15-150 Fall 2016
Homework 01

Out: Wednesday, 31 August 2016
Due: Wednesday, 7 September 2016 at 23:59 EST

1 Introduction

Welcome to 15-150! This assignment introduces the course infrastructure and the SML runtime system, then asks some simple questions related to the first week of lectures and lab.

1.1 Getting The Assignment

The starter files for the homework assignment have been distributed through our `git` repository. To learn how to use it, read the documentation at

```
http://www.cs.cmu.edu/~15150/resources/handouts/git.pdf
```

If you still need help, ask a TA promptly and get started on the non-code questions.

In the first lab, you set up a clone of this repository in your AFS space. To get the files for this homework, log in to one of the UNIX servers via SSH or sit down at a cluster machine, change into your clone of the repository, and run

```
git pull
```

This should add a directory for Homework 1 to your copy of the repository, containing a copy of this PDF and some starter code in subdirectories. If this does not work for you, contact course staff immediately.

1.2 Submission

Submissions will be handled through Autolab, at

```
https://autolab.cs.cmu.edu
```

In preparation for submission, your `hw/01` directory should contain a file named exactly `hw01.pdf` containing your written solutions to the homework.

To submit your solutions, run `make` from the `hw/01` directory (that contains a `code` folder and a file `hw01.pdf`). This should produce a file `hw01.tar`, containing the files that should be handed in for this homework assignment. Open the Autolab web site, find the page for this assignment, and submit your `hw01.tar` file via the "Handin your work" link.

The Autolab handin script does some basic checks on your submission: making sure that the file names are correct; making sure that no files are missing; making sure that your code compiles cleanly. Note that the handin script is *not* a grading script—a timely submission that passes the handin script will be graded, but will not necessarily receive full credit. You can view the results of the handin script by clicking the number (usually either 0.0 or 1.0) corresponding to the "check" section of your latest handin on the "Handin History" page. If this number is 0.0, your submission failed the check script; if it is 1.0, it passed.

Remember that your written solutions must be submitted in PDF format—we do not accept MS Word files or other formats.

Warning : On this homework and all future homeworks, we reserve the right to refuse to grade or to otherwise penalize submissions that do not follow specified formatting or the instructions in the handout. If your code does not compile you may receive a zero on those sections of the homework. Please contact course staff if you have any questions. If you attempt to contact us close to the deadline, please be aware that we may not be able to respond before the deadline. If you cannot access the Autolab site, notify the course staff immediately.

1.3 Due Date

This assignment is due on Wednesday, 7 September 2016 at 23:59 EST. Remember that you may use a maximum of one late day per assignment, and that you are allowed a total of three late days for the semester.

1.4 Methodology

In future assignments, you must use the five step methodology for writing functions, for **every** function you write. In this assignment, steps 1-4 of the five step methodology have already been completed for every function, except for the function in Task 7.1. **You must use the five step methodology for the function you write in Task 7.1**. In addition, you must complete step 5 for **every** function you write in this assignment.

The five step methodology:

1. In the first line of comments, write the name and type of the function.
2. In the second line of comments, specify via a `REQUIRES` clause any assumptions about the arguments passed to the function.
3. In the third line of comments, specify via an `ENSURES` clause what the function computes (what it returns).
4. Implement the function.
5. Provide testcases, generally in the format
   `val <return value> = <function> <argument value>`.

For example, for the factorial function:

```
(*  fact : int -> int
 *  REQUIRES:  n >= 0
 *  ENSURES: fact n evaluates to n!
 *)

fun fact (0 : int) : int = 1
  | fact (n : int) : int = n * fact(n-1)

(* Tests: *)
val 1 = fact 0
val 720 = fact 6
```

2 Course Resources

Please make sure you have access to the various course resources. We will post important information often. You can find more information about these resources in the Tools page of the course's Web site.

We are using Web-based discussion software called Piazza for the class. You are encouraged to post questions, but please do not post anything that gives away answers or violates the academic integrity policy. If you think that your question might give away answers, you can make it a *private* question, visible only to the course staff.

Task 2.1 (3 pts). You should have received an e-mail message with instructions on signing up for Piazza. Activate your account. There is announcement there that includes a picture. In one sentence, what is it of?

3 digitsum

Consider the following ML function definition:

```
fun digitsum (n : int) : int =
    if n < 10 then n else (n mod 10) + digitsum (n div 10)
```

This introduces a function named `digitsum`, of type `int -> int`.

Task 3.1 (4 pts). What are the types and values of the following expressions?
If the expression is not well typed, say why.

```
(a) digitsum 12345
(b) digitsum 12345.0
(c) digitsum (digitsum 12345)
(d) digitsum digitsum 12345
```

Task 3.2 (5 pts). We can completely characterize a function `f` of type `int -> int` by specifying its applicative behavior: for each integer `n` we say what integer `f(n)` evaluates to, or whether `f(n)` fails to terminate.

Specify the applicative behavior of `digitsum` (meaning, give a formal description of what it does). It may help to refer to decimal notation: recall that every non-negative integer `n` has a decimal representation of the form $\mathtt{d}_k \ldots \mathtt{d}_1\mathtt{d}_0$, where $k \geq 0$, each $\mathtt{d}_i$ is a decimal digit, i.e. $0 \leq \mathtt{d}_i \leq 9$, and $n = 10^k\mathtt{d}_k + \cdots + 10\mathtt{d}_1 + \mathtt{d}_0$.

