a, b,** and **c** are evaluated. **d** never gets to be evaluated.
  - When evaluating **a or b or c or d**, where **a, b, c, d** are boolean expressions, there's no need to continue evaluating when a **true** is encountered
  - Eg: if **c** is **true**, then only **a, b,** and **c** are evaluated. **d** never gets to be evaluated.
  - The same applies for any number of sub-expressions (I used only 4 just as an example).

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 9

---

---

---

---

---

---

---

---

### Operators II: Logical

- Short-circuit evaluation
  - Must be careful *never* to have a sub-expression whose evaluation changes something, because it may never be evaluated!
  - Eg:
    - `int x = 1;`
    - `bool result = (x == 3) and (++x > 0);`
  - One might think that the value of `x` is changed, but it is **not**! The sub-expression `(++x > 0)` was *never* evaluated because `(x == 3)` results in **false**.

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 10

---

---

---

---

---

---

---

---

### Operators II: Logical

- Order of evaluation
  - And** is like *multiplication*, or is like *addition*: **and** has *higher* precedence and so is evaluated *first*
  - `2 * 3 + 7` is **not** equal to **20**; it's equal to **13** because the multiplication is evaluated first
  - Likewise, in `a and b or c`, `a and b` is evaluated first, as in `(a and b) or c`
  - Just as we can write `2 * (3 + 7)` to force the addition to be evaluated first, we can also write `a and (b or c)` to force `b or c` to be evaluated first.

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 11

---

---

---

---

---

---

---

---

### Operators III: Assignment

- Assignment operator
  - `int z = 5;` declares `z`, sets its value to **5** and then returns that value
  - Thus, for eg, `(z = 7) == 2` returns **false**
  - Do **not** confuse `==` with `=`

```
if (z = 3)
{ /* do something */ }
```

is valid

`(z = 3)` returns `3`, which is interpreted as **true**, so the body of the if statement above will **always** execute. This is a **very common** mistake.

**only** `0`, `0.0`, and `null` are interpreted as **false**

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 12

---

---

---

---

---

---

---

---

## Automatic conversion

- If necessary, **chars** are converted to **shorts**, **shorts** to **ints**, **ints** to **longs**, **longs** to **floats**, and **floats** to **doubles**
- Example:
  - ♦ `char c = 'Z';`
  - ♦ `double result = 1.9 * (c + 5);`
  - ♦ `c` is converted to an **int**, added to **5**, the result is converted to a **double** and then multiplied by **1.9**. The result is a **double**.
  - ♦ This is an example of **bad code**, for at least **two** reasons. Can you guess them?
- **Any non-zero** numerical result appearing in a boolean expression is converted to **true**; **0** and **0.0** are converted to **false**

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 13

---

---

---

---

---

---

---

---

## Flow of control

- Things like **if**, **while**, **do**, **for**
- **if** (boolean expression)  
`{ /* do something */ }`
- **if** (boolean expression)  
`{ /* do something */ }`  
**else**  
`{ /* do something different */ }`
- **while** (boolean expression)  
`{ /* do something */ }`

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 14

---

---

---

---

---

---

---

---

## Flow of control

- **do**  
`{ /* something */ }`  
**while** (boolean expression);
- **for** (initializations; conditions; modifiers)  
`{ /* do something */ }`
- **for** (`int i = 0; i < MAX; ++i`)  
`{ cout << (2 * i) << endl; }`
- Watch out for `;` and `,` inside the `()`
- Use `++i` rather than `i++`, unless otherwise necessary

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 15

---

---

---

---

---

---

---

---

### Flow of control

- Use **standard conventions** and the **simplest** code that gets the job done
- Bad example:  

```
double sum = 0;  
int i = 0;  
do { sum = sum + a[i]; i++; }  
while (i < length);
```
- Good example:  

```
double sum = 0.0;  
for (int i = 0; i < length; ++i)  
{ sum += a[i]; }
```

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 16

---

---

---

---

---

---

---

---

### L-values and R-values

- Suppose we have `int x = 3;`
- `x == 5;` // does this make sense ?

