Comp 210
Barland

Abstracting over Functions

Due: Mar.18

98.Spring
Homework #7

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 the area enclosed between the curve formed by $f(x)$, the x-axis, and vertical lines passing through a and b. The area is often called the "area under the curve f between a and b."

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

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

is by:

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

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

  where $x_0 = a$, $x_n = b$; and
- approximating the area of each subinterval $[x_i, x_{i+1}]$ by the area of the trapezoid with corners (vertices) $(x_i, 0)$, $(x_{i+1}, 0)$, $(x_{i+1}, f(x_{i+1}))$, $(x_i, f(x_i))$.

The approximate value for

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

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 $(x_i, 0)$, $(x_{i+1}, 0)$, $(x_{i+1}, f(x_{i+1}))$, $(x_i, f(x_i))$, we use the formula

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

which is the base times the average height.

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.

For this problem, 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.

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 object-oriented programming. For this problem, your task is to write two functions, *set-passwd* and *check-passwd*. The function *set-passwd* takes as input a symbol *passwd* and returns a password *object*. The function *check-passwd* takes a symbol *passwd* and a password object and returns true if and only if *passwd* agrees with the password encapsulated in the object.

Hint: The password object should be a function created by *set-passwd*. You may use *local* if you wish (although it is not necessary). Your representation should be secure in the sense that there is no way that Scheme code can extract a password from a password object (beyond exhaustively testing all possible passwords).