Mission Sidequest 20.1: Stream Manipulation

Start date: 31 October 2017
Due: 12 November 2017, 23:59

Readings:

- SICP, Section 3.5

This mission consists of **three** tasks.

Task 1:

Scottie is addicted to functional programming - the simplicity, the elegance, oh my! If Scottie's friends wanted Scottie to produce a list of even numbers from 0 to 10, Scottie would simply:

```
filter(is_even, enum_list(0, 10));
```

Unfortunately, things become much less efficient when Scottie is asked for a subset of the computed result. For example, if Scottie just wanted the first 10 even numbers from 0 to 10000, Scottie might consider:

```
take(filter(is_even, enum_list(0, 10000)));
```

The inefficiency stems from the fact that `enum_list` will build the entire list from 0 to 10000. Once the list is built, `filter` will go through the entire list to remove any odd numbers. To combat the inefficiency, Scottie wrote the following so that the list is built from the ground up, and terminates early once it has accomplished its goal:

```
function get_results(pred, n, i, ys) {
    if (length(ys) >= n) {
        return ys;
    } else if (!pred(i)) {
        return get_results(pred, n, i + 1, ys);
    } else {
        return get_results(pred, n, i + 1, pair(i, ys));
    }
}
var results = get_results(is_even, 10, 1, []);

// Put into ascending order
results = reverse(results);
```

However, Scottie is still not pleased with the above implementation. Even though Scottie only wants the first 10 even numbers now, it does not mean that Scottie doesn't want the rest of the even numbers later - in other words, Scottie wants to just compute the first 10

even numbers now and "pause" the process of finding the rest of the even numbers for later, somehow.

Fortunately, you think you might have an idea to Scottie's woes - **streams**.

For this task, implement `stream_take` and `stream_drop`, such that `stream_take` returns a stream that contains the first `n` elements of the given stream, and `stream_drop` returns a stream that contains elements without the first `n` elements of the given stream. Important: Both functions should return a **stream** - recall stream discipline!

Task 2:

Scottie is very impressed with the capabilities of streams, but Scottie is not done yet! Scottie challenges you to do the following.

1) Start with an infinite sequence of integers

```
// Some sequence representing 1, 2, 3, ...
var integers = integers_from(1);
```

2) Transform the infinite sequence of integers into an infinite sequence of squared even integers

```
// Some sequence representing 2, 4, 6, ...
var even_integers = ...;

// Some sequence representing 4, 16, 36, ...
var squared_even_integers = ...;
```

3) Apply the add function to the sequence of integers from left-to-right, such that each element is the sum of all the elements before it plus itself

```
// Some sequence representing 4 + 0, 16 + 4 + 0, 36 + 16 + 4 + 0, ...
// Note: The first element of this sequence is just initial
var sum_square = stream_sequence(add, 0, squared_even_integers);
```

For this task, implement `even_integers`, `squared_even_integers`, and `stream_sequence`. `stream_sequence` should be implemented such that it returns a *stream* that lazily consumes the next element to produce the accumulated value. The returned stream should return the empty stream if it cannot find any next element to consume.

Task 3:

The help that you have offered Scottie has not gone unnoticed! One of your peers in the Source Academy recently created an entertainment bot called @realScottieBot. The bot is implemented as a long-running server that listens for the `/cookie` action, and returns either the string `"Hbebuerq"` or the string `"Kgasnsa"` by looking up a certain value.

The value is currently hard-coded, and it gets quite tiresome to have to change it every few hours. Hence, it is an excellent candidate for automation! Since the expiration date of @realScottieBot is unknown, it would be best to assume that @realScottieBot needs to know which cookie to hand out for an indefinite amount of time!

Your peer proposes the following implementation:
1) `simple_cookie_stream` is initialized, such that it has the elements - pair(duration_0, cookie_0), pair(duration_1, cookie_1), ...
2) `simple_cookie_stream` is initialized, such that cookie starts with "Hbebuerq" and alternates between "Hbebuerq" and "Kgasnsa"
3) The duration follows a step sequence - 1, 2, 3, 1, 2, 3, 1 ...

Hence, or otherwise, the above implementation should result in a stream with the following sequence:

```
pair(1, "Hbebuerq"),
pair(2, "Kgasnsa"),
pair(3, "Hbebuerq"),
pair(1, "Kgasnsa"),
pair(2, "Hbebuerq"),
pair(3, "Kgasnsa"), ...
```

For this task, implement `step_duration_stream`, `oscillating_cookie_stream`, and `stream_zip`.