Structure and Interpretation of Computer Programs
October 2000

Problem Set 2
Higher Order Procedures

Issued: Tuesday, 3 October 2000
Due: Thursday, 5 October 2000
Reading: Text (SICP 2nd Edition by Abelson & Sussman): Sections 1.2 and 1.3

Overview

This problem set deals with higher order procedures and introduces ideas about generic operators that we will see later in the term when we discuss abstract data types. We will also look into recursion and iteration as means to describe processes and write code to create them. Be careful when you verify your procedures as at times the suggested verifications can hide subtle bugs that will crop up later.

1. Warming Up

Exercise 1 Do exercise 1.30 on p. 60 of the text.

Exercise 2 Do exercise 1.31 on p. 60–61 of the text.

Exercise 3 Do exercise 1.32 on p. 61 of the text.

Exercise 4 Do exercise 1.33 on p. 61 of the text.

Exercise 5 Procedure `repeated` is defined as

```
(define (repeated p n)
  (cond ((= n 0) (lambda (x) x))
        ((= n 1) p)
        (else (lambda (x) (p ((repeated p (- n 1)) x))))))
```

Determine the values of the expressions

```
((repeated square 2) 5)
```

and

```
((repeated square 5) 2)
```

on paper before checking your answers on the computer.

2. Continued Fractions

Continued fractions play an important role in number theory and in approximation theory. In this programming assignment, you will write some short procedures that evaluate continued fractions. There is no predefined code for you to load.

An infinite continued fraction is an expression of the form

$$f = \cfrac{N_1}{D_1 + \cfrac{N_2}{D_2 + \ddots}}$$

One way to approximate an irrational number is to expand as a continued fraction, and truncate the expansion after a sufficient number of terms. Such a truncation—a so-called *k-term finite continued fraction*—has the form

$$\cfrac{N_1}{D_1 + \cfrac{N_2}{\ddots + \cfrac{N_K}{D_K + 0}}}$$

For example, if the N_i and the D_i are all 1, it is not hard to show that the infinite continued fraction expansion

$$\cfrac{1}{1 + \cfrac{1}{1 + \ddots}}$$

converges to $1/\phi \approx .618$ where ϕ is the *golden ratio*

$$\frac{1+\sqrt{5}}{2}$$

The first few finite continued fraction approximations (also called *convergents*) are:

$$1, \quad \frac{1}{2} = 0.5, \quad \frac{2}{3} \approx 0.667, \quad \frac{3}{5} = 0.6, \quad \frac{5}{8} = 0.625, \quad \ldots$$

Exercise 6 Suppose that `n` and `d` are procedures of one argument (the term index) that return the n_i and d_i of the terms of the continued fraction.

Define procedures `cont-frac-r` and `cont-frac-i` such that evaluating `(cont-frac-r n d k)` and `(cont-frac-i n d k)` each compute the value of the k-term finite continued fraction. The process that results from applying `cont-frac-r` should be recursive, and the process that results from applying `cont-frac-i` should be iterative. Check your procedures by approximating $1/\phi$ using

```
(cont-frac (lambda (i) 1)
           (lambda (i) 1)
           k)
```

for successive values of `k`, where `cont-frac` is `cont-frac-r` or `cont-frac-i`. How large must you make `k` in order to get an approximation that is accurate to 4 decimal places? Turn in a listing of your procedures.

Exercise 7 One of the first formulas for π was given around 1658 by the English mathematician Lord Brouncker:

$$4/\pi - 1 = 1/(2 + 9/(2 + 25/(2 + 49/(2 + 81/(2 + \cdots$$

This leads to the following procedure for approximating π:

```
(define (estimate-pi k)
  (/ 4 (+ (brouncker k) 1)))

(define (square x) (* x x))

(define (brouncker k)
  (cont-frac (lambda (i) (square (- (* 2 i) 1)))
             (lambda (i) 2)
             k))
```

where `cont-frac` is one of your procedures `cont-frac-r` or `cont-frac-i`. Use this method to generate approximations to π. About how large must you take k in order to get 3 decimal places accuracy?

What happens when you evaluate `(estimate-pi 20000)` when the definition of `brouncker` using `cont-frac-r`? Why? What happens when you evaluate the same expression using `cont-frac-i` in `brouncker`? Why?

Computing inverse tangents

Continued fractions are frequently encountered when approximating functions. In this case, the N_i and D_i can themselves be constants or functions of one or more variables. For example, a continued fraction representation of the inverse tangent function was published in 1770 by the German mathematician J. H. Lambert:

$$\arctan x = \cfrac{x}{1 + \cfrac{(1x)^2}{3 + \cfrac{(2x)^2}{5 + \cfrac{(3x)^2}{7 + \ddots}}}}$$

Exercise 8 Define a procedure `(atan-cf k x)` that computes an approximation to the inverse tangent function based on a k-term continued fraction representation of $\arctan x$ given above. Provided you use the correct arguments, it should be possible to define this procedure directly in terms of the `cont-frac` procedure you have already defined.

Exercise 9 Verify that your procedure works (and check its accuracy) by using it to compute the arctangent of a number of arguments, such as 0, 1, 3, 10, 30, 100. Compare your results with those computed using Scheme's `atan` procedure[1].

How many terms are required to get reasonable accuracy? Turn in your procedure definition together with some sample results.

[1] Evaluating `(atan x)` returns the same value as `(atan x 1)`, an angle in the range $(-\pi/2, +\pi/2)$. See page 23 of the *Revised*[4] *Report on the Algorithmic Language Scheme* if you're interested in reading more.

