Today’s topics

Designing and Implementing Algorithms
  Problem solving
  Pseudocode
  Java
  Syntax and Grammars

Upcoming
  More Java

Acknowledgement
  Marti Hearst, UC Berkeley
  David Smith, Georgia tech

Reading
  Computer Science, Chapter 5
  Great Ideas, Chapter 2

Problem Solving

Programming is a strenuous exercise in problem solving
  ● Understand the problem
    ➢ What are its parts? unknown, data, condition
    ➢ Does the problem make sense? Is it feasible?
    ➢ Think about the problem, get a sense of what it needs
  ● Make a plan
    ➢ Find the connection between givens and result
    ➢ What kind of problem is it? Is it familiar?
    ➢ Think about generalizations, specializations, variants
  ● Carry out the plan
    ➢ Check each step
  ● Examine the result
    ➢ Does it make sense?

Back of the envelope calculations

http://www.vendian.org/envelope/
  ● Engineering technique to approximate and check answers
    ➢ Two answers are better than one
    ➢ Quick checks
    ➢ Rules of thumb
    ➢ Practice
  ● Ad claims that salesperson drove 100,000 miles in a year. True?
  ● Newspaper article states that a United States quarter dollar coin has “an average life of 30 years.” How can you check that claim?

Why “back of the envelope” estimates?

  ● Often need to make rapid estimates
    ➢ to eliminate candidate solutions
    ➢ establish feasibility
    ➢ sketch out potential trade-offs
  ● Most remember key numbers related to their field, not every detail
  ● Hence we need to estimate
    ➢ which numbers are important
    ➢ values of numbers needed
    ➢ how to perform the calculation
  ● Emphasis is on “order of magnitude” estimates
    ➢ to nearest factor of 10 (or 2)
Orders of Magnitude

- How far away is home? Is it more like 1, or 10, or 100 miles?
  - Probably do not know exactly
  - Is it approximately "a couple", or "a few", or "a lot"
  - Estimate based on powers rather than multiples of 10
- How tall is your dorm? More like 1, 10, 100, 1000 feet?
  - 1 foot tall is like a doll house, so that’s out
  - What do we know that is about 10 feet big? Hmm... People
  - If building is a couple of people high, 10 sounds good.
  - But that means 1000, would be 100 people high, so that’s out
  - So 10 or 100 depending on how many people tall the building is
- Use orders of magnitude as brackets to find reasonable range

Example: How many piano tuners in NYC

- Approximately how many people are in New York City?
  - 10,000,000
- Does every individual own a piano?
  - No
- Reasonable to assert “individuals do not own pianos; families do”?
  - Yes
- About how many families are there in a city of 10 million people?
  - Perhaps there are 2,000,000 families
- Does every family own a piano?
  - No
- Perhaps one out of every five does
  - That would mean there are about 400,000 pianos in NYC

Example: Piano Tuners continued

- How many piano tuners are needed for 400,000 pianos?
  - Some people never get around to tuning their piano
  - Some people tune their piano every month
  - Assume "on the average" every piano gets tuned once a year, then there are 400,000 every year
- How many piano tunings can one piano tuner do?
  - Assume that average piano tuner can tune four pianos a day
  - Assume that there are 200 working days per year
  - That means every tuner can tune about 800 pianos per year
- How many piano tuners are needed in NYC?
  - Number of tuners is approximately 400,000/800 or 500

Example: Piano Tuners summary

- “Back of the Envelope” estimates have
  - Formulas: provide roadmap to upcoming calculations
  - Estimates: brief justification of approximations in formula
  - Calculations: estimates and known facts are use in formula

Piano Tuner example

- Formula:
  - \(#\text{ tuners} = \#\text{ pianos} \times \#\text{ repairs} / \#\text{ repairs per day} \times \#\text{ days}\)
- Estimates
  - \(#\text{ pianos} \approx 400,000 (20\% \text{ of 2,000,000 families own pianos})\)
  - \(#\text{ repairs} \approx 1 \text{ per piano (some many, some none)}\)
  - \(#\text{ repairs per day} \approx 4\)
  - \(#\text{working days} \approx 200 (5 \times 50 - \text{ vacation, sickness})\)
- Calculation
  - \(#\text{ tuners} \approx (400,000 \times 1) / (4 \times 200) = 500\)
Estimation General Principles

