1. (21 pts) OCaml Polymorphic Types

Consider an OCaml module Bst that implements a binary search tree:

```ocaml
module Bst = struct
  type bst =
    Empty
  | Node of int * bst * bst

  let empty = Empty (* empty binary search tree *)

  let is_empty = function (* return true for empty bst *)
    Empty -> true
  | Node (_, _, _) -> false

  let rec insert n = function (* insert n into binary search tree *)
    Empty -> Node (n, Empty, Empty)
  | Node (m, left, right) ->
    if m = n then Node (m, left, right)
    else if n < m then Node(m, (insert n left), right)
    else Node(m, left, (insert n right))

  (* Implement the following functions
     val min : bst -> int
     val remove : int -> bst -> bst
     val fold : ('a -> int -> 'a) -> 'a -> bst -> 'a
     val size : bst -> int
    *)

  let rec min = (* return smallest value in bst *)
  let rec remove n t = (* tree with n removed *)
  let rec fold f a t = (* apply f to nodes of t in inorder *)
  let size t = (* # of non-empty nodes in t *)
end
```

a. (3 pts) Is `insert` tail recursive? Explain why or why not.
b. (3 pts) Implement `min` as a tail-recursive function. Raise an exception for an empty bst. Any reasonable exception is fine.
c. (6 pts) Implement `remove`. The result should still be a binary search tree.
d. (6 pts) Implement `fold` as an inorder traversal of the tree so that the code
   ```ocaml
   List.rev (fold (fun a m -> m::a) [] t)
   ```
   will produce an (ordered) list of values in the binary search tree.
e. (3 pts) Implement `size` using `fold`. 

---

**CMSC330 Spring 2008 Homework #1**

Answer essay questions concisely using 1 sentence, or at most 2 sentences. Longer answers are not necessary and a penalty may be applied.
2. (36 pts) Recursive Descent Parser in OCaml
The example OCaml recursive descent parser 15-parseArith_fact.ml employs a number of shortcuts. For instance, the function `parseS` handles the grammar rules for $S \rightarrow T + S | T$
directly instead of first applying left factoring:

$S \rightarrow T A$
$A \rightarrow S | \epsilon$

However, we can still identify where code corresponding to `parseA` was inserted
directly in the code for `parseS`, in the comments below:

```ocaml
let rec parseS lr = (* parseS *)
    let x = parseT lr in (* S -> T A *)
    match !lr with (* parseA *)
        | ('+'::t) -> (* if lookahead = First( + S ) *)
            lr := t; (* A -> + S *)
            Sum (x,parseS lr)
        | _ -> x (* A -> \epsilon *)
```

Similarly, the function `parseF` handles the grammar rules for

$F \rightarrow F ! | U$
directly instead of rewriting the grammar, creating the following productions:

$F \rightarrow ?$
$B \rightarrow ?$

You must identify where code corresponding to `parseB` was inserted directly in the
code for `parseF` in the comments below:

```ocaml
let rec parseF lr = (* parseF *)
    let rec fHelper lr tmp =
        match !lr with (* parseB *)
            | ('!'::t) -> (* 1: if lookahead = First( ? ) *)
                lr := t; (* 2: ? -> ? *)
                Fact (fHelper lr tmp)
        in let x = parseU lr in (fHelper lr x) (* 4: ? -> ? *)
```

a. (3 pts) What rule should have been applied to the productions for $F$?
b. (6 pts) What productions for $F$ & $B$ would be created by applying the rule?
c. (3 pts) What sentential form should appear in place of $?$ in comment 1?
d. (3 pts) What production should appear in place of $?$ in comment 2?
e. (3 pts) What production should appear in place of $?$ in comment 3?
f. (3 pts) What production should appear in place of $?$ in comment 4?
3. (6 pts) Function arguments
   a. In the following code, identify each funarg and whether it is upward or downward.
      let f x = let g y = x + y in let app a b = a b in app g 1 ;;
   b. In the following code, identify each funarg and whether it is upward or downward.
      let f x = let g y = x + y in g ;;

4. (6 pts) Static vs. Dynamic Scoping
   Consider the following OCaml code.
   let a = 1 ;;
   let f = fun () -> a ;;
   let a = 2 ;;
   f ();
   a. What value is returned by the invocation of f() with static scoping? Explain.
   b. What value is returned by the invocation of f() with dynamic scoping? Explain.

