CS3: Introduction to Symbolic Programming

Lecture 3:
Review of the first two weeks
The “Difference between dates” case study

Fall 2007

Nate Titterton
nate@berkeley.edu

Announcements

- Nate's office hours, for next week:
  - Wednesday, 2-4
  - 329 Soda
- Readers are coming up to speed this week, so look for things to be graded soon…
- Check out the Weiner lecture archives
  - http://wla.berkeley.edu
  - Video lectures and notes from an earlier version of CS3 (still mostly relevant in the earlier weeks)

Schedule

2	Sep 3-7	Lecture: Introduction, Review, Conditionals Reading: Simply Scheme, ch. 3-6 Lab: Conditionals
3	Sep 10-14	Lecture: Conditionals, Case Studies Reading: "Difference between Dates" case study, in the reader (first version) Lab: Explore "Difference between Dates“ Start miniproject 1
4	Sep 17-21	Lecture: Data abstraction in DbD Lab: Finish miniproject 1 Begin recursion
5	Sep 24-28	Lecture: Recursion Lab: More complex recursion
6	Oct 1-4	Lecture: *Midterm 1* Lab: Advanced recursion

Concepts from first two weeks (1/3)

1. How scheme evaluates input
2. Words and sentences
   - Sentences as a "container"
3. Conditionals
   - `cond` and `if` are special forms
   - booleans

     truth (`#t`, or anything) and *non-truth* (`#f`)
   - logical operators

     `and`, `or`, `not`
   - predicates

     procedures that return booleans

     (These end in a ? usually: odd?, vowel?, …)

Concepts from last week (2/3)

1. **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 (3/3)

1. **Helper procedures**
   - Choosing when to write helper procedures is an … art.   There is no one right way.
   - This is an important skill in programming, and one you will need to focus on.

Functional abstraction

- <u>Abstraction</u> 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

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

Reminder

- This week, (I think) I will leave in many SchemeHandler activities.
  - Many of these you can do in emacs. Some you can't.
  - Remember, try using the unix command `clearcache`, and then restart firefox, if you are having trouble.
  - Let your TAs help you – we are trying to track down this bug…

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.

CS3: Introduction to Symbolic Programming

Lecture 3:
Review of the first two weeks
The “Difference between dates” case study

Fall 2007

Nate Titterton
nate@berkeley.edu

Announcements

- Nate's office hours, <u>for next week</u>:
  - Wednesday, 2-4
  - 329 Soda
- Readers are coming up to speed this week, so look for things to be graded soon…
- Check out the <u>Weiner lecture archives</u>
  - http://wla.berkeley.edu
  - Video lectures and notes from an earlier version of CS3 (still mostly relevant in the earlier weeks)

Schedule

2	Sep 3-7	Lecture: Introduction, Review, Conditionals Reading: Simply Scheme, ch. 3-6 Lab: Conditionals
3	Sep 10-14	Lecture: Conditionals, Case Studies Reading: "Difference between Dates" case study, in the reader (first version) Lab: Explore "Difference between Dates“ Start miniproject 1
4	Sep 17-21	Lecture: Data abstraction in DbD Lab: Finish miniproject 1 Begin recursion
5	Sep 24-28	Lecture: Recursion Lab: More complex recursion
6	Oct 1-4	Lecture: *Midterm 1* Lab: Advanced recursion

Concepts from first two weeks (1/3)

1. **How scheme evaluates input**
2. **Words and sentences**
   - Sentences as a “container”
3. **Conditionals**
   - `cond` and `if` are special forms
   - booleans
     - *truth* (`#t`, or anything) and *non-truth* (`#f`)
   - logical operators
     - `and`, `or`, `not`
   - predicates
     - procedures that return booleans
     - (These end in a ? usually: odd?, vowel?, …)

1 Does the expression contain parentheses? (I.e. is it a "simple" expression without parentheses or a "complicated" expression with parentheses?) Note that a quoted expression such as '(x y) is "complicated", since it really is (quote (x y)). If it's a number, it's self-evaluating; its value is the number itself. If it's a word, it should have been associated with a value, so that value is returned.

2 Otherwise, the expression starts with a left parenthesis. Is "quote" the first word after the left parenthesis? If so, return the quoted word or sentence. Quote is called a special form since it is evaluated in this special-case way.

3 Otherwise, the first word after the left parenthesis should name a procedure; it is looked up among the name of procedures that are either built-in or that have been defined by the user.

4 The arguments are counted to make sure they match the number of placeholder names.

*5 The arguments are evaluated; that is, scheme will work through these 7 steps separately for each of the arguments. (This has the effect of the "inside-out" evaluation we did with expressions involving + and *.)*

6 The argument values are substituted for the corresponding placeholder names throughout the body of the procedure.

7 The body expression is evaluated, and the result is the value of the procedure call.

Concepts from last week (2/3)

1. **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 (3/3)

1. **Helper procedures**
   - Choosing when to write helper procedures is an … art. There is no one right way.
   - 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

```
(load "lib/datesv2.scm")
(load "lib/datesv2.scm")
(load "lib/datesv2.scm")
```

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

Reminder

- This week, (I think) I will leave in many SchemeHandler activities.
  - Many of these you can do in emacs. Some you can't.
  - Remember, try using the unix command `clearcache`, and then restart firefox, if you are having trouble.
  - Let your TAs help you – we are trying to track down this bug…

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.