Welcome!

Program Design and Analysis II for Engineers
CompSci 100E
Soc Psy 126
W, F 2:50-4:05

Professor Jeff Forbes

http://www.cs.duke.edu/courses/spring11/cps100e/forbes
http://www.cs.duke.edu/forbes
Computer Science in a Nutshell?
Computer Science in a Nutshell?
Computer Science in a Nutshell?
Computer Science in a Nutshell?

- What is Computer Science?

- What are some of the subfields of CS?

- What does a Computer Scientist do? Lots of scientists and engineers

- What does a programmer do?
What is Computer Science?

What is it that distinguishes it from the separate subjects with which it is related? What is the linking thread which gathers these disparate branches into a single discipline? My answer to these questions is simple --- *it is the art of programming a computer*. It is the art of designing efficient and elegant methods of getting a computer to solve problems, theoretical or practical, small or large, simple or complex.

C.A.R. (Tony) Hoare
Programming != Computer Science

- What is the nature of intelligence? How can one predict the performance of a complex system? What is the nature of human cognition? Does the natural world 'compute'?

- It is the interplay between such fundamental challenges and the human condition that makes computer science so interesting. The results from even the most esoteric computer science research programs often have widespread practical impact. Computer security depends upon the innovations in mathematics. Your Google search for a friend depends on state-of-the-art distributed computing systems, algorithms, and artificial intelligence.

Course Overview

- **Active Lectures, Labs, Quizzes, Programs**
  - Labs based on questions given out in previous week
    - Hands-on practice with programming
    - Discuss answers, answer new questions
    - More opportunities for questions to be answered.
  - Active Lectures based on readings, questions, programs
    - In-class questions used to ensure understanding
  - Programs
    - Theory and practice of data structures and OO programming
    - Fun, practical, tiring, ...

- **Exams/Tests**
  - Semester: open book/note*
  - Final: open book/note
Is this course for you?

● **You should already know...**
  - Basics of programming: variables, conditionals, loops, arrays/lists/vectors
  - Prereq: CompSci 6, EGR 53, AP CS, Programming experience, or ambition, ...

● **Don’t need to know**
  - Java, but some already do

● **This course introduces**
  - Java & Object Oriented Programming
  - Data structures and abstraction
  - Algorithm design and analysis
  - The awesome ability to write programs to solve your problems and amaze your peers
What’s this course really about?

● **Solving problems**
  - Understanding problem statements
  - Designing solutions to the problems
  - Implementing the problem solution in a program
  - Understanding and testing the results
  - Documenting and presenting our results

● **How will we do it?**
  - Written classwork and homework
  - Programming projects
  - Algorithmic Programming Testing
Tradeoffs

Programming, design, algorithmic, data-structural

Simple, elegant, quick, efficient: what are our goals in programming? Don’t worry about getting it right the first time.

Fast programs, small programs.
Runtime, space, your time, CPU time... Time vs. space

How do we decide what tradeoffs are important? Tension between generality, simplicity, elegance, ...
Anyway, I thought you'd be interested to know that in 2 of the 5 technical interviews I had, I recognized problems from CS courses at Duke. Specifically, they asked me to write algorithms for the "intersection of two sets" problem and a variation of the "boggle" problem. I thought that was pretty interesting. ... For what it's worth for any of your students interviewing, I prepared for the interview mostly by practicing APT problems from the Duke CS100 course page, and I felt that that prepared me very well for about 80% of the questions that were asked. It certainly helped me get into the mindset of the types of things they ask, especially after a few years of being away from those types of algorithms.

- Duke CS ‘07 alum
Languages

Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do. - Donald Knuth

- **Machine languages.**
- **Natural languages.**
- **High-level programming languages.**

- Kids Make Nutritious Snacks.
- Red Tape Holds Up New Bridge.
- Police Squad Helps Dog Bite Victim.
- Local High School Dropouts Cut in Half.

[real newspaper headlines, compiled by Rich Pattis]

- Pictures
- Natural language (English)
- Pseudo-code
- Specific high-level programming language
- Machine Language

More precisely expressed
More easily expressed

CompSci 100e
© Sedgewick & Wayne
Why Java?

- **Java features.**
  - Widely used.
  - Widely available.
  - Embraces full set of modern abstractions.
  - Variety of automatic checks for mistakes in programs.

- **Java economy.**
  - Mars rover.
  - Cell phones.
  - Blu-ray Discs.
  - Web servers.
  - Medical devices.
  - Supercomputing.
  - ...

James Gosling
http://java.net/jag
Why Java?

- Java features.
  - Widely used.
  - Widely available.
  - Embraces full set of modern abstractions.
  - Variety of automatic checks for mistakes in programs.
  - Buzzword-enabled

  "Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high performance, multi-threaded, and dynamic language"

- Caveat. No perfect language.

