Comp 210
Barland

Abstracting over Functions

Due: Oct.28

98.Fall
Homework #7

As always, your functions should include contracts, test cases, etc.

1. Given a non-negative function f defined on the closed interval $[a, b]$ of real numbers, the *integral* of f from a to b

$$\int_a^b f(x)\,dx$$

is defined as the area enclosed between the curves $f(x)$, the x-axis, and the vertical lines passing through a and b.

Given the function f, one way to compute an approximate value for

$$\int_a^b f(x)\,dx$$

is to:

- Subdivide $[a, b]$ into many small sub-intervals

  $$[x_i, x_{i+1}],\ i = 0, 1, \ldots, n-1$$

  where $x_0 = a$, $x_n = b$; and

- Approximate the area of each subinterval $[x_i, x_{i+1}]$ by the area of the trapezoid with corners (vertices) $(x_i, 0)$, $(x_{i+1}, 0)$, $(x_{i+1}, f(x_{i+1}))$, $(x_i, f(x_i))$.

Then the approximate value for all of

$$\int_a^b f(x)\,dx$$

is the sum of the areas of the trapezoids for the intervals

$$[x_0, x_1], \ldots, [x_{n-1}, x_n]\ .$$

To compute the area A of the trapezoid with corners $(x_i, 0)$, $(x_{i+1}, 0)$, $(x_{i+1}, f(x_{i+1}))$, $(x_i, f(x_i))$, multiply the base times the average height:

$$A = (x_{i+1} - x_i) * \frac{f(x_{i+1}) + f(x_i)}{2}$$

.

The choice of sub-intervals has a significant effect on the accuracy of the approximation. The simplest choice would be to subdivide $[a, b]$ into n equal sub-intervals. Unfortunately, this approach requires a large number of sub-intervals for parts of the function that have sharp peaks and valleys, which is wasteful for the parts of the function that are smooth. Plus, it's not clear how large large should be, for different functions.

For this homework, we will use a better method: subdivide $[a, b]$ into intervals in an *adaptive* manner until the difference between trapezoid area for the interval $[x_i, x_{i+1}]$ and the sum of the trapezoid areas for $[x_i, x_{mid}]$ and $[x_{mid}, x_{i+1}]$—where x_{mid} is the midpoint of $[x_i, x_{i+1}]$—is less than $(x_{i+1} - x_i) * \epsilon$. The constant ϵ is called the "unit error bound"; you can see that the overall error of the final answer will hopefully[1] not be larger than $\epsilon(b - a)$.

Your assignment is to to write a Scheme function *integrate* that takes four parameters:

- *f*: the mathematical function to be integrated;
- *a*: the left endpoint of the integration interval $[a, b]$,
- *b*: the right endpoint of the integration interval $[a, b]$,
- *epsilon*: the unit error bound for the computed result.

The result of *(integrate f a b epsilon)* is the area under the integral according to the adaptive quadrature method described above. The approximate error in the answer is $\epsilon(b - a)$.

[1]Note that the error bound is not guaranteed, since we are comparing the differences between successive approximations rather the differences between approximations and exact answers. Can you concoct a function where the actual error is greater than the error bound?

2. Returning functions. The symbols α, β, γ are type-variables: they stand for any desired type of value (number, symbol, boolean, etc.). As always, show tests which make you confident of correctness.

- Write the function *compose*, which takes two functions $f : \beta \to \gamma$ and $g : \alpha \to \beta$, and returns a function $h : \alpha \to \gamma$, where $h(x) = f(g(x))$. For example,

```
  (define sqrt-plus-one (compose add1 sqrt))
  (sqrt-plus-one 4)
= 3
```

- Write the function *iterate*, which takes a function $f : \alpha \to \alpha$ and a natural number n, and returns a function which applies f to its input n times repeatedly. (This is sometimes denoted $f^{(n)}$.) (To consider: What is a function which takes an input and applies f zero times?)

3. Data hiding is an important aspect of structured programming. For this problem, your task is by write two functions, *create-lock* and *check-lock*. The function *create-lock* takes as input a symbol *passwd* and returns a "lock", whatever that is. The function *check-lock* takes a symbol *key* and a lock, and returns `#t` if and only if *key* agrees with the password used to create the lock. Example:

```
(define my-passwd (create-lock 'secret-handshake))

(check-lock my-passwd 'is-this-it?)       = #f
(check-lock my-passwd 'secret-handshake)  = #t

my-passwd
;; No interesting information should be printed in drscheme!
```

Note that using a symbol or structure for the lock would not be secure at all: evaluating the placeholder `my-passwd` above would reveal the symbol. Your representation should be secure in the sense that if you didn't see my call to `create-lock`, there is nothing you could type at my drscheme window to get `check-lock` to answer `#t`, short of making an incredibly lucky guess. So what type of Scheme value might a lock be? Well, this ¡em¿is¡/em¿ the

homework about functions which accept, create, and return functions.... ¡p¿ In writing `create-lock`, you may use *local* if you wish, though it is not necessary nor advised. ¡p¿