4 Totality

Task 4.1 (6 pts). A function `f` of type `int -> int` is said to be *total* if `f(n)` terminates for all integers `n`. Which of the following functions is/are total?
(No need for detailed proofs, but give brief justifications.)

```
(a) digitsum
(b) fn (x:int) => digitsum (x+1)
(c) fn (x:int) => digitsum (digitsum x)
```

5 Calendar calculations

We can represent a date as a triple consisting of a day number, a month number, and a year number. A triple `(d, m, y)` is said to be *valid* (for the Gregorian calendar) if $1 \leq \texttt{m} \leq 12$, $\texttt{y} \geq 1582$, and `d` is an allowed day number for month `m` of year `y`. January is the first month. The following quotes from Wikipedia are relevant:

> A year that is exactly divisible by four is a leap year, except for years that are exactly divisible by 100, but these centurial years are leap years if they are exactly divisible by 400. For example, 1700, 1800, and 1900 are not leap years, but 2000 is.
>
> Thirty days hath September,
> April, June, and November.
> All the rest have thirty-one,
> Excepting February alone,
> Which hath but twenty-eight days clear,
> And twenty-nine in each leap year.

Task 5.1 (5 pts). Write an ML function `is_leap_year` of type `int -> bool` such that when $\texttt{y} \geq 1582$, `is_leap_year y` evaluates to `true` if year `y` is a leap year, and evaluates to `false` otherwise.

Task 5.2 (5 pts). Write an ML function `month_length` of type `int * int -> int` such that when $\texttt{y} \geq 1582$ and $1 \leq \texttt{m} \leq 12$, `month_length (m, y)` evaluates to the number of days in the $\texttt{m}^{th}$ month of year `y`.

Task 5.3 (5 pts). Write an ML function `is_valid_date` of type `int * int * int -> bool` such that `is_valid_date (d, m, y)` evaluates to `true` if `(d, m, y)` is a valid date,and evaluates to `false` otherwise.

Your code should fit int the following template:

```
fun is_leap_year (y : int) : bool = (* code here *)
(* Test cases for is_leap_year here *)

fun month_length (m:int, y:int) : int = (* code here *)
(* Test cases for month_length here *)

fun is_valid_date (d:int, m:int, y:int) : bool = (* code here *)
(* Test cases for is_valid_date here *)
```

You can use any built-in arithmetical or boolean constructs of ML.
In particular you can use `andalso`, `if-then-else` and `case`-expressions.
Remember, even though steps 1-4 of the five step methodology are already completed, you must write test cases for these functions.

6 Brent's Theorem

A famous result due to Richard Brent says that a job that has "work" w and "span" s can be done on a machine with p processors in $max(w/p, s)$ steps. We'll say a lot more about work and span later in the semester, but you don't need to understand what these terms mean yet. The work and span of a job are positive integers, and the math notation above uses `/` to stand for "division" of integers (which produces a real-valued result), whereas in ML we need to keep a clear distinction between real numbers and integers. Indeed, ML uses `/` for real number division and `div` for integer division, e.g.

```
3 div 2 = 1
3.0 / 2.0 = 1.5
```

and mixing up the types here can easily result in ill-typed code, e.g.

```
3 div 2.0   (* not well typed *)
3 / 2     (* not well typed *)
```

Similarly ML has two "maximum" functions

```
Real.max : real * real -> real
Int.max : int * int -> int
```

ML also has functions for converting from integers to reals, and for converting from reals to integers (rounding down):

```
real : int -> real
floor : real -> int
```

Here is an ML function `brent_i` of type `int * int * int -> int`
that computes Brent's quantity with integer arithmetic:

```
fun brent_i (w:int, s:int, p:int):int = Int.max(w div p, s)
```

Task 6.1 (6 pts). Write an ML function `brent_r : int * int * int -> int`
that computes the same quantity using real arithmetic (without using `div` or `Int.max`, and without calling `brent_i`). Your code should fit the following template:

```
fun brent_r (w:int, s:int, p:int):int = (* code here *)
(* Test cases here *)
```

Task 6.2 (6 pts). Suppose we know that for all positive integers `x` and `y`, and all real numbers `a` and `b`, the following equations hold:

```
(1)   floor (real x) = x
(2)   floor ((real x)/(real y)) = x div y
(3)   floor (Real.max(a, b)) = Int.max(floor a, floor b)
```

Prove that your function `brent_r` produces the same answers as `brent_i` when applied to a triple of positive integers.

7 Reverse engineering

Task 7.1 (5 pts). Write a (non-recursive!) ML function `foo` of type `int -> int` such that

```
   n     0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
foo(n)   0 1 2 3 4 5 6 7 8 9  1  2  3  4  5  6  7  8  9  1  2
```

and the pattern continues, so that `foo(n) = foo(n-9)` for `n>18`.

Your answer should fit the following template:

```
val foo : int -> int = fn (n:int) => (* code here *)
(* Test cases here *)
```

Remember to complete the entire five step methodology for `foo`, including comments for the name and type of `foo`, the `REQUIRES` clause, and the `ENSURES` clause. Don't forget to include test cases for `foo` directly below its implementation.

This assignment has a total of 50 points.