Exercise 10 The idea of a continued fraction can be generalized to include arbitrary nested expressions such as (1) and (2) below:

$$(N_1/\,(D_1 + N_2/\,(D_2 + \cdots + N_k/\,(D_k + 0)))) \tag{1}$$

$$(N_1 + D_1 \times (N_2 + D_2 \times \cdots \times (N_k + D_k \times 1))) \tag{2}$$

or in general

$$(T_1\mathcal{O}_1\,(T_2\mathcal{O}_2 \cdots (T_k\mathcal{O}_k R)))$$

where the T_i are the terms of the expression, the $\mathcal{O}_i$ are the arithmetic operators, and R is the residual term (perhaps representing the effects of terms beyond the kth in an approximation). For example, in (2) the $\mathcal{O}_i$ are alternately the addition and the multiplication operators, and $R = 1$.

Define a procedure `nested-acc` such that evaluating `(nested-acc op r term k)` computes the value of a nested expression. Argument `op` is a procedure of one argument, that when applied to the term index i returns the ith arithmetic operation $\mathcal{O}_i$ (which is a procedure of two arguments). The `r` argument represents the effect of the R term. Turn in a listing of your procedure.

How would you use your `nested-acc` procedure to compute a k-term approximation to the function

$$f(x) = \sqrt{x + \sqrt{x + \sqrt{x + \cdots}}}$$

Exercise 11 Verify that your `nested-acc` procedure works properly by computing an approximation to $f(1)$, whose value is the golden ratio. How large must you make `k` in order to get an approximation that is accurate to 4 decimal places? Turn in a listing of your procedures.

Exercise 12 In certain continued fractions all N_i and D_i terms are equal and it is convenient to regard the fraction as being built-up from a base through a repetitive operation. For example, given the base function B and the augmentors N and D, one can build $N/(D + B)$. This can be computed in a straightforward fashion by the procedure `build`

```
(define (build n d b)
  (/ n (+ d b)))
```

The value of the expression that is represented by the two-term continued fraction

$$\frac{N}{D + \frac{N}{D+B}}$$

can then be computed by evaluating `(build n d (build n d b))`. When further composition is of interest, it is possible to use the `repeated` procedure defined above.

Exercise 12 Write a procedure `repeated-build` of four arguments `k`, `n`, `d`, and `b`, which returns a result that is equivalent to applying the `build` procedure k times. In particular `(repeated-build 2 n d b)` should return the same result as `(build n d (build n d b))`. Your implementation of `repeated-build` should make use of the `repeated` procedure given above (in a non-trivial way).

Exercise 13 Verify that your `repeated-build` procedure works properly by evaluating the continued fraction for the reciprocal of the golden ratio:

$$\cfrac{1}{1+\cfrac{1}{1+\ddots}}$$

converges to $1/\phi \approx 0.618$ where ϕ is the golden ratio.

Now consider the problem of computing the rational functions $r_1(x) = (1+0x)/(1+x)$, $r_2(x) = (1+x)/(2+x)$, $r_3(x) = (2+x)/(3+2x)$,

$$r_k(x) = \frac{a_k + a_{k-1}x}{a_{k+1} + a_k x} = \frac{1}{1 + r_{k-1}(x)}$$

where a_k is the k-th Fibonacci number ($a_0 = 0$).

Exercise 14 Making use of your `repeated-build` procedure, define a procedure `r` of one argument `k` that evaluates to a procedure of one argument `x` that computes $r_k(x)$. For example, evaluating `((r 2) 0)` should return 0.5.

Optional Problem

One difficulty with using continued fractions to evaluate functions is that there is generally no way to determine in advance how many terms of an infinite continued fraction are required to achieve a specified degree of accuracy. The result obtained by computing the value of a continued fraction by working backwards from the k-th to the first term cannot generally be used in the computation of the $k+1$ term continued fraction.

Remarkably, there is a general technique for computing the value of a continued fraction that overcomes this problem. This result is based on the recurrence formulas

$$A_{-1} = 1, \quad B_{-1} = 0, \quad A_0 = 0, \quad B_0 = 1 \tag{3}$$

$$A_j = D_j A_{j-1} + N_j A_{j-2} \tag{4}$$

$$B_j = D_j B_{j-1} + N_j B_{j-2}. \tag{5}$$

It is easy to show that the k-term continued fraction can be computed as

$$f_k = \frac{A_k}{B_k}. \tag{6}$$

By determining whether f_j is close enough to f_{j-1}, it is easy to determine whether to include more terms in the approximation.

Define a procedure that computes the value of a continued fraction using this algorithm. The process that evolves when the procedure is applied should be iterative. Test that your procedure works by computing the tangent of various angles. Use the tracing feature of Scheme to determine how many terms are evaluated.

Problem Set Evaluation To help us evaluate how well the course is going, and how good a job we are doing, please tear off this page, answer the following questions, and attach it to your problem set when you hand it in. While we do not require you to fill in this survey, doing so will help correct any problems and improve the course not only for future students, but for you as well.

Number of hours spent on the reading assignment: _____

Number of total hours spent working directly on the problem set _____

Number of collaborative hours spent working directly on the problem set _____

How hard was this problem set (1–10: 1, piece of cake; 5, just right; 10, far too hard) _____

Hardest problem _____

Fraction of time spent on hardest problem, multiplied by 10 (1–10: 1, only a small fraction; 5, half of the time; 10, every single minute) _____

Coherence between problem set contents and lecture/recitation contents (1–10: 1, seemingly random; 5, good correspondence; 10, perfect coherence) _____

Coherence between problem set contents and assigned reading contents (using the same scale) _____

How much did you enjoy this problem set (1–10: 1, nearly walked out of the program; 5, neutral; 10, recruited other students because it was so good) _____

How much did you enjoy the lectures and recitations for this problem set (using the same scale) _____

You may also submit fully anonymous comments via the electronic means described in class.