# CSE 341, Spring 2011, Assignment 1
# Due: Thursday 7 April, 11:00PM

Write a series of SML functions to solve the following problems. Your answers should be collected in a single SML source file named `hw1.sml`. Your solutions should use basic SML features corresponding to those covered in class so far and described in the Ullman book up through sec. 3.2. These include expressions, top-level bindings, tuples, lists, and functions. You should not use pattern matching or `let` in this assignment — we'll have plenty of time to practice those later. You also must not use references or arrays or other non-functional constructs. You are free to define additional "helper" functions if you like.

Programs should be written in good style, with appropriate names, clear indentation, useful comments and so forth. The comment describing each function should include the problem number.

1. Write a function `sumTo` that takes an integer n as an argument and that computes the sum of the first `n` reciprocals:

$$\sum_{i=1}^{n} \frac{1}{i}$$

   For example, `sumTo`(2) should return 1.5. The function should return 0.0 if `n` is 0. You may assume that the function is not passed a negative value of `n`.

2. Write a function `repeat` that takes a string and an integer as arguments and that returns a string composed of the given number of occurrences of the string. For example, `repeat`("abc", 3) should return "abcabcabc". You may assume that the function is not passed a value for the second argument that is negative.

3. Write a function `twos` that takes a single integer argument and that returns the number of factors of two in the number. For example, the number 84 when expressed as a product of prime factors is 2 * 2 * 3 * 7, which means that it has 2 factors of 2. Similarly the number ~30 would be expressed as ~(2 * 3 * 5), which means that it has 1 factor of 2. You may assume that the value passed to the function is not equal to 0.

4. Write a function `numNegative` that takes a list of integers as an argument and that returns a count of the number of negative integers in the list. For example, `numNegative`([3, 17, ~9, 34, ~7, 0]) should return 2.

5. Write a function `absList` that takes a list of `int * int` tuples and that returns a new list of `int * int` tuples where every integer is replaced by its absolute value. For example, `absList`([(~38, 47), (983, ~14), (~17, ~92), (0, 34)]) should return [(38, 47), (983, 14), (17, 92), (0, 34)]. This is easier to solve if you write a helper function to process one tuple.

6. Write a function `split` that takes a list of integers as an argument and that returns a list of the tuples obtained by splitting each integer in the list. Each integer should be split into a pair of integers whose sum equals the integer and which are each half of the original. For odd numbers, the second value should be one higher than the first. For example, `split`([5, 6, 8, 17, 93, 0]) should return [(2, 3), (3, 3), (4, 4), (8, 9), (46, 47), (0, 0)]. You may assume that all of the integers in the list passed to the function are greater than or equal to 0.

7. Write a function `getNth` that takes a list of ints and an `int` $n$ and returns the $n^{th}$ element of the list where the head of the list is $1^{st}$. If the list has too few elements, your function should apply `hd` to the empty list, which will raise an exception.

8. Write a function `daysInMonth` that takes an integer representing a month as an argument and that returns the number of days in that month. You may assume that the month value passed is between 1 and 12 inclusive. You may also assume that the month is not part of a leap year.

9. Write a function `absoluteDay` that takes two integers representing a month and day as parameters and returns the absolute day of the year between 1 and 365 that is represented by that month/day. For example, Jan 1 is absolute day #1; Jan 2 is #2; Jan 31 is #31; Feb 1 is #32; and so on, up to Dec 31, which is #365. You may assume that the month value passed is between 1 and 12 inclusive and that the day value passed is a positive integer. You may also assume that the date is not part of a leap year.

10. Write a function `numberBeforeReachingSum` that takes an `int` (which you can assume is non-negative) and an `int list` and returns an `int`. It returns the number of elements, starting from the beginning, in the list that add up to at most `sum` without going over. If `sum` is greater than the sum of all numbers in the list, your function should apply `hd` to the empty list, which will raise an exception.

11. Write a function `whatMonth` that takes a day of year (i.e., a number between 1 and 365) and returns what month that day is in (1 for January, 2 for February, etc.). Use a list holding 12 integers and your answer to the previous problem.

12. Write a function `isSorted` that takes a list of integers and that returns whether or not the list is in sorted (nondecreasing) order (true if it is, false if it is not). By definition, the empty list and a list of one element are considered to be sorted.

13. Write a function `insert` that take an int and a sorted int list as parameters and that returns the list obtained by inserting the int into the list so as to preserve sorted order. For example, `insert`(8, [1, 3, 7, 9, 22, 38]) should return [1, 3, 7, 8, 9, 22, 38].

14. Write a function `isort` that takes an int list as a parameter and that returns the list obtained by sorting the list. In writing isort, you should call your insert function from the previous problem. The result will be an ML solution to the classic insertion sort algorithm.

**Type Summary**
Evaluating a correct homework solution should generate these bindings:

```
val sumTo = fn : int -> real
val repeat = fn : string * int -> string
val twos = fn : int -> int
val numNegative = fn : int list -> int
val absList = fn : (int * int) list -> (int * int) list
val split = fn : int list -> (int * int) list
val getNth = fn : int list * int -> int
val daysInMonth = fn : int -> int
val absoluteDay = fn : int * int -> int
val numberBeforeReachingSum = fn : int * int list -> int
val whatMonth = fn : int -> int
val isSorted = fn : int list -> bool
val insert = fn : int * int list -> int list
val isort = fn : int list -> int list
```

Of course, generating these bindings does not guarantee that your solutions are correct. *Test your functions.*

**Assessment**
Your solutions should be:

- Correct
- In good style, including indentation and line breaks
- Written using the features we have used in class so far as outlined at the beginning of the assignment.

**Turn-in Instructions**

- Put all your solutions in one file named `hw1.sml`.
- The first line of your `.sml` file should be an ML comment with your name and the phrase `homework 1`.
- Use the regular Catalyst dropbox to submit your assignment.