Computer Science 15-212, Spring 2003

Assignment 1

DUE: Wednesday, January 29, 2003, 2:12 A.M.
papers due at recitation.

CAUTION: Due Tuesday night / early Wednesday morning.

[**Maximum points: 50**]

Handin instructions

- Put your `SML` code into a file named `hw1.sml`. All of your definitions and comments should be in this one file. Copy this file to your `hw1` handin directory. Your `hw1` handin directory is `/afs/andrew/course/15/212sp/handin/`*your-Andrew-ID*`/hw1/`
- Please type your name, userid, and your recitation instructor's name in a comment at the top of your file.
- Hand in your solutions to Questions 1.5, 2.1, 2.2, and 3.2 either written on paper at recitation or as (properly formatted) comments in your code file. Please write your name, userid, and recitation instructor on all written work.

Guidelines

- You should only use the functional aspects of `SML`. Your code should contain no mutation or imperative constructs.
- Be sure to comment your code as described in class.
- Strive for elegance! 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. Do try to make your functions small and easy to understand. Use tasteful layout and avoid longwinded and contorted code. None of the problems requires a very large number of lines of `SML` code.
- Make sure that your entire file compiles and runs. If you are unable to complete portions of the assignment, comment out the part of the code that does not work properly, and explain what you did, what worked, and what didn't. It is your responsibility to explain as carefully as you can why you think you were unable to get the code working, what you think is wrong, and how you might go about fixing it. The quality of such an explanation will be important to us in deciding whether to give you partial credit.
- **WARNING:** If your file is not named `hw1.sml` or if it does not load, you will lose 20 points. E.g., a misnamed file that is otherwise perfect will receive a score of 30.
- Homeworks must be all your own work. Read the policy on cheating in the syllabus.
- Late homeworks will be accepted *only* until the start of lecture on Thursday, with a 25% penalty. See the syllabus for the complete policy on late assignments.
- If you have any questions about this assignment, contact Michael Erdmann (me@cs.cmu.edu) or use the newsgroup **cmu.andrew.academic.cs.15-212.discuss**

Problem 1: Warm-Up Exercises (16 points)

These exercises are designed for you to explore SML's basic functions and libraries.

Question 1.1 (3 points)

Write a function `check : int -> bool` that decides whether its integer argument is a strictly negative number whose value is 7 more than a multiple of 13. For instance:

```
check(~6) ⟹ true
check(~20) ⟹ false
check(20) ⟹ false
check(~123) ⟹ true
```

Question 1.2 (3 points)

Write a function `average: int * int -> real`. The function should take a pair consisting of two integers, `n` and `m`, and return their average $\frac{n+m}{2}$ as a real number. For instance,

```
average(5,10) ⟹ 7.5
average(20, 30) ⟹ 25.0
```

Question 1.3 (3 points)

Write a function `positive : (real->real) * real -> real`. The function should take a pair as argument. The first component of the pair should be a function `f` of type `real -> real` and the second component a number `x` of type `real`. The function `positive` should return the value `f(x)` whenever that value is positive; otherwise `positive` should return the value `0.0`. For instance,

```
positive(Math.sin, 2.0) ⟹ 0.909297426826
positive(Math.cos, 2.0) ⟹ 0.0
```

Question 1.4 (3 points)

Write a function `duplicate : string -> string` that takes a string as argument and returns the string concatenated with itself. For instance,

```
duplicate("here") ⟹ "herehere"
duplicate("pop") ⟹ "poppop"
duplicate("") ⟹ ""
```

Question 1.5 (4 points)

(a) Does the following expression typecheck? If so, what are its type and value?

```
fun f(x:int):int = 10 * f(x mod 10);
```

(b) Given the function `f` as above, does the following expression typecheck? If so, what are its type and value?

```
f(10);
```

Problem 2: Binding and Scope (13 points)

You should solve the next two questions in your head, without first trying them out in `SML`. The type of mental reasoning involved in answering these questions should become second-nature.