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 17

---

---

---

---

---

---

---

---

### L-values and R-values

- Suppose we have `int x = 3;`
- `x == 5;` // does this make sense ? Yes
- `x = 5;` // and this ?

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 18

---

---

---

---

---

---

---

---

### L-values and R-values

- Suppose we have `int x = 3;`
- `x == 5;` // does this make sense ? Yes
- `x = 5;` // and this ? Yes
- `8 == x;` // how about this ?

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 19

---

---

---

---

---

---

---

---

### L-values and R-values

- Suppose we have `int x = 3;`
- `x == 5;` // does this make sense ? Yes
- `x = 5;` // and this ? Yes
- `8 == x;` // how about this ? Yes
- `8 = x;` // and this ?

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 20

---

---

---

---

---

---

---

---

### L-values and R-values

- Suppose we have `int x = 3;`
- `x == 5;` // does this make sense ? Yes
- `x = 5;` // and this ? Yes
- `8 == x;` // how about this ? Yes
- `8 = x;` // and this ? No ! Why not ?
- Because **8** is a **literal** value

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 21

---

---

---

---

---

---

---

---

## L-values and R-values

- `8 = x; // illegal !`
- You can't assign values to a literal value
- Literal values are **not L-values** - you cannot have them on the **left** side of an assignment
- Values that **can** be on the **left** side are called **L-values**, those which can be on the **right** side are called **R-values**.
- The distinction is important when you consider functions and their return values (more on this later in the course).

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 22

---

---

---

---

---

---

---

---

## Functions I

- `main()`
  - ◆ Every program **must** have one
  - ◆ It's the entry point of execution of your program
  - ◆ **Must** return an integer
  - ◆ `int main()`
  - ◆ Should return `0` if no problems are encountered
  - ◆ Traditionally returns an error code if problems are encountered

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 23

---

---

---

---

---

---

---

---

## Functions II

- `// function declarations`
- `void drawMonster(int monster_id, int x, int y)`
- `void moveMonster(int monster_id, int dx, int dy)`
- `// function definition`
- `int max(int a, int b)`

```
{
    if (a > b)
        return a;
    else
        return b;
}
```
- `// function call`
- `int z = max(r, s);`
- `drawMonster(troll, 25, 17);`

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 24

---

---

---

---

---

---

---

---

## Functions III

- Call by value
  - ◆ The **value** of the argument is **copied** into a local variable inside the function
  - ◆ `void f(int x, float v)`
- Call by reference
  - ◆ The **address in memory** of the argument is passed to the function
  - ◆ Lets you change the value of what's being passed to the function, from inside the function
  - ◆ `void f(int &x, float &v)`
  - ◆ `&` means "address of"

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 25

---

---

---

---

---

---

---

---

## A puzzle...

- How do you swap two **integer** variables ?
- The integers are stored in two variables, say, **x** and **y**

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 26

---

---

---

---

---

---

---

---

## No problem, right ?

- We could write this:

```
void swap(int x, int y)
{
    x = y;
    y = x;
}
```
- But it does **not** work ! Why not ?

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 27

---

---

---

---

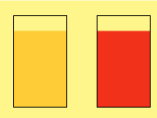
---

---


---

---

### A similar puzzle...



- Say you have a wine-glass full of beer and a beer-mug full of red wine.
- Now suppose that you want to swap the contents of the two containers.
- How do you do it?



1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 28

---

---

---

---

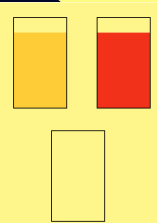
---

---

---

---

### A similar puzzle...



- Say you have a wine-glass full of beer and a beer-mug full of red wine.
- Now suppose that you want to swap the contents of the two containers.
- How do you do it? You need another container !

