Name:____

CSE341, Fall 2011, Midterm Examination
October 31, 2011

Please do not turn the page until the bell rings.

Rules:

- The exam is closed-book, closed-note, except for one side of one 8.5x11in piece of paper.
- **Please stop promptly at 3:20.**
- You can rip apart the pages, but please staple them back together before you leave.
- There are **100 points** total, distributed **unevenly** among **5** questions (all with multiple parts).
- When writing code, style matters, but don't worry much about indentation.

Advice:

- Read questions carefully. Understand a question before you start writing.
- Write down thoughts and intermediate steps so you can get partial credit.
- The questions are not necessarily in order of difficulty. **Skip around.** Make sure you get to all the problems.
- If you have questions, ask.
- Relax. You are here to learn.

Name:____

1. This problem uses this datatype binding, which describes "expression trees" where leaves are constants or variables and internal nodes are additions or multiplications.

```
datatype exp = Constant of int
             | Variable of string
             | Add of exp * exp
             | Multiply of exp * exp
```

(a) (**10** points) Write an ML function `has_variable` of type `exp * string -> bool` that returns `true` if and only if the string appears somewhere in the expression. You can use `=` to compare strings.

(b) (**10** points) Write an ML function `const_not_under_add` of type `exp -> bool` that returns `true` if and only if there exists at least one constant that is not "underneath" at least one addition. For example,
`Multiply(Add(Constant 3, Constant 4), Multiply(Variable "x", Constant 6))`
would produce `true` because the 6 is not under an addition, but
`Add(Multiply(Constant 1, Constant 2), Multiply(Constant 3, Constant 4))`
would produce `false` because all the constants are under an addition even though they are not "directly" under it.

Solution:

(a)
```
fun has_variable (e,s) =
    case e of
        Constant _ => false
      | Variable x => s=x
      | Add(e1,e2) => has_variable (e1,s) orelse has_variable (e2,s)
      | Multiply(e1,e2) => has_variable (e1,s) orelse has_variable (e2,s)
```

(b)
```
fun const_not_under_add e =
    case e of
        Constant _ => true
      | Variable _ => false
      | Add _ => false
      | Multiply(e1,e2) => const_not_under_add e1 orelse const_not_under_add e2
```

Name:____

2. This problem considers this ML code:

```
exception BadArgs

fun f (g,xs,ys) =
    case (xs,ys) of
        ([],[]) => true
      | (x::xs, y::ys) => f(g,xs,ys) andalso g(x,y)
      | _ => raise BadArgs
```

(a) (**12** points) For each of the following expressions that use `f`, fill in the blank with an argument that causes the overall expression to evaluate to `true`.

i. `f((fn (x,y) => x > y), [3,4,5], ____________)`

ii. `not (f((fn (x,y) => x > y), [3,4,5], ____________))`

iii. `((f ((fn (x,y) => x > y), [3,4,5], ____________)) ; false)  handle BadArgs => true`

(b) (**5** points) In English, briefly describe *what* `f` computes (not *how* it computes it).

(c) (**4** points) Is `f` tail-recursive? Explain briefly.

Solution:

(a) i. Any list `[v1,v2,v3]` of ints where `v1<3`, `v2<4`, and `v3<5`. For example, `[0,0,0]`.

ii. Any list `[v1,v2,v3]` of ints where `v1>=3`, `v2>=4`, or `v3>=5`. For example, `[2,5,4]`.

iii. Any list of ints that has a length other than 3. For example, `[2]`.

(b) It takes a function `g` that takes a pair and two lists `xs` and `ys`. It returns true if and only if for all i, `g` applied to the i^{th} elements of the two lists returns true. It raises an exception if the two lists have different lengths. (Note it applies `g` to the elements closer to the tail of this list first and uses short-circuiting to return false as soon as a call to `g` returns false, but if `g` has no side effects and always terminates, this is only an implementation detail. Solutions did not have to discuss this issue.)

(c) No. The recursive call to `f` is not the last work the caller needs to do (even though the caller's result may be the result of the recursive call). The caller needs to check the result and use it to determine whether `g(x,y)` is evaluated. This may be easier to see if we remember `e1 andalso e2` is sugar for `if e1 then e2 else false`. The recursive call here is `e1` and the first subexpression of a conditional is not in tail position. (Much shorter explanations are fine. Answers suggesting `g(x,y)` must be computed lost a small amount since it is only evaluated sometimes, but sometimes is enough to prevent tail recursion.)