- Recall Einstein’s famous advice
  - Everything should be made as simple as possible, but no simpler
- Do not worry about constant factors of 2, π, etc.
  - Round to “easy” number or nearest order of magnitude
- Guess numbers you do not know
  - Within bounds of common sense (accuracy increases with experience)
- Adjust geometry, etc., to suit you
  - Assume a cow is spherical if it helps
- Extrapolate from what you do know
  - Use ratios to assume unknown value is similar to known quantity
- Apply a ‘plausibility’ filter
  - If answer seems unbelievable, it probably is
  - Can usually set range of reasonable values that indicates major mistake (e.g., speed cannot be faster than light!)

Central role of algorithms in CS

Properties of good algorithms

- Good algorithms must be
  - Correct
  - Complete
  - Precise
  - Unambiguous
- And should be
  - Efficient
  - Simple
  - Contain levels of abstraction

An algorithm is an ordered set of unambiguous, executable steps, defining a terminating process.
Algorithms

- Hand-waving not allowed!

- Specifying algorithms requires you to say what is really involved in making it work.

- Example:
  - How does a computer work?
  - Hand-wave: zeros & ones
  - Real answer: see later part of class.

- You learn to know when you don’t know
  - “I know nothing except the fact of my ignorance.”
  - Socrates, from Diogenes Laertius, Lives of Eminent Philosophers

Describing Algorithms

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

More easily expressed

More precise

Pseudocode

- A shorthand for specifying algorithms
- Leaves out the implementation details
- Leaves in the essence of the algorithm

```pseudocode
procedure Greetings
    Count ← 3;
    while (Count < 0) do
        (print the message “Hello” and Count ← Count +1)

What does this algorithm do?

How many times does it print Hello?
```

Sequential search

```pseudocode
procedure Search (List, TargetValue)
    if (List empty)
        then (Declare search a failure)
    else (Select the first entry in List to be TestEntry;)
        while (TargetValue > TestEntry and there remain entries to be considered)
            do (Select the next entry in List as TestEntry.)
        if (TargetValue = TestEntry)
            then (Declare search a success.)
        else (Declare search a failure.)
    end if
```
Picking courses

1. Make a list of courses you want to register for, in order of priority
2. Start with empty schedule. Number of courses = 0.
3. Choose highest priority class on list.
4. If the chosen class is not full and its class time does not conflict with classes already scheduled, then register for the class (2 steps):
   1. Add the class to the schedule
   2. Increment the number of classes scheduled
5. Cross that class off of your list.
6. Repeat steps 3 through 5 until the number of classes scheduled is $\geq 4$, or until all classes have been crossed out.
7. Stop.

Flowcharts

1. Make list of classes you want to take
2. Choose highest priority class on list
3. Add the class to your schedule. Increment Num Classes.
4. Cross the class off your list.
5. Is this class full?
   - Yes: Go back to step 3
   - No: Go back to step 3
6. Is there a time conflict?
   - Yes: Go back to step 3
   - No: Go back to step 3
7. Num Classes $\geq 4$?
   - Yes: Stop
   - No: More classes on list?
   - Yes: Go back to step 3
   - No: Stop

Programming Primitive Operations

- Assign a value to a variable
- Call a method
- Arithmetic operation
- Comparing two numbers
- Indexing into an array
- Following an object reference
- Returning from a method

Components of Computing Algorithms

Any computing algorithm will have AT MOST five kinds of components:

- Data structures to hold data
- Instructions change data values
- Conditional expressions to make decisions
- Control structures to act on decisions
- Modules to make the algorithm manageable by abstraction, i.e., grouping related components
Java!

- Java is a buzzword-enabled language
- From Sun (the developers of Java),
  "Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high performance, multi-threaded, and dynamic language."

