1. [20 pts] Consider that we extend IMP with a switch command:

```plaintext
switch (e) {
  case v1 : c1;
  ...
  case vn : cn;
  default : c;
}
```

Here, $e$ is an integer expression, $v_1, \ldots, v_n$ are distinct integer constants, and $c_1, \ldots, c_n, c$ are commands. If the value of $e$ matches some $v_k$, then command $c_k$ is executed; otherwise, the program executes $c$. The execution doesn’t fall-through from one case to the next case.

(a) [7 pts] Write small-step semantic rules to describe this execution.

(b) [7 pts] Write a Hoare-rule for the switch command.

(c) [6 pts] Given an abstract analysis domain $\text{Abs}$, write down the analysis of the switch command $C'[\text{switch} (e) \{\ldots\}]S$, where $S \in \text{Abs}$ is the abstract store before the statement. Assume that the domain $\text{Abs}$ does not hold enough information to statically determine whether two expressions are equal or different.

2. [18 pts] The following questions ask you to compare different language features.

(a) [6 pts] Give an advantage of static typing over dynamic typing, and an advantage of dynamic typing over static typing.

(b) [6 pts] Why do languages like Pascal or Modula-3 require static (or access) links, but languages like C or C++ don’t?

(c) [6 pts] In ML you can pass in functions as parameters and return functions. In Pascal or Modula-3, you can only pass in functions, but you can’t return them. How does this simplify things?

3. [7 pts] We want to translate pairs into options (sum types). To model options, we use the following ML datatype:

```plaintext
datatype S = L | R
```

Then, we derive a function $T[e]$ that translates each expression $e$ with pairs into a semantically equivalent expression with sums. The translation for the selection operator $\text{fst}$ is:

```plaintext
T[\text{fst } e] = T[e] (L)
```

Write an appropriate translation for pairs $T[(c_1, c_2)]$.

Note: your translation must preserve the call-by-value semantics for the evaluation of pairs. For instance, the evaluation of $T[\text{fst } (1, 2+3)]$ must evaluate $2+3$ at some point.
4. [12 pts] Consider the following ML function declaration:

\[
\text{fun f(x) = x(f(x))}
\]

(a) [6 pts] What is the type that ML infers for \( f \)?
(b) [6 pts] Consider the evaluation of expression \( f(fn \_ \Rightarrow 0) \). If this evaluation terminates, write the resulting value. If it doesn’t, explain why and mention a language where a similar expression would terminate.

5. [7 pts] Below is a program written in some language with nested procedures:

\[
\begin{align*}
\text{procedure A =} \\
& \begin{align*}
& \text{var x : integer;} \\
& \text{procedure B = begin print(x) end B;} \\
& \text{procedure C = ...} \\
& \text{begin x := 1; C(); end A;}
\end{align*}
\end{align*}
\]

Fill in the declaration of procedure \( C \) such that the program produces different outputs under static and dynamic scoping.

6. [7 pts] Write type-checking rules for the following ML-style exception constructs:

\[
e_1 \text{ handle } (x : t) \Rightarrow e_2
\]

\[
\text{raise } e
\]

Here, \( x \) is the formal argument of the exception handler \( e_2 \) (so \( x \) may occur free in \( e_2 \)), \( t \) is the type of \( x \), and \( e \) is the actual value being passed when the exception is raised. If an exception occurs, and is handled by \( \ldots \text{ handle } (x : t) \Rightarrow e_2 \)”, the result is the value of \( e_2 \). Your rules must be sound and least restrictive.

7. [7 pts] Suppose I declare the following classes in Java 1.5:

\[
\begin{align*}
\text{class A { int a; }} \\
\text{class B extends A { int b; }} \\
\text{class C<T> { ... }}
\end{align*}
\]

Then, I write the following:

\[
\begin{align*}
\text{C<B> ob = new C<B>();} \\
\text{C<A> oa = ob;}
\end{align*}
\]

Java will reject this program because it won’t type-check the second assignment. If you think Java is too conservative and it is okay to accept the program, write a sound subtyping rule that will allow this code to type-check. If you think it is unsafe to run this program, fill in the missing parts to show how this can lead to a run-time type error.

8. [8 pts] Consider two Java classes:

\[
\begin{align*}
\text{class A { } } & \text{ void foo(A a) \{ System.out.println("1"); \} } \\
& \text{ void foo(B b) \{ System.out.println("2"); \} }
\end{align*}
\]

\[
\begin{align*}
\text{class B extends A { } } & \text{ void foo(A a) \{ System.out.println("3"); \} } \\
& \text{ void foo(B b) \{ System.out.println("4"); \} }
\end{align*}
\]
and the following code fragment:

```java
A a = new B();
a.foo(a);
```

(a) [4 pts] What output does the above Java code produce?

(b) [4 pts] What would be produced if Java had multi-method dispatch?

9. [7 pts] Name three advantages that modules bring to software development, and briefly describe them in three sentences.

10. [7 pts] Consider the following Prolog program:

```prolog
f(X,Z) :- f(X, [], Z).
f([H|T], Y, Z) :- f(T, [H|Y], Z).
f([], Y, Y).
```

What is the result of the following query: 

\[
f([a, [b, c], R], [a, S, T])?
\]