Nelson Batista

CSC 33500 Programming Language Paradigms — Section R

9/22/16

Homework 3

1.29

A solution is given below:

```
(define (simpson f a b n)
  (let ((h (/ (- b a) n)))
    (* (/ h 3) (simpson-iter f a b n 0 0))))


(define (simpson-iter f a b n count result)
  (let ((h (/ (- b a) n)))
    (cond ((> count n) result)
          ((or (= count 0) (= count n))
           (simpson-iter f a b n (+ count 1)
                         (+ result (f (+ a (* count h))))))
          ((= (modulo count 2) 0)
           (simpson-iter f a b n (+ count 1)
                         (+ result (* 2 (f (+ a (* count h)))))))
          (else
           (simpson-iter f a b n (+ count 1)
                         (+ result (* 4 (f (+ a (* count h))))))))))
```

Running `simpson` with `cube`, `0`, and `1` as the arguments for `f`, `a`, and `b`, respectively, returns 0.25 for both `n = 100` and `n = 1000`, so we know the function definitely approximates the correct value for the definite integral of the function $f(n) = n^3$ between 0 and 1.

1.30

The substitutions have been made in the following code:

```
(define (sum term a next b)
```

```
(define (iter a result)
  (if (> a b)
      result
      (iter (next a) (+ result (term a)))))
(iter a 0))
```

1.31

a

The `product` function can be implemented like so:

```
(define (product term a next b)
  (if (> a b)
      1
      (* (term a)
         (product term (next a) next b))))
```

We can use this function to calculate factorials almost trivially, since a factorial of n is just the multiplication of all the numbers from 1 to n. Here is the code:

```
(define (factorial n)
  (define (nextx x) (+ x 1))
  (define (termx x) x)
  (product termx 1 nextx n))
```