- What do all of those terms mean?

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

- A programming language
  - A vocabulary and set of syntactical (grammatical) rules for instructing a computer to perform specific tasks
  - You can do most anything in any programming language
  - A particular language encourages one to do things in a certain way

- A Question for the course: What makes a good language?

Based on popular languages called C and C++
- C: old, pretty bare bones language
- C++: newer, more complicated language
- Start from C and add some of C++’s more useful features
  - From Gosling, the creator, “Java omits many rarely used, poorly understood, confusing features of C++ that in our experience bring more grief than benefits.”

- Question: Is Java really all that simple?
A distributed system is one where multiple separate computer systems are involved.

- Electronic card catalogs
- The web

Java was designed for the web.

Question: What are examples of a distributed task in your lives?

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

- A high-level language
- High-level languages must be translated to a computer’s native tongue, machine language
- Interpreted high-level languages are translated to an intermediate form and then converted to machine language and run

Why?
- We’ll learn more about this later

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

Security: techniques that ensure that data stored on a computer cannot be read or compromised.

- A program is running on your computer. What is to stop it from erasing all of your data, accidentally or otherwise?
- Question: Is Java really all that secure?

Programs will have errors, but a good program degrades reasonably.

A robust program may not do exactly what it is supposed to do, but it should not bring down other unrelated programs down with it.

Question: Give me an example of a non-robust program you have seen?
“Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high performance, multi-threaded, and dynamic language.”

- A language is architecture-neutral if it does not prefer a particular type of computer architectures
- E.g. The Macintosh processor family (PowerPC) and the PC (x86-Pentium) family have their own respective strengths and weaknesses. It is not too hard to construct a program that will run faster on one than on the other.

- A particular program is never entirely architecture neutral though
- Question: When is being architecturally neutral a bad thing?

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

- A program is portable if it will work the same (roughly) on many different computer systems
- HTML is also platform-independent or portable
- A whole lot of effort is currently spent porting non-portable code

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

- Performance: speed in completing some task
- Performance is everything to most computer and software manufacturers.

- Story:
  - If the transportation industry kept up with the computer industry, one would be able to now buy a Roll Royce that could drive across country in 5 minutes for $35.
- Rebuttal:
  - It would crash once a week, killing everyone on board.

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

- A thread is a part of the program that can operate independently of its other parts
- Multi-threaded programs can do multiple things at once
  - e.g. download a file from the web while still looking at other web pages

- Question: What is the problem with multiple agents working at the same time?
  - Synchronization
“Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high performance, multi-threaded, and dynamic language.”

- Dynamic refers to actions that take place at the moment they are needed rather than in advance
  - Antonym: static
- A dynamic program can
  - Ask for more or less resources as it runs
  - Use the most recent version of some code that is available
- Question: Why is being dynamic a good thing?

A Java Program

```java
import java.awt.*;
public class HelloWorld extends java.applet.Applet {
    TextField m1;
    public void init() {
        m1 = new TextField(60);
        m1.SetText("Hello World");
        add(m1);
    }
}
```

Definitions

- **Algorithm**: ordered set of unambiguous executable steps, defining a terminating process
- **Program**: instructions executed by a computer
- **Applet**: Java program that is executed in a program such as appletviewer or a Java-enabled web browser
- **Class**: family of components sharing common characteristics consisting of:
  - Data: information
  - Method: functionality
- **Object**: instance of a class
- **Variable**: represent value stored in computer memory. A variable must be defined or declared before being used
  - Sometimes synonymous with object

Grammar

- **English and other natural languages have structure**
  - `<S> => <NOUN-PHRASE> <VERB-PHRASE>
   => <NOUN> | <ARTICLE> <NOUN> | <PP>
   <VERB-PHRASE> => <VERB> | <VERB> <NOUN-PHRASE>
   <NOUN> => DOG | FLEAS | PERSON | ...
   <VERB> => RAN | BIT | ...
  - Process of taking sentence and fitting it to grammar is called **parsing**
  - DOG BIT PERSON
  - `<S> => <NOUN> <VERB> <NOUN>
  - `<NOUN-PHRASE> <VERB-PHRASE>`
  - `<S>`
- **Parsing English is complex because of context dependence**
Formal specifications

- Need a precise notation of syntax of a language
- Grammars can be used for generation and also can be used
- Context-free grammars
  
  <name> => sequence of letters and/or digits that begins with a letter
  <name> => guessB
  <name> => msg2

- Substitute as many times as necessary. All legal statements can be generated this way
  
  - Want person = firstn + " " + lastn;
  - How do we get this from our grammar?

