Please do not turn the page until everyone is ready.

Rules:

- The exam is closed-book, closed-note, except for one side of one 8.5x11in piece of paper.

- Please stop promptly at 10:20.

- You can rip apart the pages, but please staple them back together before you leave.

- There are 65 points total, distributed unevenly among 5 questions (all with multiple parts).

- When writing code, style matters, but don’t worry 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.

- If you have questions, ask.

- Relax. You are here to learn.
1. This problem uses this datatype definition:

```plaintext
datatype my_string_list = Nothing | Something of string * my_string_list
```

(a) (4 points) Write a function `total_size` that computes the sum of the sizes of the strings in a `my_string_list`. Use the ML library function `String.size`, which computes a string’s size and has type `string->int`.

(b) (7 points) Consider this ML program:

```plaintext
exception Foo

fun f (lst,n) =
  if n<=0
  then Nothing
  else case lst of
      Nothing => raise Foo
    | Something(s,lst) => Something(s,f(lst,n-1))
```

Describe what `f` computes (not how it computes it). Be sure to cover all possible cases.

(c) (3 points) Suppose we modify `n<=0` to be `n=0` in `f`. Describe how the behavior of `f` does or does not change for all possible cases.

**Solution:**

(a) `fun total_size lst =
   case lst of
     Nothing => 0
   | Something(s,lst) => (String.size s) + (total_size lst)

(b) Given a `my_string_list` `lst` and a positive number `n`, `f` returns a `my_string_list` that contains the first `n` elements of `lst`. For example, `f(Something("x",Something("y",Nothing)), n)` evaluates to `Something("x",Nothing)` if `n` is 1 and `Nothing` if `n` is 0. If `n` is greater than the length of `lst` (i.e., the number of `Something` constructors in the value bound to `lst`), then `f` raises the exception `Foo`. If `f` is passed a non-positive number, it returns `Nothing`.

(c) The behavior is exactly the same as before except that it now raises an exception if passed a negative number. (Note it cannot go into an infinite loop because `lst` always has finite length.)
2. For each of the following programs, give the value that \texttt{ans} is bound to after evaluation.

(a) (4 points)
\begin{verbatim}
val x = 1
val f = fn x => fn y => x + y
val x = 2
val g = f x
val x = 3
val ans = g x
\end{verbatim}

(b) (4 points)
\begin{verbatim}
val x = 1
val f = fn y => y x
val x = 7
val g = fn y => x - y
val ans = f g
\end{verbatim}

(c) (4 points)
\begin{verbatim}
fun f x = List.map hd x
fun g x =
  case x of
    a::(b::c) => b
  | _ => 0
val ans = g (f [[1,2],[3,4],[5,6],[7,8]])
\end{verbatim}

Solution:

(a) 5
(b) 6
(c) 3
3. (a) (8 points) Write a function `majority` that takes a function `f` and a list `lst` and returns true if and only if `f` returns true for a strict majority of the list elements.
   - `majority` should take its argument in *curried form* with `f` first.
   - Write and use a helper function that returns an `int` (which might be positive or negative).
   - Do not use any ML library functions.

(b) (3 points) What is the type of `majority`?

(c) (3 points) Use a `val` binding and `majority` to define `mostly_positive`, which should take a list and return true if and only if a strict majority of its elements are strictly greater than 0.

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

Solution:

(a) fun majority f lst =  
    let fun vote lst =  
        case lst of  
          [] => 0  
        | hd::tl => (if f hd then 1 else ~1) + (vote tl)  
      in (vote lst) > 0 end  
(b) ('a -> bool) -> 'a list -> bool  
(c) val mostly_positive = majority (fn x => x > 0)  
(d) int list -> bool
4. Consider these two implementations of `fold` for ML lists. The first one is what we studied in lecture.

```ml
fun fold1 f acc lst =
  case lst of
    [] => acc
  | hd::tl => fold1 f (f(acc,hd)) tl

fun fold2 f acc lst =
  case lst of
    [] => acc
  | hd::tl => f(fold2 f acc tl, hd)
```

(a) (3 points) Which of the `fold` functions above is tail-recursive?

(b) (4 points) What does

```
fold1 (fn (acc,next) => if acc=next then 17 else acc+next) 0 [0,1]
```

evaluate to?

(c) (3 points) What does

```
fold2 (fn (acc,next) => if acc=next then 17 else acc+next) 0 [0,1]
```

evaluate to?

**Solution:**

(a) `fold1` is tail-recursive; `fold2` is not

(b) 18

(c) 1
5. Suppose version 1.0 of your software uses this ML structure definition:

```
structure M :> MSIG =
struct
  datatype age = Older | Younger
  datatype contact = Friend of age | Enemy of age
  fun makeFriend a = Friend a
  fun makeEnemy a = Enemy a
  fun isFriend c = case c of Friend _ => true | _ => false
  fun isOlder c = case c of Friend(Older) => true | Enemy(Older) => true | _ => false
end
```

Now suppose in version 2.0 of your software you want to replace the structure with this one:

```
structure M :> MSIG =
struct
  datatype age = Older | Younger
  datatype relation = Friend | Enemy
  type contact = age * relation
  ...
end
```

(a) (5 points) Provide 4 function bindings to complete version 2.0 of the structure so that it provides the same functionality as the version 1.0 structure.

(b) (5 points) Complete this signature such that both version 1.0 and version 2.0 of structure M would type-check. Use one abstract type definition and 4 val bindings.

```
signature MSIG =
  sig
    datatype age = Older | Younger
    ...
  end
```

(c) (3 points) Explain how version 1.0 of the structure could be made a few characters shorter by exploiting a notion of function equivalence we studied.
Solution:

(a)  
```ml
fun makeFriend a = (a,Friend)
fun makeEnemy a = (a,Enemy)
fun isFriend (_,r) = r = Friend (* pattern-matching solutions also fine *)
fun isOlder (a,_) = a = Older (* pattern-matching solutions also fine *)
```

(b)  
```ml
signature MSIG =
sig
  datatype age = Older | Younger
  type contact
  val makeFriend : age -> contact
  val makeEnemy : age -> contact
  val isFriend : contact -> bool
  val isOlder : contact -> bool
end
```

(c) The definitions of `makeFriend` and `makeEnemy` use unnecessary function wrapping. We could write:

```ml
val makeFriend = Friend
val makeEnemy = Enemy
```