Name:____

3. For each of the following programs, give the value that `ans` is bound to after evaluation.

   (a) (**5** points)

```
val x = 2
val y = 3
fun f z =
    let
        val y = x
        val x = y
    in
        x + y + z
    end
val z = 4
val ans = f x
```

   (b) (**6** points)

```
val x = 1
fun f y =
    if y > 2
    then fn z => x + z
    else fn z => x - z
val x = 3
val g = f 4
val x = 5
val ans = g 6
```

   (c) (**5** points)

```
fun f x =
    case x of
        [] => 0
      | (a,b)::[] => a + b
      | (a,b)::(c,d)::_ => a + d
val ans = f (List.map (fn x => (1,x)) [2,4,8,16,32])
```

Solution:

(a) 6

(b) 7

(c) 5

Name:____

4. (a) (**10** points) Without using any helper functions, write an ML function `filter_map`, which combines aspects of `List.filter` and `List.map`, as follows:

   - It takes two arguments *in curried form*: (1) a function `f` that takes list elements and produces options and (2) a list `xs`.
   - It returns a list.
   - If `v1` is a value in the input list and `f v1` returns `SOME v2`, then `v2` is in the output list. Notice `v2` is in the output list, ***not*** `SOME v2`.
   - Like `List.map`, it preserves the order of results: if `v1` precedes `v2` in the input, `f v1` is `SOME v3`, and `f v2` is `SOME v4`, then `v3` precedes `v4` in the output.
   - Like `List.filter`, the result list may be shorter than the input list: if `f` returns `NONE` for n elements, then the result will have n fewer elements.

   (b) (**4** points) What is the type of `filter_map`?

   (c) (**6** points) Use a `val` binding and `filter_map` to define `positive_lengths`, which should take a list of strings and return the lengths of all non-empty strings. For example, `positive_lengths ["", "hi", "currying", "", "", "341"]` evaluates to `[2,8,3]`. Use `String.size` as *part* of your solution.

   (d) (**2** points) What is the type of `positive_lengths`?

   (e) (**2** points) Here is an alternate implementation of `filter_map` if you fill in the blanks with the right ML library functions. Do so.

```
fun filter_map f = (___________ valOf) o (___________ isSome) o (___________ f)
```

Solution:

(a)
```
fun filter_map f xs =
    case xs of
      []    => []
    | x::xs => case f x of
                 NONE => filter_map f xs
               | SOME y => y :: filter_map f xs
```

(b) `('a -> 'b option) -> 'a list -> 'b list`

(c) (Using a let-binding not required)

```
val positive_lengths =
    filter_map (fn x =>
                  let
                      val s = String.size x
                  in
                      if s > 0 then SOME s else NONE
                  end)
```

(d) `string list -> int list`

(e) `fun filter_map f = (List.map valOf) o (List.filter isSome) o (List.map f)`

Name:____

5. ***This problem continues onto the next page and has a part (b).***

   Consider this structure definition:

```
structure NonEmptyStringList :> NESTRINGLIST =
struct
type t = string list
fun newList s = [s]
fun cons (s,ss) = s::ss
fun longest ss =
    case ss of
        [] => raise List.Empty
      | [s] => s
      | s::ss => if String.size s >= String.size(longest ss) then s else longest ss
end
```

   (a) (**16** points) For each of the **five** following definitions of `NESTRINGLIST`, decide which of the following is true for client code (code outside the module) and ***briefly*** *justify your choice*:

   - A: It can cause an exception by calling `NonEmptyStringList.longest` with an empty list.
   - B: It cannot call `NonEmptyStringList.longest` at all.
   - C: It can call `NonEmptyStringList.longest`, but not in a way that can cause an exception.

```
signature NESTRINGLIST =
sig
type t = string list
val newList : string -> t
val cons : string * t -> t
val longest : t -> string
end
```

```
signature NESTRINGLIST =
sig
type t = string list
val cons : string * t -> t
val longest : t -> string
end
```

```
signature NESTRINGLIST =
sig
type t = string list
val newList : string -> t
val cons : string * t -> t
end
```

Name:____

```
signature NESTRINGLIST =
sig
type t
val newList : string -> t
val cons : string * t -> t
val longest : t -> string
end

signature NESTRINGLIST =
sig
type t
val newList : string -> string list
val cons : string * t -> t
val longest : t -> string
end
```

(b) (**3** points) Even for the signature(s) where the answer is (C), why is `longest` a very poorly written function? Describe an argument that a client could create for which `longest` would perform very badly.

Solution:

(a) (Five signatures in order:)

- A: Because clients know `longest` takes a `string list`, they can call `NonEmptyStringList.longest []`
- A: Same reason as previous
- B: `longest` is not in the signature and there is no other way to call it
- C: Because `t` is abstract and this the argument to `longest`, the only way to get a `t` is via calls to `newList` and `cons`, and no such call would produce `[]`
- B: Clients do not know that `newList` can produce a `t`. The only function known to produce a `t` is `cons` but it also requires a `t` argument, so no `t` can ever be produced and therefore `longest` cannot be called.

(b) `longest` can take exponential time since it is possible for it to use recursion twice at each step for the tail of the list. The worst-case is when the longest string is not near the beginning of the list since running time is exponential in the number of elements that precede the longest string. Even under a signature where `t` is abstract, clients could build such a list via calls to `newList` and `cons`.

Name:____

This page intentionally blank.