5. (8 pts) Parameter passing
   Consider the following C code.
   int i = 2;
   void foo(int f, int g) {
     f = f-i;
     g = f;
   }
   int main() {
     int a[] = {2, 0, 1};
     foo(i, a[i]);
     printf("%d %d %d %d\n", i, a[0], a[1], a[2]);
   }
   a. (2 pts) Give the output if C uses call-by-value
   b. (3 pts) Give the output if C uses call-by-reference
   c. (3 pts) Give the output if C uses call-by-name
6. (10 pts) Polymorphism

Consider the following Java classes:

```java
class A { public void a() { … } }
class B extends A { public void b() { … } }
class C extends B { public void c() { … } }
```

Explain why the following code is or is not legal

a. `int count(Set<A> s) { … } … count(new TreeSet<A>( ));`
b. `int count(Set<A> s) { … } … count(new TreeSet<B>( ));`
c. `int count(Set s) { … } … count(new TreeSet<A>( ));`
d. `int count(Set<? s) { … } … count(new TreeSet<A>( ));`
e. `int count(Set<? extends A> s) { … } … count(new TreeSet<B>( ));`
f. `int count(Set<? extends B> s) { … } … count(new TreeSet<A>( ));`
g. `int count(Set<? extends B> s) { for (A x : s) x.a( ); }`
h. `int count(Set<? extends B> s) { for (C x : s) x.c( ); }`
i. `int count(Set<? super B> s) { for (A x : s) x.a( ); }`
j. `int count(Set<? super B> s) { for (C x : s) x.c( ); }`

7. (6 pts) Java multithreading

a. Using Java Conditions, you must implement a synchronization construct called MyBarrier. A MyBarrier object is created with a certain value n. When a thread calls the method enter( ), it enters the barrier and blocks until a total of n threads have entered the barrier. When the n<sup>th</sup> thread enters the barrier, all the threads waiting at the barrier wake up and unblock, and the n<sup>th</sup> thread continues without blocking. When a thread calls the method reset( ), the barrier is reset so that it starts fresh in counting up to n (i.e., n more threads must enter the MyBarrier). You may start by modifying the following code fragment:

```java
public class MyBarrier {
    public void MyBarrier (int n) { … }
    public enter( ) { … }
    public reset( ) { … }
}
```

8. (6 pts) Garbage collection

Consider the following Java code.

```java
Object a, b, c;
public foo( ) {
    a = new Object();  // object 1
    b = new Object();  // object 2
    c = new Object();  // object 3
    a = b;
    b = c;
    c = a;
}
```

a. (3 pts) What object(s) are garbage when foo( ) returns? Explain why.
b. (3 pts) Describe the difference between mark-and-sweep & stop-and-copy.
9. (4 pts) Markup languages
   a. Creating your own XML tags, write an XML document that organizes the following information: 1-hour test on Spanish Monday in Jiménez worth 15%. 1-hour test on Computers Tuesday in CSIC worth 10%. 30-minute test on Computers Friday in AVW worth 5%.

10. (8 pts) Lambda calculus
    Evaluate the following λ-expressions as much as possible
    a. \((\lambda z.z)\ (\lambda y.y)\ (\lambda x.x\ a)\)
    b. \((\lambda z.z)\ (\lambda z.z\ z)\ (\lambda z.z\ y)\)
    c. \((\lambda x.\lambda y.x\ y\ y)\ (\lambda a.a)\ b\)
    d. \((\lambda x.\lambda y. x\ y\ y)\ (\lambda y.y)\ y\)

11. (24 pts) Lambda calculus encodings
    Prove the following using the appropriate λ-calculus encodings
    a. \(\text{not}\ (\text{not}\ \text{true}) = \text{true}\)
    b. \(\text{if}\ \text{false}\ \text{then}\ x\ \text{else}\ y = y\)
    c. \(\text{succe}\ 2 = 3\)
    d. \(\text{(*}\ 1\ 3) = 3\)
    e. \(\text{(+}\ 2\ 1) = 3\)
    f. \((\text{Y}\ \text{fact})\ 2 = 2\) \-comment you do not need to expand any operators except fact & Y

12. (27 pts) Miscellaneous
    a. Describe the difference between OCaml modules and Java classes.
    b. Describe the difference between strong and weak typing.
    c. Explain how call-by-name simplifies implementing lazy evaluation.
    d. Describe the difference between an L-value and an R-value.
    e. Describe the difference between ad-hoc and parametric polymorphism.
    f. Describe the difference between starvation and deadlock.
    g. Describe how functional programming may be used to simulate OOP.
    h. Describe the difference between HTML and XML.