

Problem Set 2
Recursion and Iteration

Issued: Section 201 (Dimock): Friday, 17-Sep-2004, Section 202 (Yanco): Thursday, 16-Sep-2004
Due: Section 201 (Dimock): Friday, 24-Sep-2004, Section 202 (Yanco): Thursday, 23-Sep-2004
Reading: Text (SICP 2nd Edition by Abelson & Sussman): Through Chapter 1

Overview

This problem set asks you to write some Scheme procedures using the techniques from Chapter 1, and to think about the performance characteristics of the processes generated by some of the procedures. Use the MzScheme language in DrScheme.

What to turn in

Keep your answers in a file named `ps2-ans.ss` (the file that saves your definitions buffer). Put the code for each problem sequentially in the definitions buffer. (If you work problems in a different sequence, put them in order before you submit.) Put the problem number in a comment before the code for that problem (use a semi-colon to make a line a comment).

Underneath the code for each problem, cut and paste the appropriate sample runs. Please put semi-colons in front of the sample runs, which will comment them out and allow you to load the entire buffer in another session.

Print the definitions buffer (or the file `ps2-ans.ss`, which should be equivalent) and turn it in during the class on the due date. Also submit the answers using the submit command:

Section 201: `submit dimock ps2 ps2-ans.ss`

Section 202: `submit holly 301-ps2 ps2-ans.ss`

Problems

Problem 1: The following code is a very slow way of determining whether a number is even or odd.

```
(define is-even
  (lambda (x)
    (if (= x 0)
        #t
        (is-odd (- x 1))))

(define is-odd
  (lambda (x)
    (if (= x 0)
        #f
        (not (is-even x)))))
```

Another version of these routines is:

```
(define is-even
  (lambda (x)
    (if (= x 0)
        #t
        (is-odd (- x 1))))

(define is-odd
  (lambda (x)
    (if (= x 0)
        #f
        (is-even (- x 1)))))
```

Use the substitution model¹ to evaluate `(is-even 4)` in both versions of the code. Are these processes generated from these procedures recursive or iterative?

Problem 2: Write a recursive procedure `binary` that takes a decimal number and returns what appears to be its binary representation. You will not actually be converting from decimal to binary, but instead converting a decimal number to another number that has a decimal representation that consists only of 1s and 0s. `(binary 13)` should return 1101 since 13 base 10 = 1101 base 2.

You may want to use the standard Scheme procedure `even?` or `odd?`.²

Your code only has to work for numbers ≥ 0 .

Suggestion: you might want to try tracing your code.

```
(require (lib "trace.ss"))
(trace binary)
(binary 13)
(untrace binary)
```

¹Feel free to be informal in your use of the substitution model: Just write a line for each procedure application in the process, a line for the execution of each postponed operation, and a line for the final result.

²If we were using a base other than 2, you would probably want to use "remainder" or "modulo".

Problem 3: Write a procedure generating an iterative process `binaryi` that, given the same number for input as the `binary` procedure above, returns the same output.

You may want to use `trace` to debug this version. Since you are dealing with an internal `define` for the iterative procedure, you would need to put `trace` into your code in the scope of the internal `define`. If you do so, remember to take it out before submitting the completed code as part of `ps2-ans.ss`.

Problem 4: Take the following procedure that generates a tree-recursive process and convert it to an procedure that generates an iterative process using the same method that was used in the book for iterative `fib`.

```
(define (f n)
  (cond ((< n 1) 0)
        ((<= n 2) 1)
        (else (+ (* 2 (f (- n 2)))
                  (f (- n 3))))))
```

You should write a function `g` based on the following outline:

```
(define (g n)
  (define (f-iter a b c count)
    (cond ((<= count 0) <fill in here>)
          (else <fill in here>)))
  (f-iter <fill in here>))
```

Problem 5: In the first problem set, we used `factorial` to create a function that calculated the binomial coefficient $\binom{n}{m}$. Here is a different algorithm. The binomial coefficient counts the number of ways to choose m distinct objects from a collection of n objects.

How many ways can you choose m objects from a collection containing n objects?

If $m > n$ or $n < 0$ then this is an impossible task: there are 0 ways to choose.

If $m = n$ there is one way: choose them all.

If $m = 0$ there is one way: don't choose any of them.

How about if $0 < m < n$?

This divides into two cases: If you choose the first of the n objects then you have to choose $m-1$ objects from the remaining $n-1$ objects. If you don't choose the first object then you have to choose all m objects from the $n-1$ remaining objects. So $\binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}$

Write a procedure `choose` that implements the recurrence relation and boundary conditions given above. Your procedure should take two integers, n and m . It should be able to handle negative integers.

Submit the code for your procedure `choose` as part of `ps2-ans.ss`.

How does the time to execute the process, $T(n)$ grow as n changes if m is set close to $n/2$?

Get some feel for the order of growth:

```
(time (choose 18 9))
(time (choose 20 10))
(time (choose 22 11))
(time (choose 24 12))
```

How does the time to execute the process grow as n changes if $m = 1$?

Problem 6: Exercise 1.20, page 49

Problem 7: Write a procedure `max-of-two-fns` that takes two numerical procedures `f` and `g`, and a number `x` as inputs, and returns the maximum of applying `f` to `x` and `g` to `x`.

```
> (max-of-two-fns sin cos 1.5706)
0.9999999807278949
```

```
> (max-of-two-fns log tan 1.5706)
5093.548171447941
```

Now define a procedure to sum the values of two functions at a number `x`.

```
> (sum-of-two-fns sin cos 1.5706)
1.0001963075215303
```

Now define a procedure to combine using a function `c` that takes two numbers.

```
> (combine-of-two-fns max sin cos 1.5706)
0.9999999807278949
```

```
> (combine-of-two-fns < sin cos 1.5706)
#f
```

Problem 8: Help the course staff with their grading:

Write a function `m-to-n` that takes three parameters: an integer `lo`, an integer `hi` that is $\geq lo$ and a function `f`.

Your function should call the function `f` on each value from `lo` to `hi` (inclusive of both endpoints), and then return `#f`.

You might want to try your function as

```
(m-to-n
 5 10
 (lambda (x) (begin (display (binary x))
                    (newline))))
```

Problem 9: Help the course staff even more with their grading:

Write a function `m-to-n2` that takes two parameters: an integer `lo`, an integer `hi` that is $\geq lo$. `m-to-n2` returns a *function* that takes two functions as parameters and compares their values (using `=`) on values `lo` to `hi` until either `hi+1` is reached, in which case return `#t` or the values of the two passed functions differ on some number, in which case return the number that they differ on.