MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall Semester, 1997

Problem Set 2

Issued: September 16, 1997

Due: In Recitation, Wednesday September 24

Tutorial exercises due: in tutorial, the week of September 22

Reading: Complete Chapter 1 of *SICP*.

1. Tutorial Exercises

Exercise 2.1 Do exercise 1.16 of the text.

Exercise 2.2 Do exercise 1.17 of the text.

Exercise 2.3 Do exercise 1.26 of the text.

Exercise 2.4 Procedure `repeated` takes as arguments a procedure `p` and a number `n`, where `p` itself takes one argument, and returns a new procedure of one argument by "composing `p` with itself `n` times." Namely,

```
(define (repeated p n)
  (if (zero? n)
      (lambda (x) x)
      (lambda (x) (p ((repeated p (dec n)) x)))))
```

Determine the values of the expressions

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

and

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

In tutorial you may be asked how the interpreter evaluates expressions similar to these.

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:

$$1, \frac{1}{2} = .5, \frac{2}{3} \approx .667, \frac{3}{5} = .6, \frac{5}{8} = .625, \cdots$$

Pre-Lab Exercise 2.1 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.

Lab Exercise 2.2 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 2 decimal places accuracy? (Don't try for more places unless you are very patient.)

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:

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

Pre-Lab Exercise 2.3 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 $\tan^{-1} x$ 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.

Lab Exercise 2.4 Verify that your procedure works (and check its accuracy) by using it to compute the tangent 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].

[1]Evaluating `(atan x)` returns the same value as `(atan x 1)`, an angle in the range $(-\pi/2, +\pi/2)$.

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

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 $R = 1$.

Pre-Lab Exercise 2.5 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

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

Lab Exercise 2.6 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.

Lab Exercise 2.7 The Taylor series for computing the sine function is

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots \tag{3}$$

Define a simple recursive procedure `sine-taylor` such that `(sine-taylor x n)` is the sum of the first n terms of the series.

Now verify that your `nested-acc` procedure works properly by using it to compute the values of this Taylor series and comparing its results to those of `sine-taylor`.

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 by 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.

Pre-Lab Exercise 2.8 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).

Lab Exercise 2.9 Verify that your `repeated-build` procedure works properly by evaluating the continued fraction for the reciprocal, $1/\phi$ of the golden ratio:

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

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 ($a_0 = 0$).

Lab Exercise 2.10 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

$$A_{-1} = 1; B_{-1} = 0; A_0 = 0; B_0 = 1 \tag{4}$$

$$A_j = D_j A_{j-1} + N_j A_{j-2} \tag{5}$$

$$B_j = D_j B_{j-1} + N_j B_{j-2}. \tag{6}$$

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

$$f_k = \frac{A_k}{B_k}. \tag{7}$$

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 your SCHEME system to determine how many terms are evaluated.

See Next Page

Turn in this sheet along with your answers to the questions in the problem set:

How much time did you spend on this homework assignment?

We encourage you to work with others on problem sets as long as you acknowledge it (see the 6.001 General Information handout).

If you cooperated with other students, LA's, or others, or found portions of your answers for this problem set in references other than the course text (such as some of the course archives), please indicate your consultants' names and your references in the space below. Also, explicitly label all text and code you are submitting which is the same as that being submitted by one of your coolaborators.

Otherwise, write "I worked alone using only the course reference materials." and sign your statement.