COMP 212 Spring 2015
Lab 10

1 Survey

Please fill out the following survey: https://www.surveymonkey.com/r/comp212

2 Unit

The type `unit` represents an "empty tuple", and has value `()`. It is useful for functions that do their work imperatively rather than functionally.

3 Input and output

In this lab, you will use functions from the `TextIO` structure; see https://www.cs.princeton.edu/~appel/smlnj/basis/text-io.html.

The types `TextIO.instream` and `TextIO.outstream` represent "something you can read from" and "something you can write to", respectively.

Here are some ways of making input and output streams:

- `TextIO.stdIn : TextIO.instream` read something you type at the terminal
- `TextIO.stdOut : TextIO.outstream` write output to the terminal

Here are some functions for reading and writing:

- `TextIO.inputLine : TextIO.instream -> string option` read a line of input
- `TextIO.output : TextIO.outstream * string -> unit` write a string

Task 3.1 In `smlnj`, try out these functions, using them to read and write from the console.

Task 3.2 Write a function

```
val copy : TextIO.instream -> TextIO.outstream -> unit
```

that copies the entire input stream to the output stream. Try it out interactively (you will need to use `C-c` to stop).

The following functions create input and output streams from files:

- `TextIO.openIn : string -> TextIO.instream`
- `TextIO.openOut : string -> TextIO.outstream`
  WARNING: overwrites the file specified by the file name

Task 3.3 Write a function

```
val copy_files : string -> string -> unit
```

that takes two filenames and copies the contents of the first to the second.

Task 3.4 Try this out on some file. Open the second file; what do you see? Now quit SMLNJ; what do you see?

Task 3.5 The problem is that writes to a file are *buffered* and not necessarily done when you say to do them. Add a call to `TextIO.flushOut` to your file copy function to force the writes to be done at the end of that function.

4 Mapreduce on a file

The signature

```
signature MAP_REDUCE =
sig
    type 'a mapreducable
    val mapreduce :
           ('a -> 'b)      (* handle single element *)
        -> 'b              (* result for empty *)
        -> ('b * 'b -> 'b) (* merge results: must be associative and commutative *)
        -> 'a mapreducable -> 'b
end
```

represents a data source that we can do a map-reduce on.

We can implement this signature using a `TextIO.instream` (which can stand for a file or for the console). However, to think of a file or the console as an `'a mapreducable` for some specific type `'a`, we need to have a way to convert lines of the file into `'a`'s. Thus, we say that the type

```
TextIO.instream * (string -> 'a)
```

is map-reducable, where the second component of the pair is used to parse each **line** of the file into a piece of data.

Task 4.1 Implement the `mapreduce` function in `FileMR`. **Your implementation should not use any space beyond what is necessary to store the `'b` values that are produced—in particular, it should not use linear stack space.**

Task 4.2 Make a value

```
val numbersFromStdIn : int FileMR.mapreducable
```

that reads numbers from `TextIO.stdIn`: for each line the user types, if it is a number, then include it in the data, otherwise include `0` for that line. Hint: use `Int.fromString`.

Task 4.3 Write a function

```
val add : int FileMR.mapreducable -> int
```

that adds the numbers in an `int mapreducable`.

Task 4.4 Test this on `numbersFromStdIn`. Note that you will need to type `Control-d` to signal the end of input, which will also (unfortunately) quit SMLNJ.

Task 4.5 Make an `int FileMR.mapreducable` that reads from a file and test your function on a file too.