Streams II

delayed / lazy list

null or pair

one evaluated item → head.

when to wrap a nullary function?

eval_stream → have head function !!!

at least 1 elem

stream_filter : works as filter.

integers_from(1)

add_streams(integers, one_stream()).

integers → implemented using one_stream.

```
pair(1,
    () => add_streams(
        integers,
        one_stream()
    )
);
```

=>

```
pair(2,
    () => add_streams(
        stream_tail(integers),
        one_stream().
    )
);
```

How to solve : inductive reasoning.

pattern recognition

memo_fun => store results.

=> not be destroyed.

=> Space complexity. O(1)

destroy?, Not really in source.

Stream map.

Stream filter

etc.

pi_summands(n)

=> series sum.

```
pair(1/n, () => stream_map(x => -x,
                           pi_summands(n + 2)));
```

National University of Singapore
School of Computing
CS1101S: Programming Methodology
Semester I, 2021/2022

S11
Streams

Problems:

Getting started

1. Describe the streams A and B produced by the following definitions. Assume that `integers` is the stream of positive integers (starting from 1):

```
function scale_stream(c, stream) {
    return stream_map(x => c * x, stream);
}

const A = pair(1, () => scale_stream(2, A));

function mul_streams(a,b) {
    return pair(head(a) * head(b),
                () => mul_streams(stream_tail(a), stream_tail(b)));
}

const B = pair(1, () => mul_streams(B, integers));
```

$2^0 \, 2^1 \, 2^2 \, 2^3 \, 2^4 \ldots 2^n$
→ list(1, 2, 4, 8, 16, ...).
every no. = 2x prev num.

0! 1! 2! 3! 4! 5! ... n!
→ list(1, 1, 2, 6, 24, 120, ...). every num = integer at that pos +1 × prev. number.

Using streams to represent power series

2. The following power series

$$
\begin{aligned}
e^x &= 1 + x + \frac{x^2}{2} + \frac{x^3}{3\cdot 2} + \frac{x^4}{4\cdot 3\cdot 2} + \cdots \\
\cos x &= 1 - \frac{x^2}{2} + \frac{x^4}{4\cdot 3\cdot 2} - \cdots \\
\sin x &= x - \frac{x^3}{3\cdot 2} + \frac{x^5}{5\cdot 4\cdot 3\cdot 2} - \cdots
\end{aligned}
$$

can be represented as streams of infinitely many terms. That is, the power series

$$a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots$$

will be represented as the infinite stream whose elements are $a_0, a_1, a_2, a_3, \ldots$.[1]

[1] In this representation, all streams are infinite: a finite polynomial will be represented as a stream with an infinite number of trailing zeroes.

Why would we want such a method? Well, let's separate the idea of a series representation from the idea of evaluating a function. For example, suppose we let $f(x) = \sin x$. We can separate the idea of evaluating f, e.g., $f(0) = 0, f(.1) = 0.0998334$, from the means we use to compute the value of f. This is where the series representation is used, as a way of storing information sufficient to determine values of the function. In particular, by substituting a value for x into the series, and computing more and more terms in the sum, we get better and better estimates of the value of the function for that argument. This is shown in the table, where $\sin \frac{1}{10}$ is considered.

Coefficient	x^n	term	sum	value
0	1	0	0	0
1	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$	.1
0	$\frac{1}{100}$	0	$\frac{1}{10}$	.1
- $\frac{1}{6}$	$\frac{1}{1000}$	- $\frac{1}{6000}$	$\frac{599}{6000}$	.099833333333
0	$\frac{1}{10000}$	0	$\frac{599}{6000}$	.099833333333
$\frac{1}{120}$	$\frac{1}{100000}$	$\frac{1}{12000000}$	$\frac{1198001}{12000000}$	.09983341666

The first column shows the terms from the series representation for sine. This is the infinite series with which we will be dealing. The second column shows values for the associated powers of $\frac{1}{10}$. The third column is the product of the first two, and represents the next term in the series evaluation. The fourth column represents the sum of the terms to that point, and the last column is the decimal approximation to the sum.

With this representation of functions as streams of coefficients, series operations such as addition and scaling (multiplying by a constant) are identical to the basic stream operations. We provide series operations, though, in order to implement a complete power series data abstraction:

```
function add_streams(s1, s2) {
    return is_null(s1)
        ? s2
        : is_null(s2)
            ? s1
            : pair(head(s1) + head(s2),
                   () => add_streams(stream_tail(s1),
                                     stream_tail(s2)));
}

function scale_stream(c, stream) {
    return stream_map(x => c * x, stream);
}

const add_series = add_streams;

const scale_series = scale_stream;

function negate_series(s) {
    return scale_series(-1, s);
```

```
}

function subtract_series(s1, s2) {
    return add_series(s1, negate_series(s2));
}
```

We also provide two ways to construct series. The function `coeffs_to_series` takes a list of initial coefficients and pads it with zeroes to produce a power series. For example,

```
coeffs_to_series(list(1,3,4))
```

produces the power series $1 + 3x + 4x^2 + 0x^3 + 0x^4 + \ldots$.

```
function coeffs_to_series(list_of_coeffs) {
    const zeros = pair(0, () => zeros);

    function iter(list) {
        return is_null(list)
            ? zeros
            : pair(head(list),
                   () => iter(tail(list)));
    }
    return iter(list_of_coeffs);
}
```

The function `fun_to_series` takes as argument a function p of one numeric argument and returns the series

$$p(0) + p(1)x + p(2)x^2 + p(3)x^3 + \cdots$$

The definition requires the stream `non_neg_integers` to be the stream of non-negative integers: $0, 1, 2, 3, \ldots$.

```
function fun_to_series(fun) {
   return stream_map(fun, non_neg_integers);
}
```

To get some initial practice with streams, write definitions for each of the following:

- `alt_ones`: the stream $1, -1, 1, -1, \ldots$ in as many ways as you can think of.
- `zeros`: the infinite stream of 0's. Do this using `alt_ones` in as many ways as you can think of.

Now, show how to define the series:

$$\begin{array}{rcl} S_1 & = & 1 + x + x^2 + x^3 + \cdots \quad x^0 + x^1 + x^2 + \cdots \\ S_2 & = & 1 + 2x + 3x^2 + 4x^3 + \cdots \end{array}$$

Turn in your definitions and a couple of coefficient printouts to demonstrate that they work.

const alt_ones = pair(1, () => pair(-1, () => alt_ones));

const zeros = pair(1 - head(alt_ones),
() => pair(1 + head(stream_tail(alt_ones)),
() => zeros));

= subtract_series(alt_ones, alt_ones);

= add_streams(alt_ones, negate(alt_ones));
or stream_tail.

```
function S_1(x) {
    let pow = -1;
    function helper() {
        pow = pow+1;
        return pair(math_pow(x, pow), () => helper());
    }
    return helper();
}
```

one_stream'

```
function S_2(x) {
    let p = -1;
    function helper() {
        p = p+1;
        return pair((p+1) * math_pow(x, p), () => helper());
    }
    return helper();
}
```

integers_from(1).

non_neg_integers = integers_from(0).