## 1 Accumulator-Style Program Design

## Portfolio Problems

For each of the following problems work out the solution in four different ways:

- using the design recipe
- modifying the previous solution by using an acumulator
- implementing the solution using the Scheme loop *foldl*
- implementing the solution using the Scheme loop *foldr*

**Problems:**

Continue with the problem covered in the first lecture and in the lab. The producer now requires that the profits from the commercials depend on when the show broadcasts as well as show's rating.

1. Modify the data definitions so that it includes the information about the show's rating (on a scale from 1 to 10) and the show time (recorded only as in one of four time categories).
2. Modify the function that computes the show's profit as follows. The base profit is multiplied by the show time category (from 1 to 4) and by the show'd rating factor (one tenth of the show's rating, i.e. 0.1, 0.2, up to 1.0).
3. Define the function that computes the total profit from the given ad during the show. Make four versions of this function: a basic one that follows the *Design Recipe*, one that uses the accumulator, and two versions that use *foldl* and *foldr*.
4. Problem 31.3.4 in HtDP

## Pair Programming Assignment

### 1.1 Problem

A. Design the data to represent a list of nodes in a graph and a list of edges connecting the nodes.

For example, here is a graph and the information to represent:

```
B<----->A<-----------C
|       ^           /
v       |          /
D------>E         /
^       |        /
|       |       /
|       |      /
F       |     /
 \      |    /
   \    |   /
     \  |  /
        v
        G
```

Nodes: *A*, *B*, *C*, *D*, *E*, *F*, *G*

Edges: (*A*, *B*) (*B*, *A*) (*B*, *D*) (*C*, *A*) (*C*, *G*) (*D*, *E*) (*E*, *A*) (*E*, *G*) (*F*, *D*) (*F*, *G*) (*E*, *G*)

There is no ordering of the list of nodes or the list of edges. Each edge has the source and the target, and so the edge from *A* to *B* is different from the edge from *B* to *A*, though it is possible that both exist, as shown.

Note that each node has a name (typically just one letter) and we may choose to record its location, so we can later draw the graph on a *Canvas*.

B. Design the function *neighbors* that for a given node and a list of edges produces a list of all neighbors of the given node. For example, neighbors of the node C above are nodes *A* and *G*.

   Follow the design recipe precisely. It is sufficient to produce the names of the nodes, not the entire information about each neighbor.

C. Define the function *neighbors2* that solves the same problem, but uses the accumulator.

D. Define the function *neighbors3* that solves the same problem, but uses the *foldl* loop.

E. Define the function *neighbors4* that solves the same problem, but uses the *foldr* loop.

**Note:** Use the *check-expect* format of the *testing.ss* teachpack to define all your test cases.

## 1.2 Problem

We now want to draw the graph on the *Canvas*.

A. Design the function *draw-node* that draws the node as a small red circle with its letter label in the middle.

B. Design the function *draw-edge* that draws the given edge as a red line between the locations of its two endpoints. Think carefully about what information will this function need to accomplish its task.

C. Create a simple line drawing, represent it as a list of edges, and a list of nodes. Now design a function *draw-edges* that draws all edges in a given graph.

D. Design the function *draw-graph* that will draw the whole graph on the *Canvas*.

## 1.3 Problem

Your boss tells you that she would like to record for each node of the graph a list of the names of its neighbors.

A. Design the data definition for *Node2* that includes the information abotu its neighbors. Translate the original data definitions to this one.

B. Design the function *find-edges* that extracts from a list of *Node2* data the list of all edges in that graph.

C. Design the function *draw-graph2* that will draw the whole graph (a list of *Node2* data) on the *Canvas*.