1. Evaluation Order

The following is Euclid’s algorithm for greatest common divisor assuming that \( a \geq b > 0 \), and \( \text{rem} \) is the remainder after integer division (so \( \text{rem} \ 9 \ 4 \) is 1).

\[
\text{(define (gcd a b)}
\text{(if (= b 0)}
\text{a)
\text{(gcd b (rem a b)))))
\]

Assume that \( \text{rem} \) raises an error if \( a < b \) or if \( b = 0 \).

(a) (4 pts) Using a substitution model, evaluate \( \text{gcd} \ 8 \ 6 \) using applicative-order (eager) evaluation.
   i. (3 pts) show the body of gcd just before evaluating the if on each recursive call.
   ii. (1 pt) How many times is \( \text{rem} \) called?

(b) (4 pts) Using a substitution model, evaluate \( \text{gcd} \ 8 \ 6 \) using normal-order (lazy without memoization) evaluation.
   i. (3 pts) show the body of gcd just before evaluating the if on each recursive call.
   ii. (1 pt) How many times is \( \text{rem} \) called?

(c) (2 pts) redefine \text{gcd} as follows:

\[
\text{(define (my-if prop cons alt)}
\text{(if prop cons alt))}
\]

\[
\text{(define (gcd a b)}
\text{(my-if (= b 0)}
\text{a)
\text{(gcd b (rem a b)))))
\]

what would the result of \( \text{gcd} \ 8 \ 6 \) be under
i. applicative-order evaluation?
ii. normal-order evaluation?
2. **Prolog programming**  
Given the Prolog definition of append (variables begin with Upper Case letters):

```prolog
append([], X, X).
append([Head | Tail], X, [Head | List] :- append(Tail, X, List).
```

(a) (5 pts) Show all results of the query

```prolog
?- append(X, Y, [1, [2, 3], [4]])
```

Show the results of the successive applications and the variable bindings produced by unification as the program executes.

(b) (5 pts) Write a Prolog program to copy a list: copy(OldList, NewList).  
Your program must fail if the input does not consist of a list. Does your program care in which parameter position (first or second) you pass the actual OldList? Explain. What would your program return if it were called as copy([1,2,3],[1|X]).? Explain.

3. **ML, Curried Functions**

(a) (6 pts) Write a (curried) function reduce that takes two parameters: a function \( F \) and a list of elements of suitable type and has the following inductive definition.

i. If \( n = 1 \), i.e., if the list has just one element \( a \), the result is \( a \).

ii. If \( n > 1 \), then let \( b \) be the result of reducing the tail of the list \([a_2, a_3, \ldots, a_n]\) by the function \( F \). Then the reduction of the whole list \([a_1, a_2, \ldots, a_n]\) by \( F \) is \( F(a_1, b) \).

(b) (4 pts) Write a function `uncurry2` that takes as input a function of type `'a -> ('b -> 'c)` and returns a function of type `(a * b) -> c`.

4. **Scheme and other stuff**

Consider the following function \( f \) defined on the natural numbers:

\[
\begin{align*}
f(0, y) &= 0 \\
f(1, y) &= y \\
f(x + 1, y) &= y + f(x, y)
\end{align*}
\]

(a) (2 pts) Write a Scheme function (procedure) `fll` that computes \( f \) in linear time and space.

(b) (2 pts) Write a Scheme function `flc` that computes \( f \) in linear time and constant space.

(c) (2 pts) Give a simple mathematical expression (closed form) for (the body of) \( f \).

(d) (2 pts) Prove that your answer to (c) is correct.

(e) (2 pts) Write a Scheme function `fcc` that computes \( f \) in constant time and space.
5. **Fixed Points**

Find the least fixed points of

(a) (3 pts) \( f = \lambda n.\text{if } n = 0 \text{ then } 1 \text{ else } f(n^2) \)

(b) (3 pts) \( f = \lambda n.\text{if } n = 0 \text{ then } 1 \text{ else } f(\text{floor}(n/2)) \)

The above is Stansifer’s notation. Other authors may use the notation:

(a) \( F = \lambda f.\lambda n.\text{if } n = 0 \text{ then } 1 \text{ else } f(n^2) \)

(b) \( F = \lambda f.\lambda n.\text{if } n = 0 \text{ then } 1 \text{ else } f(\text{floor}(n/2)) \)

The domain over which the functions \( f \) are (partially) defined is the non-negative integers. Make sure you show the construction of each fixed point.

(c) (2 pts) What makes the fixed point you constructed the “least fixed point”?

(d) (2 pts) What is another fixed point of (a)? What do you need to use as a first value in your fixed point construction to get that value?

6. **Logic programming**

(a) (1 pt) What is a clause?

(b) (2 pts) What is a Horn clause?

