Today’s topics

- Algorithms and pseudocode
- Notes adapted from Marti Hearst at UC Berkeley and David Smith at Georgia Tech

Upcoming
- Beginning Java

Reading
- “Cyberspace as a Human Right”, William McIver
- Back of the Envelope Calculations
- Chapter 2 in Great Ideas

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.

What’s wrong with this algorithm?

(From back of shampoo bottle)

Directions:
- Wet Hair
- Apply a small amount of shampoo
- Lather
- Rinse
- Repeat
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

```
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

```
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 >= 4, or until all classes have been crossed out.
7. Stop.

Flowcharts

Begin

Make list of classes you want to take

Num Classes = 0

Choose highest priority class on list

yes

Is this class full?

no

Is there a time conflict?

no

Add the class to your schedule. Increment Num Classes.

no

Cross the class off your list.

yes

Num Classes >= 4?

no

More classes on list?

End

Origami as Algorithm

From Brookshear; Copyright 2003 Pearson Education
Origami primitives

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turn paper over</td>
<td>Distinguishes between different sides of paper</td>
</tr>
<tr>
<td>as in</td>
<td></td>
</tr>
<tr>
<td>Shade one side of paper</td>
<td></td>
</tr>
<tr>
<td>as in</td>
<td></td>
</tr>
<tr>
<td>Represents a valley fold</td>
<td></td>
</tr>
<tr>
<td>so that</td>
<td></td>
</tr>
</tbody>
</table>

Represent a mountain fold

so that

Fold over

so that

Push in

so that

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