- Our approach.
  - Minimal subset of Java.
  - Develop general programming skills that are applicable to: C, C++, C#, Perl, Python, Ruby, Matlab, Fortran, Fortress, ...
Programming in Java

- **Programming in Java.**
  - *Create* the program by typing it into a text editor, and save it as `HelloWorld.java`

```java
/*******************************************
* Prints "Hello, World"
* Everyone's first Java program.
*******************************************/

```public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, World");
  }
}
```

**HelloWorld.java**
Programming in Java

- *Programming in Java.*
  - Create the program by typing it into a text editor, and save it as `HelloWorld.java`
  - **Compile** it using Eclipse or by typing at the command-line: `javac HelloWorld.java`

  ```command-line
  javac HelloWorld.java
  ```

  (or click the Save button in Eclipse)

- This creates a Java bytecode file named: `HelloWorld.class`
HelloWorld.class

`sout` 1 0 0 27 `f`

`println` 0 0 007 `null`

`println` 0 0 007 `null`

`println` 0 0 007 `null`
## A Rich Subset of the Java Language

### Built-In Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>double</td>
</tr>
<tr>
<td>long</td>
<td>String</td>
</tr>
<tr>
<td>char</td>
<td>boolean</td>
</tr>
</tbody>
</table>

### System

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System.out.println()</code></td>
</tr>
<tr>
<td><code>System.out.print()</code></td>
</tr>
<tr>
<td><code>System.out.printf()</code></td>
</tr>
</tbody>
</table>

### Math Library

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Math.sin()</code></td>
</tr>
<tr>
<td><code>Math.cos()</code></td>
</tr>
<tr>
<td><code>Math.log()</code></td>
</tr>
<tr>
<td><code>Math.exp()</code></td>
</tr>
<tr>
<td><code>Math.sqrt()</code></td>
</tr>
<tr>
<td><code>Math.pow()</code></td>
</tr>
<tr>
<td><code>Math.min()</code></td>
</tr>
<tr>
<td><code>Math.max()</code></td>
</tr>
<tr>
<td><code>Math.abs()</code></td>
</tr>
<tr>
<td><code>Math.PI</code></td>
</tr>
<tr>
<td><code>Math.log()</code></td>
</tr>
<tr>
<td><code>Math.sin()</code></td>
</tr>
<tr>
<td><code>Math.cos()</code></td>
</tr>
</tbody>
</table>

### Flow Control

<table>
<thead>
<tr>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>if</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>for</td>
</tr>
<tr>
<td>while</td>
</tr>
</tbody>
</table>

### Parsing

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Integer.parseInt()</code></td>
</tr>
<tr>
<td><code>Double.parseDouble()</code></td>
</tr>
</tbody>
</table>

### Boolean

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
</tr>
<tr>
<td>false</td>
</tr>
<tr>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>!</td>
</tr>
</tbody>
</table>

### Punctuation

<table>
<thead>
<tr>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>(</td>
</tr>
<tr>
<td>)</td>
</tr>
<tr>
<td>,</td>
</tr>
<tr>
<td>;</td>
</tr>
</tbody>
</table>

### String

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
</tr>
<tr>
<td>length()</td>
</tr>
<tr>
<td>charAt()</td>
</tr>
</tbody>
</table>

### Arrays

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[i]</td>
</tr>
<tr>
<td>new</td>
</tr>
<tr>
<td>a.length</td>
</tr>
</tbody>
</table>

### Objects

<table>
<thead>
<tr>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>this</td>
</tr>
<tr>
<td>class</td>
</tr>
<tr>
<td>static</td>
</tr>
<tr>
<td>public</td>
</tr>
<tr>
<td>private</td>
</tr>
<tr>
<td>toString()</td>
</tr>
<tr>
<td>equals()</td>
</tr>
<tr>
<td>new</td>
</tr>
<tr>
<td>main()</td>
</tr>
</tbody>
</table>
Problem Solving and Programming

● How many words are in a file?
  ➢ What’s a word?
  ➢ What’s a file?
  ➢ How do we solve this: simply, quickly, ...?
    • What’s the best we can do? Constraints?

● How many different/unique words are in a file?
  ➢ How is this related to previous task?

● How many words do two files have in common?
  ➢ Spell-checking, stemming, Did you mean ..?
● How many codons does DNA have in common?
Problem 1: Text Clouds

- **Text clouds: A simple yet powerful idea**
  - Visualization of most frequently occurring words within some body of text
  - Great visual/statistic: [http://chir.ag/phernalia/preztags/](http://chir.ag/phernalia/preztags/)
    - What information is stored in the URL of the NYTimes site above?
  - Color or font size indicates word frequency

- **What is involved with generating text clouds?**
  - Steps? Issues?
  - See `SimpleWordCount.java` and `SimpleCloudMaker.java`
Problem 2: Data processing

- Scan a large (~ \(10^7\) bytes) file
- Print the words together with counts of how often they occur
- Need more specification?

- How do you do it?

- What is we only wanted the top \(k\) (say 20) words?
Possible solutions

1. Use heavy duty data structures (Knuth)
   - Hash tries implementation
   - Randomized placement
   - Lots o' pointers
   - Several pages

2. UNIX shell script (Doug Mclroy)
   ```bash
   tr -cs "[:alpha:]" "[\n*]" < FILE |
   sort |
   uniq -c |
   sort -n -r -k 1,1
   ```

3. See SimpleWordCount.java
   - Which is better?
     - K.I.S.?