Question 2.1 (8 points)

Consider the code fragment:

```
(1)        val x:int = 5;

(2)        fun convolute(x:int):(int*int*int) =
(3)          let
(4)             val y:int = let
(5)                            val x:int = 2
(6)                            val y:int = 7*x
(7)                            val y:int = 3*y
(8)                            val x:int = 100
(9)                         in
(10                            y
(11)                        end
(12)         in
(13)            (x, y, x)
(14)         end;

(15)      convolute(17);
```

(a) To which value is the identifier `x` in line (6) bound? Briefly explain why.

(b) To which values are the two identifiers `x` in line (13) bound? Briefly explain why.

(c) To which value is the identifier `y` in line (13) bound? Briefly explain why.

(d) What is the value returned by the function call in line (15)? Briefly explain why.

Question 2.2 (5 points)

Consider the code fragment:

```
(1)        val k:int = 7;
(2)        fun diff(m:int, n:int):int = (m - n) mod k;
(3)        val k:int = 10;
(4)        val d:int = diff(35, k);
```

What is the value of `d`? Briefly explain why.

Problem 3: Recursion and Induction (16 pts)

One of the basic techniques you will need to master quickly in this course is the ability to express iterative ideas recursively. This problem gives you some practice doing so.

Question 3.1 (5 pts)

Consider the problem of adding up the values of some function $f(x)$ evaluated on the first k positive integers:

$$f(1) + f(2) + \cdots + f(k) \qquad \text{with } k \geq 1.$$

Implement a recursive function `sum: int * (int->real) -> real` that computes this sum. For instance,

`sum(1000, fn (n:int) => 1.0 / (real (n*n)))` $\Longrightarrow$ 1.644 (approximately).[1]

You should assume and state in your specifications that $k \geq 1$.

Question 3.2 (6 pts)

Using induction on k, prove that your code for Question 3.1 correctly implements the function `sum` as specified, that is, `sum(k, f) ==> f(1) + ... + f(k)`.

Be sure to state the following four items clearly *before* you start working on the details of your proof:

1. The overall Theorem.
2. The Base Case.
3. The Inductive Hypothesis.
4. What you are trying to prove in the Inductive Step.

Also, state clearly where you use the Inductive Hypothesis in the proof of the Inductive Step.

[1] You probably remember from high school that $\sum_{n=1}^{\infty} 1/n^2 = \pi^2/6$, which is approximately 1.6449340668.

Question 3.3 (5 pts)

Let us make the problem marginally more complicated. Instead of adding up the values of f obtained on the first k integers, let's use an indexing function `ix` to tell us which integers to pass to f. Specifically, implement a function `sumix: int * (int-> int) * (int->real) -> real` such that

`sumix(k, ix, f)` $\Longrightarrow$ `f(ix 1) + f(ix 2) ... + f(ix k)`.

For instance, `sumix(k, fn (i:int) => i*i, f)` $\Longrightarrow$ `f(1) + f(4) ... + f(k`^{2}`)`,

and therefore: `sumix(6, fn (i:int) => i*i, fn (n:int) => real n)` $\Longrightarrow$ 91.0.

Once again, you should assume and state in your specifications that $k \geq 1$.

HINT: There is a short fairly elegant solution that uses the function `sum`.

Problem 4: Largest Factor (5 points)

A positive integer k is said to be a *factor* of another positive integer n if k divides evenly into n. Said differently, k is a factor of n if n can be written as the product $n = k * m$ for some integer m.

A given integer n may have many factors or very few. Note that 1 and n are always factors of n. In some cases (if n is prime) these are the only factors.

Write a function `maxfactor : int -> int` such that `maxfactor(n)` returns the largest factor of `n` less than `n` itself. For instance,

`maxfactor(17)` $\Longrightarrow$ 1

`maxfactor(100)` $\Longrightarrow$ 50

`maxfactor(180469)` $\Longrightarrow$ 719

You should assume (and state in your specifications) that $n > 1$.

You will probably want to define an auxiliary recursive function locally within the body of the function `maxfactor`. Be sure to write full specifications for that function as well as `maxfactor`.