CSci 148, Spring 2002, Exam 1

Question 1. ($39 = (6 \times 3) + (2 \times 3) + (2 \times 3) + (3 \times 3)$ points)

(a) What are the types of the following expressions?

- `(1.5,("3",[4,5]))`
- `[[1,2],nil,[3]]`
- `[(2,3.5),(4,5.5)]`
- `([#"a",#"b"],[nil,[true]])`
- `[SOME (), NONE]`
- `(fn x => x, fn (x,y) => x^y)`

(b) Consider the function definition

```
fun f(0,y) = y
  | f(x,y) = f(x-1,x*y);
```

- What is the type of `f`?
- What is the value of `f(2,3)`?

(c) Consider the function definition

```
fun g [] = []
  | g (NONE::xs) = g xs
  | g ((SOME x)::xs) = x::(g xs)
```

- What is the type of `g`?
- What is the value of `g [NONE,(SOME 1,(SOME 2, NONE]`?

(d) Consider the function definition

```
fun h [] ys = ys
  | h (x::xs) ys = x::(h xs ys);
```

- What is the type of `h`?
- What is the value of `h [1,2] [3,4]`?
- What is the value of `let val k = h [2] in k [1] end`?

Question 2. (13 = 3 + 4 + 6 points) Suppose that `x` and `y` are expressions of type `bool`.

(a) Write an expression that has the same value as `x andalso y`, except that it uses only `if-then-else`.

(b) Write an expression that has the same value as `x orelse y`, except that it uses only `case`.

(c) Define a function `all` of type `bool list -> bool` such that `all xs` is `true` iff all of the elements of `xs` are `true` (and thus `all xs` is `false` if at least one of the elements of `xs` is `false`).

Question 3. (14 = 4 + 10 points) Consider the following binary tree datatype:

```
datatype 'a btree = Empty | Node of 'a * 'a btree * 'a btree;
```

(a) Give an example element of type `int btree` that has two nodes.

(b) Write a function `sumprod` of type `int btree -> (int * int)` such that, if `t` is a binary tree, then `sum t` is a pair `(s,p)`, where `s` is the sum of all values in `t` and `p` is the product of all values in `t`.

Question 4. (16 points) Let us say that x is a *tiny integer* if $0 \leq x \leq 99$. The following code implements a `sumlist` function of type `int list -> int` that takes a list of tiny integers and returns the sum, except that `99` is used in case of overflow:

```
exception Overflow;
fun sum(x,y) =
    let val s = x + y in
        if s < 100 then s else raise Overflow
    end;
fun sumlist xs =
    let fun slist [] = 0
          | slist (x::xs) = sum(x, slist xs)
    in
        slist xs handle Overflow => 99
    end;
```

Suppose we eliminate the exception `Overflow` and rewrite the `sum` function to return an `int option` instead:

```
fun sum(x,y) =
    let val s = x + y in
        if s < 100 then SOME s else NONE
    end;
```

Modify the definition of `sumlist` accordingly so that it uses the new version of `sum`. (Hint: modify the local function `slist` so that it returns an `int option` as well.)