COMP2212 Programming Language Concepts Coursework

Semester 2 2015/16

Introduction

Streams are potentially unbounded sequences of data. They are common in mathematics, computer science and everyday life: e.g. a streaming movie on the web, a sequence of bits flowing through a wire in a digital circuit, or temperature measurements outside of building 53 taken every hour. The theory of (discrete) signal processing deals with infinite sequences of reals. Similarly, in computer science, dataflow is the study of networks of nodes and channels through which a potentially unbounded number of data elements flow.

Your task is to *design and implement* a domain specific programming language that performs computations on potentially unbounded integer sequences. You are required to (1) invent an appropriate syntax and (2) write an interpreter (possibly using `ocamllex` and `ocamlyacc` for lexing and parsing). Your design should be guided by the five example problems listed below, for which you will be required to produce *source files in your own language* that solve them.

The specification is left deliberately loose. If we haven't specified something precisely, it is a design decision for you to make: e.g. how to handle syntax errors, illegal inputs, etc. A significant part (30%) of the mark will be awarded for these qualitative aspects, which includes the elegance of your design. The remaining 70% of the mark will be awarded for correctly solving a number of problems using your programming language. The correctness of your solution will be checked with a number of automated tests.

Problems

We will model streams using finite sequences of integers and, in general, inputs will take the following form:

```
a11 a21 a31 a41 ... an1
a12 a22 a32 a42 ... an2
.
.
.
a1k a2k a3k a4k ... ank
```

where every line is terminated with the new line character `EOL` and the file ends with `EOF`. The `a`'s are arbitrary (positive or negative) 32 bit integers (you can safely use a 64 bit implementation, we will not test for architecture specific overflow behaviour). The number n is the number of input sequences, which are the individual columns in the input. You can assume that all of the input streams will have the same length, or in other words, that all of the columns will be of the same size. For example, in the above input, the second sequence is

```
a21 a22 a23 ... a2k
```

The values in each row are separated by a single space character. For example, the following is an input that describes 2 sequences of length 3:

2 1
3 −1
4 1

Each problem specifies how many sequences you are expected to produce on `stdout`. For example, if a particular problem specifies two output sequences, then your output is expected be of the form:

b11 b21
b12 b22
b11 b22
.
.
.
b1l b2l

where the length of the output—that is, the number of lines—will usually depend on the length of the input. Every line must be terminated with the Unix new line character `EOL` (i.e. **not** Windows `CR+LF`).

Problem 1 - Prefixing

Take a sequence $a_1\ a_2\ a_3\ a_4\ a_5\ \ldots$ as an input and output the sequence $0\ a_1\ a_2\ a_3\ \ldots$, that is, the sequence that is the same as the input sequence, but starting with a single 0 character.

Example input:	Expected output:
1	0
2	1
3	2
4	3
	4

Problem 2 - Copying

Take a sequence $a_1\ a_2\ a_3\ a_4\ a_5\ \ldots$ as an input and output two copies of it.

Example input:	Expected output:
−5	−5 −5
0	0 0
3	3 3

Problem 3 - Stream arithmetic

Take two sequences $a_1\ a_2\ a_3\ a_4\ \ldots$ and $b_1\ b_2\ b_3\ b_4\ \ldots$, and produce the sequence

$$a_1 + 3b_1\ a_2 + 3b_2\ a_3 + 3b_3\ a_4 + 3b_4\ \ldots$$

Example input:	Expected output:
1 5	16
2 4	14
3 3	12
4 2	10
5 1	8

Problem 4 - Accumulator

Take a sequence $a_1\ a_2\ a_3\ a_4\ \ldots$ and output the sequence $a_1\ a_1+a_2\ a_1+a_2+a_3\ a_1+a_2+a_3+a_4\ \ldots$, where each term of the output is the sum of all the input terms up to that point.

Example input:	Expected output:
1	1
2	3
3	6
4	10

Problem 5 - Fibonacci

Take a sequence $a_1\ a_2\ a_3\ a_4\ a_5 \ldots$ and output the sequence

$$a_1\ a_1 + a_2\ 2a_1 + a_2 + a_3\ 3a_1 + 2a_2 + a_3 + a_4\ 5a_1 + 3a_2 + 2a_3 + a_4 \ldots$$

where the coefficients of each input term in the sums follows the Fibonacci series 1 1 2 3 5 8 ... from when it first appears. Recall that the Fibonacci series starts with two 1s and then the subsequent terms are always the sum of the previous two.

Example input:	Expected output:
1	1
0	1
0	2
0	3
0	5

Example input:	Expected output:
1	1
2	3
3	7
4	14
5	26

Submission instructions

First submission - due Thursday March 10 4pm

You will be required to submit a zip file containing:

- the sources for the interpreter for your language, written in OCaml
- five programs, written in **YOUR** language, that solve the five problems specified above. The programs should be in files named `pr1.spl`, `pr2.spl`, `pr3.spl`, `pr4.spl`, `pr5.spl`.

We will compile your interpreter using the command `make` on linuxproj, so you will need to include a Makefile in your zip file that produces an executable named `mysplinterpreter`. Prior to submission, you are required to make sure that your interpreter compiles **on linuxproj**: if your code does not compile then you will be awarded 0 marks.

Your interpreter is to take a file name (that contains a program in your language) as a single command line argument and then expect input on standard input (`stdin`). The interpreter should produce any output on standard output (`stdout`) and any error messages on standard error (`stderr`).

For each problem, we will test whether your code performs correctly by using a number of tests. We only care about correctness and performance will not be assessed (within reason). You can assume that for the tests we will use correctly formatted input.

For example, when assessing your solution for problem 1 we will run

```
mysplinterpreter pr1.spl < input
```

where `input` is be a textfile containing the input for a particular test.

All submissions will be done through handin. The precise submission instructions will be released a few days before the deadline. The marks awarded for the first submission will count 20% of the total coursework mark (4% for each of the five problems). You will receive feedback based on the our testing prior to the second deadline.

Second submission - due Thursday April 28 4pm

Shortly after the first deadline we will release a further five problems. You will be required to submit two separate files.

First, you will need to submit a zip file containing programs (`pr6.spl`, `pr7.spl`, `pr8.spl`, `pr9.spl`, `pr10.spl`) written in your language that solve the additional problems. We will run our tests on your solutions and award marks for solving the additional problems correctly. This will form 50% of the total coursework mark (10% each for the five additional problems). You will have the option of resubmitting the interpreter, for a 50% penalty on this component. Thus, if you decide to resubmit your interpreter in the second submission the maximum possible total coursework mark is capped at 75%.

Second, you will be required to submit a 3 page user manual for your language in pdf format that explains the syntax, and describes any additional features (programmer convenience, type checking, informative error messages, etc.) This report, together with the five programs will be evaluated qualitatively and your marks will be awarded for the elegance and flexibility of your solution and the clarity of the user manual. These qualitative aspects will be worth 30% of the total coursework mark.

As you know, the coursework is to be done in pairs. As part of the second submission we will require a declaration of how marks are to be distributed amongst the members of your group. You will receive all feedback and your marks by Thursday May 12.