

## Homework 14: A Guessing Game

Assigned: Friday, 10 November 2006

Due: Tuesday, 14 November 2006

*No extensions!*

**Summary:** In this assignment, you will explore how you might use the techniques of binary search in other contexts.

**Purpose:** To help you think more about the divide-and conquer strategy.

**Expected Time:** One to two hours.

**Collaboration:** You may work in a group of any size between one and four, inclusive. You may consult others outside your group, provided you cite those others. You need only submit one assignment per group.

**Submitting:** Email me your work, using a subject of *CSC151 Homework 14*.

**Warning:** So that this exercise is a learning assignment for everyone, I may spend class time publicly critiquing your work.

### Background: Applying Binary Search in Other Contexts

In a recent laboratory, you explored the *binary search* procedure, which searches for a keyed value in a sorted vector by identifying a section of interest and repeatedly refining that section by checking the middle element and discarding half. It turns out that there are many applications that use the strategy of dividing your values into two halves, figuring out which half to discard, and then recursing on the other half.

For example, we can use this strategy to automate a simple guessing game. Suppose in this game, another player thinks of a name and you try to guess it. What strategies might you use to guess the game?

- You could make a list of all the names you know, and ask about each in turn.
- You could make a list of all the names you know, and randomly choose one at each turn. (It seems that some of Sam Rebelsky's children regularly use this strategy when we play twenty questions.)

However, both of these strategies will require you to ask a lot of questions before getting to the right name. How do we do better? If the only question we can ask is "Is it *name*?", we can't do much better. However, if we can also ask "Does it come before ...?", we can use binary search.

For example, here is a session with a system that asks such questions, as it tries to guess the names of the faculty currently teaching CSC151.

Think of a name and I'll try to guess it.  
Is the name Jett? no  
Does the name alphabetically precede Jett? no  
Is the name Monserrat? no  
Does the name alphabetically precede Monserrat? no  
Is the name Saul? no  
Does the name alphabetically precede Saul? yes  
Is the name Raina? no  
Does the name alphabetically precede Raina? no  
Is the name Roman? no  
Does the name alphabetically precede Roman? no  
Is the name Rylee? no  
Does the name alphabetically precede Rylee? no  
Is the name Sammy? no  
Does the name alphabetically precede Sammy? yes  
Is the name Salma? no  
Does the name alphabetically precede Salma? no  
Is the name Samantha? no  
Does the name alphabetically precede Samantha? yes  
Is the name Salvatore? no  
Does the name alphabetically precede Salvatore? no  
Is the name Sam? yes  
I win!

Play again? yes  
Think of a name and I'll try to guess it.  
Is the name Jett? no  
Does the name alphabetically precede Jett? yes  
Is the name Daniela? no  
Does the name alphabetically precede Daniela? no  
Is the name George? no  
Does the name alphabetically precede George? no  
Is the name Iyana? no  
Does the name alphabetically precede Iyana? no  
Is the name Janiyah? no  
Does the name alphabetically precede Janiyah? yes  
Is the name Jaime? no  
Does the name alphabetically precede Jaime? no  
Is the name James? no  
Does the name alphabetically precede James? no  
Is the name Jana? no  
Does the name alphabetically precede Jana? no  
Is the name Janessa? no  
Does the name alphabetically precede Janessa? no  
Is the name Janiah? no  
Does the name alphabetically precede Janiah? yes  
Is the name Janet? yes  
I win!

Play again? no

Where do these names come from? We've put together a list of 2000 popular names, which you can find at </home/rebelsky/Web/Courses/CS151/2006F/Examples/names.scm>. Note that a properly working binary search procedure needs to divide 2000 in half only twelve or so times before it runs out of names.

## Assignment

Write a procedure, `(guess-name)`, that plays a name guessing game (as above), using the binary search strategy. Your interaction with the user should look something like the one given above.

You can find a variety of hints and helpers in the file `guessing-game.scm`. You can simply add your procedure to that file. We discuss some of the helpers below.

## Helpers

### Names

The file `/home/rebelsky/Web/Courses/CS151/2006F/Examples/names.scm` defines a vector, `names`, that contains 2000 strings, sorted in alphabetical order. We load that file at the beginning. A similar file

`/home/rebelsky/Web/Courses/CS151/2006F/Examples/fewernames.scm`, also defines a vector, `names`, that contains about one dozen strings in alphabetical order. For testing, you may want to use that file instead.

```
> (vector-length names)
2000
> (vector-ref names 100)
"Alondra"
> (vector-ref names 826)
"Heaven"
> (vector-ref names 1452)
"Megan"
```

### Input and Output

Of course, you'll have a bit of difficulty completing the assignment as given without some extra help. In particular, while you've learned to read from files, you haven't learned to read what the user types on the keyboard. Fortunately, it's not that much different. In particular, all of the procedures that read input, when called without an input port, read from the keyboard. Hence, you can read a value with `(read)`, read a character with `(read-char)`, and look at the next character with `(peek-char)`. Putting that all together, we might write the following procedure to read one line from the keyboard.

```
;;; Procedure:
;;; read-line
;;; Parameters:
;;; source, an input port
;;; Purpose:
;;; Read one line of input from a source and return that line
;;; as a string.
;;; Produces:
;;; line, a string
;;; Preconditions:
;;; The source is open for reading. [Unverified]
;;; Postconditions:
;;; Has read characters from the source (thereby affecting
```

```

;;; future calls to read-char and peek-char).
;;; line represents the characters in the file from the
;;; "current" point at the time read-line was called
;;; until the first end-of-line or end-of-file character.
;;; line does not contain a newline.
(define read-line
  (letrec ((read-line-of-chars
            (lambda ()
              (cond
                ((eof-object? (peek-char)) null)
                ((char=? (peek-char) #\newline) (read-char) null)
                (else (cons (read-char) (read-line-of-chars)))))))
    (lambda ()
      (list->string (read-line-of-chars)))))

```

To make your program even easier to write, we've also provided you with a procedure, `(yes-or-no? question)`, that prints the questions, waits for a response, and returns `#t` if the response is yes and no otherwise.

## An Example

In case you want to think about these helpers in context, we've written a few guessing game procedures that use less optimal strategies than the one you are to write. Here's one that guesses names randomly.

```

(define random-guess-name
  (lambda ()
    ; Remember the length of the vector
    (let ((len (vector-length names)))
      ; In the kernel, we just guess a random position
      (letrec ((kernel
                (lambda ()
                  (cond
                    ((yes-or-no? (string-append "Is the name "
                                                  (vector-ref names (random len))
                                                  "?")))
                    (else (kernel))))))
        (print-line "I win!")
        (else (kernel))))))
    (print-line "Think of a name and I'll try to guess it.")
    (kernel)
    (newline)
    (if (yes-or-no? "Play again?")
        (random-guess-name)
        (print-line "Thanks for playing."))))

```

Note that this procedure, like an overly energetic child, never gives up.

---

Copyright © 2006 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.