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 29

---

---

---

---

---

---

---


---

### No problem, right ?

- We could then write this:

```
void swap(int x, int y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

- But it **still** doesn't work ! Why not ?
- Inside the function body, **x** and **y** are **local** variables whose initial values are **copies** of the arguments



1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 30

---

---

---

---

---

---

---

---

## Call by reference

- We **must** write this:

```
void swap(int &x, int &y)
{
    int temp = x;
    x = y;
    y = temp;
}
```
- Now it **does** work!
- Inside the function body, **x** and **y** are **not** local variables; they represent the actual arguments
- When the function exits (returns), **x** and **y** have their values exchanged
- We'll see in detail how **call by reference** is accomplished when we study **activation frames** next week

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 31

---

---

---

---

---

---

---

---

## Function overloading

- **Signature** of a function
  - ◆ Name of the function
  - ◆ Number of arguments
  - ◆ Types of those arguments
  - ◆ Order of those arguments
- The signature of a function includes **neither** the function's return type **nor** the **names** of the params
- It's ok to define many functions with the same name, as long as they have different **signatures**:
- Examples:

```
void swap(int &x, int &y)
void swap(char &x, char &y)
void swap(float &x, float &y)
void swap(double &x, double &y)
```
- This gets boring, is error-prone, and duplicates code...
- Is there a way to define a single swap function that applies to *every* type?
- Yes! That's done with **Templates**. We'll meet them later in the course.

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 32

---

---

---

---

---

---

---

---

## 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**.

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 33

---

---

---

---

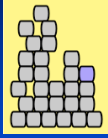
---

---

---

---

### 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:  **$O(n)$**

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 34

---

---

---

---

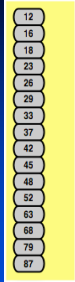
---

---

---

---

### LongBookShelf, Inc.



- This competitor bookstore keeps its books all in one very long **array, sorted by ISBN**

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 35

---

---

---

---

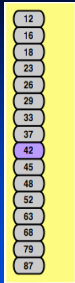
---

---

---

---

### 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

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 36

---

---

---

---

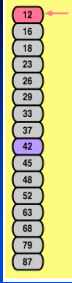
---

---

---

---

### 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

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 37

---

---

---

---

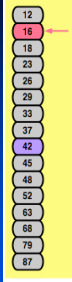
---

---

---

---

### 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

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 38

---

---

---

---

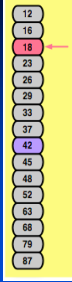
---

---

---

---

### 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

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 39

---

---

---

---

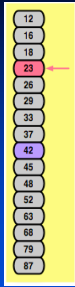
---

---

---

---

### 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

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 40

---

---

---

---

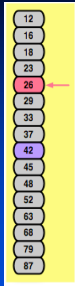
---

---

---

---

### 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

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 41

---

---

---

---

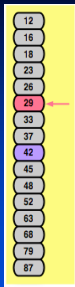
---

---

---

---

### 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

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 42

---

---

---

---

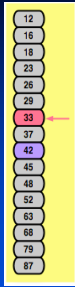
---

---

---

---

### 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

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 43

---

---

---

---

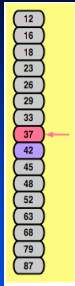
---

---

---

---

### 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

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 44

---

---

---

---

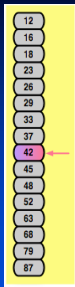
---

---

---

---

### 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

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 45

---

---

---

---

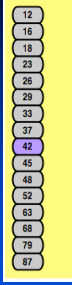
---

---

---

---

### 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

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 46

---

---

---

---

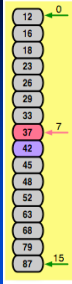
---

---

---

---

### 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

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 47

---

---

---

---

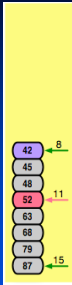
---

---

---

---

### 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

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 48

---

---

---

---

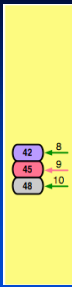
---

---

---

---

## 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

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 49

---

---

---

---

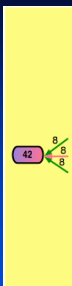
---

---

---

---

## 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 = ?**

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 50

---

---

---

---

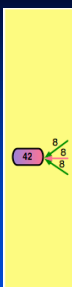
---

---

---

---

## 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  $\log_2(1,000,000) \approx 20$  ]

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 51

---

---

---

---

---

---

---

---

**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-Oh notation "measures" that efficiency
  - ◆  $O(n)$  is **slower** than  $O(\log_2 n)$

**You'll learn all about *Data Structures and Algorithms* in CS 14**

1.2.1 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 2 52

---

---

---

---

---

---

---

---