A Grammar for Java

- Need a set of rules
- Our first one was a good start:
  
  - <name> => any string of alphanumeric symbols that begins with a letter
- Let’s add something to define a simple statement:
  
  - <statement> => <name> = <expression> ;
- And then work on the details:
  
  - <expression> => <string-expression> | <int-expression> | <oth-expression>
  - <string-expression> => <string-expression>
  - <string-expression> => <string>
  - <string-expression> => <expression>
  - <string> => "any sequence of characters"

A Simple Statement

- Now have enough to generate a statement like: `msg = "hello" ;`
  
  - Start with:
    <statement> => <name> = <expression> ;
  - Then using: <name> => any string of alphanumeric symbols that begins with a letter
    
    msg = <expression> ;
  - Then, using: <expression> => <string-expression> | <int-expression> | <oth-expression>
    
    msg = <string-expression> ;
  - Using: <string-expression> => <string>
    
    msg = <string> ;
  - Using: <string> => "any sequence of characters"
    
    msg = "hello" ;

A Grammar for Java

- Including more rules to describe programs we have:
  
  1. <name> => any string of alphanumeric symbols that begins with a letter
  2. <statement> => <name> = <expression> ;
  3. <statement> => <name> = new <class> (<arguments>) ;
  4. <statement> => <name> . <method> (<arguments>) ; | <method> (<arguments>) ;
  5. <arguments> => possibly empty list of <expression>s separated by commas
  6. <expression> => <string-expression> | <int-expression> | <oth-expression>
  7. <string-expression> => <string-expression> + <string-expression>
  8. <string-expression> => <string>
  9. <string> => "any sequence of characters"
 10. <string> = <name>
Using our Grammar

- Use this to generate: `person = firstn + " " + lastn;`

<table>
<thead>
<tr>
<th>Rule</th>
<th>Statement being Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>#2</td>
<td><code>&lt;statement&gt; =&gt; &lt;name&gt; = &lt;expression&gt;;</code></td>
</tr>
<tr>
<td>1</td>
<td><code>&lt;statement&gt; =&gt; person = &lt;expression&gt;;</code></td>
</tr>
<tr>
<td>6</td>
<td><code>&lt;statement&gt; =&gt; person = &lt;str-expression&gt;;</code></td>
</tr>
<tr>
<td>7</td>
<td><code>&lt;statement&gt; =&gt; person = &lt;str-expression&gt; + &lt;str-expression&gt;;</code></td>
</tr>
<tr>
<td>8</td>
<td><code>&lt;statement&gt; =&gt; person = &lt;string&gt; + &lt;str-expression&gt;;</code></td>
</tr>
<tr>
<td>10</td>
<td><code>&lt;statement&gt; =&gt; person = &lt;name&gt; + &lt;str-expression&gt;;</code></td>
</tr>
<tr>
<td>1</td>
<td><code>&lt;statement&gt; =&gt; &lt;statement&gt; =&gt; person = firstn + &quot; &quot; + lastn;</code></td>
</tr>
</tbody>
</table>

Proving Grammatical Correctness

- Why go through the process we went through?
  - Shows that desired statement can be generated from this grammar
- Actually proves that the statement is grammatically correct!
  - Same rigor as a mathematical proof
- (Doesn’t prove that logic is correct, though)

- Actually need more rules to handle the level of Java we’ve covered so far
  - Summary of rules shown on pages 78-79 of *Great Ideas*
  - Also give an example for a complete applet
  - Too long to go through in class – Please Read!