(c) (4 pts) Convert the following formula to prenex conjunctive normal form:

\[
\forall z. (P(z) \lor Q(z) \rightarrow (\forall x. \exists y. R(z, x) \land \neg S(y, z)))
\]

(d) (3 pts) Use Robinson’s resolution algorithm on the clauses

\[
P(a) \land \neg P(x) \

\neg P(y) \
P(f(a, z))
\]

where \( a \) is a constant, \( x, y, z \) are variables \( f \) is a function symbol, and \( P \) is a predicate symbol. Be explicit about unifications and renamings.

7. **ML Type Inference**

Given the Standard ML expression

```ml
let fun twice f = fn x => f (f x)
    fun id x = x
in
    ((twice id) "a", (twice id) 1)
end
```

(a) (2 pts) What is the most general type of the expression?

(b) (6 pts) Show how to derive the type of the expression. Show all the unifications involved.
(c) (2 pt) What if anything would change if we rewrote the let as a $\beta_v$-redex:

\[
((\text{fn twice} => \text{let fun id x = x} \\
\quad \text{in} \\
\quad \quad ((\text{twice id}) "a", (\text{twice id}) 1) \\
\quad \text{end}) \\
\quad (\text{fn f => fn x => f (f x))))
\]

8. Hoare logic

(a) (3 pts) If $P$ and $Q$ are logical formulae, and $C$ is a command, what is the meaning of the general form of the Hoare-logic (partial correctness) formula $\{P\} C \{Q\}$?

(b) (3 pts) What is the assignment rule for Hoare logic?

(c) (4 pts) Using assignment and sequencing rules of Hoare logic (plus any auxiliary rules that you can describe that may be needed), show:

$\{A=1 \land B=3\} \begin{array}{l}T := A; A := B; B := T \end{array} \{A=3 \land B = 1\}$

9. Calling conventions

To swap the contents of two locations, we often use a spare location:

\[
temp := a; \\
a := b; \\
b := temp;
\]

We can swap the contents of two locations, without using a spare location, using the exclusive or operation as follows:

\[
a := a \text{ xor} b; \\
b := a \text{ xor} b; \\
a := a \text{ xor} b;
\]

(a) (2 pts) Define call-by-reference (sometimes called pass-by-reference).

(b) (2 pts) define call-by-value-return (pass-by-value-return).

(c) (3 pts) Show how the following code can fail to swap for some inputs in a call-by-reference language:

\[
\text{void swap (int a, int b) \\
a := a \text{ xor} b; \\
b := a \text{ xor} b; \\
a := a \text{ xor} b; \\
return;}
\]

(d) (3 pts) Why does $\text{swap}$ always work for call-by-value-return?
10. **Subtype (inclusion) polymorphism**

(a) (2 pts) What does it mean for one type to be a subtype of another type?

(b) (4 pts) If we have two function types where the first is a subtype of the second:

\[(A \rightarrow B) \ll (C \rightarrow D)\]

what is the subtype relation between A and C? Between B and D?

(c) (4 pts) Consider the following Java-like code:

```java
1 class C
2    public int m1 (C c) {
3        D x;
4        ...
5        x = new E();
6    }
```

```java
class D {
    public int m2 (int x) { ... }
    public int m3 (int x) { ... }
    public int m4 (int x) { ... }
}
```

```java
class E extends D {
    public int m3 (int x) { ... }
    public int m5 (int x) { ... }
}
```

```java
class F extends D {
    public int m4 (int x) { ... }
}
```

If line 5 was method call `x.m?` where `?` is one of 1, 2, 3, 4, 5; what methods could be called, and in what class is each callable method defined (i.e. where is the text for the method written)?
11. Rewrite Rules

(a) (4 pts) What is the Church-Rosser property?
(b) (3 pts) Why is it important for $\lambda$-calculus?
(c) (3 pts) We can ask whether any set of rewrite rules has the Church-Rosser property. Consider the following set of rewrite rules for a calculus where the only terms are the symbols $A, B, C,$ and $D$.

\[
\begin{align*}
A & \rightarrow B \\
B & \rightarrow A \\
B & \rightarrow C \\
C & \rightarrow C \\
A & \rightarrow D \\
D & \rightarrow D
\end{align*}
\]

Does this calculus (set of symbols and rules) have the Church-Rosser property? Explain.

12. Syntax and Denotational Semantics

(a) (2 pts) What are the necessary properties of a denotational semantics?
(b) (2 pts) Write a BNF grammar for binary numerals (leading 0s are allowed) and fully parenthesized arithmetic expressions using the symbols “+” and “*”. One of the grammar rules should be

\[
digit ::= 0 \mid 1
\]

(c) (6 pts) Write down all definitions and functions that are necessary to give a denotational semantics for this language.