CPSC 201: Introduction to Computer Science
Carsten Schürmann
Date: January 27, 2003

Homework 2

Due: Wednesday, January 29, 2003.

Guidelines

While we acknowledge that beauty is in the eye of the beholder, you should nonetheless strive for elegance in your code. Not every program which runs deserves full credit. Make sure to state invariants in comments which are sometimes implicit in the informal presentation of an exercise. If auxiliary functions are required, describe concisely what they implement. Do not reinvent wheels, and try to make your functions small and easy to understand. Use tasteful layout and avoid long winded and contorted code. None of the problems requires more than a few lines of SML code.

Exercise 1 Recall the datatype for the gameboard we have discussed in class:

```
datatype Board
  = Tower of Tower
  | Board of Board * Board
           * Board * Board
```

On way of addressing the individual fields on this board is by an object of the datatype `Address`, which is defined as follows.

```
datatype Address
  = Here
  | UpperLeft of Address
  | UpperRight of Address
  | LowerLeft of Address
  | LowerRight of Address
```

To compute the *relative* `Address` of a Tower on a `Board` do the following. When you start, the address is `Here`. If you have a board in front you, and the tower you are looking for is in the upper left square, all you have to do is to prefix the address of the tower in that sqaure with the constructor `UpperLeft`. The other three squares can be denoted similarly. This might not look easy at first glance, but it is. It is just recursive.

Another way to describe an address on a board is as an *absolute* pair of integer numbers. The upperleft square is numbered `(1,1)`, and the lowerright square `(n, n)`.

The board is always a perfect square of size n. It will have $2^n x 2^n$ fields for all $n \geq 1$.

1. Write a function `absoluteToRelative` that computes a relative `Address` from a Pair of size and absolute address of type `int * int`.

```
absoluteToRelative : int * (int * int) -> Address
```

2. Write the reverse function `RelativeToAbsolute`.

```
relativeToAbsolute : Address -> int * int
```

Hint: Write a helper function `Address -> int * (int * int)` that returns a pair of the size of the current board and an absolute address.

3. Prove formally that for all addresses A the following holds:

`absoluteToRelative (relativeToAbsolute` A`)` $\stackrel{*}{\Longrightarrow} A$.

Exercise 2

1. Write a function

```
intToRoman : int -> RomanNumber
```

that converts integers between 1 and 999 to their Roman numeral equivalent. Roman numbers should be elements of the following datatype

```
datatype RomanDigit
  = I | V | X | D | L |  C | M
datatype RomanNumber
  = Empty
  | Cons of RomanDigit * RomanNumber
```

The following chart summarizes how integer digits are written as Roman numerals are written.

	1	2	3	4	5	6	7	8	9
ones	I	II	III	IV	V	VI	VII	VIII	IX
tens	X	XX	XXX	XL	L	LX	LXX	LXXX	XC
hundreds	C	CC	CCC	CD	D	DC	DCC	DCCC	CM

Let's use the number 843 as an example of how the conversion takes place. First, we look up the hundreds digits (8) in the hundreds row and find DCCC. Second we look up the thens digits (4) and the ones digit (3) and find XL and III, respectively. Finally, we concatenate the three together to form the final Roman numeral: DCCCXLIII. *Hint:* To create the function, break the problem into subproblems (simpler functions). One particular way of partitioning this problem is given below. Since this is not the only good partitioning, you are encouraged to come up with your own solution.

```
hundreds    : int -> RomanNumber
tens        : int -> RomanNumber
ones        : int -> RomanNumber
intToRoman  : int -> RomanNumber
```

2. Write a function that does the inverse of what `intToRoman` does:

```
romanToInt : RomanNumber -> int
```