CS3:
Introduction to Symbolic Programming

Lecture 3:
Review of conditionals
The “Difference between dates” case study

Spring 2008

Nate Titterton
nate@berkeley.edu

Announcements

- **Nate's office hours (this week only) :**
  - **Wednesday, 10-12, 329 Soda**
  - **usually they are W 2-4... Better?**

- **Readers (graders) are coming up to speed this week, so look for things to be graded soon…**

A video resource

- **http://wla.berkeley.edu**

Weiner lecture archives

- **The "course" is an earlier CS3**
  - **Different emphasis; early lectures may work better than later ones**
  - **Very different lab experience**
  - **Same book**

Schedule

2	Jan 28-Feb1	Lecture: Introduction, Review, Conditionals Reading: Simply Scheme, ch. 3-6 Lab: (1) Conditionals and booleans (2) Words/sentences and conditionals
3	Feb 4-Feb 8	Lecture: Conditionals, Case Studies Reading: "Difference between Dates" case study, in the reader Lab: Explore "Difference between Dates“ Start on Miniproject #1
4	Feb 11-15	Lecture: DbD, recursion Reading (Thur/Fri): Simply Scheme chap. 11 Lab: (1) Miniproject 1 (2) Recursion
5	Feb 18-22	Lecture: Recursion Lab: Recursion, Recursion, Recursion

Concepts from last week (1/4)

1. Conditionals
   - `cond` and `if`
   - These are special forms, and don't follow the standard rules of evaluation
2. Booleans
   - truth (#t, or anything) and non-truth (#f)
3. logical operators
   - `and`, `or`, `not`
4. Predicates
   - procedures that return booleans
   - (These end in a ? usually: odd?, vowel?, …)

Concepts from last week (2/4)

- **Writing conditionals using only `and/or` or `if/cond`.**

- **Organizing a series of conditionals**

Concepts from last week (3/4)

- **Testing**
  - There is much more to programming than writing code. *Testing* is crucial, and an emphasis of this course
    - Analysis
    - Debugging
    - Maintenance.
    - "Design"
  - Testing is an art (there is no one right way)
    - boundary cases, helper procedures, etc.

Concepts from last week (4/4)

- **Helper procedures**
  - Choosing when to write helper procedures is an … art. There is no one right way.
  - Issues include
    - Readability – clear and verbose are usually better than complex and brief
    - Maintenance – fixing or adding to code later
  - This is an important skill in programming, and one you will need to focus on.

Functional abstraction

- **Abstraction helps make programs understandable by simplifying them.**
  - **By letting the programmer or maintainer ignore details about a task at hand**
  - **Helper functions, when done correctly, do this**

What does it mean to “understand a program” ?

This week: Case Studies

- **Reading!?**
- **A case study…**
  - **starts with a problem statement**
  - **ends with a solution**
  - **in between, a story, or narrative**
  - ***How a program comes to be***
- **You will write "day-span", which calculates the number of days between two dates in a year**

You need to read this!

- **The lab will cover the case study through a variety of activities.**
- **We just may base exam questions on it**
- **It will make you a better programmer!**
  4 out of 5 educational researchers say so.

Some important points

- **There is a large "dead-end" in this text**
  - **Like occur in many programming projects**
  - **Good "style" helps minimize the impacts of these**

- **There is (often) a difference between good algorithms and between human thinking**

Miniproject 1

- **By the end of the week, you will start on miniproject 1:**

  - **write `century-day-span`, extending the `day-span` program to correctly handle dates in (possibly) different years.**

  - **Consider a central lesson of the case study: there are easier and harder ways to solve problems. Choose easier.**

This is your first large program

Use helper functions

- Break out self-contained tasks into helper procedures: they should be easy to name.
- If you can get your main procedure to read like English, you are doing well.

- **Test, and test some more.**
  - Remember to put test cases above each helper procedure.
- **Reuse code that you have already written**
- **Add comments!**
  - Above each procedure, at least.
  - Within some `cond` cases, additionally.