MYSTERY grammar (for reference):

```
⟨Program⟩      ::=  ⟨Block⟩
                |   ⟨Block⟩ ';'
⟨Decls⟩        ::=  ⟨DeclList⟩ | ε
⟨DeclList⟩     ::=  ⟨Decl⟩
                |   ⟨Decl⟩ ';' ⟨DeclList⟩
⟨Decl⟩         ::=  'VAR' id ⟨OptType⟩
                |   'TYPE' id '=' ⟨Type⟩
                |   ⟨ProcDecl⟩
⟨OptType⟩      ::=  ε | ':' ⟨Type⟩
⟨ProcDecl⟩     ::=  'PROCEDURE' id '(' ⟨Formals⟩ ')' ⟨OptType⟩ '=' ⟨Block⟩
                |   'PROCEDURE' id '(' ⟨Formals⟩ ')' '=' ⟨Block⟩
⟨Formals⟩      ::=  ⟨FormalList⟩ | ε
⟨FormalList⟩   ::=  ⟨Formal⟩
                |   ⟨FormalList⟩ ',' ⟨Formal⟩
⟨Formal⟩       ::=  id ':' ⟨Type⟩
⟨Type⟩         ::=  'INTEGER'
                |   'UNIT'
                |   ⟨SubrTy⟩
                |   ⟨ArrayTy⟩
                |   id
                |   ⟨ProcTy⟩
⟨SubrTy⟩       ::=  '[' number 'TO' number ']'
⟨ArrayTy⟩      ::=  'ARRAY' ⟨SubrTy⟩ 'OF' ⟨Type⟩
⟨ProcTy⟩       ::=  'PROCEDURE' '(' ⟨Formals⟩ ')' ⟨OptType⟩
⟨Block⟩        ::=  ⟨Decls⟩ 'BEGIN' ⟨Stmts⟩ 'END'
⟨Stmts⟩        ::=  ⟨StmtList⟩ | ε
⟨StmtList⟩     ::=  ⟨Stmt⟩
                |   ⟨StmtList⟩ ';' ⟨Stmt⟩
⟨Stmt⟩         ::=  ⟨Assignment⟩
                |   ⟨Return⟩
                |   ⟨Block⟩
                |   ⟨Conditional⟩
                |   ⟨Iteration⟩
                |   ⟨Output⟩
                |   ⟨Expr⟩
⟨Assignment⟩   ::=  ⟨Expr⟩ ':=' ⟨Expr⟩
⟨Return⟩       ::=  'RETURN' ⟨Expr⟩
⟨Conditional⟩  ::=  'IF' ⟨Expr⟩ 'THEN' ⟨StmtList⟩ 'ELSE' ⟨StmtList⟩ 'END'
⟨Iteration⟩    ::=  'WHILE' ⟨Expr⟩'DO' ⟨StmtList⟩ 'END'
⟨Output⟩       ::=  'PRINT' ⟨Expr⟩
⟨Expr⟩         ::=  ⟨Operand⟩
                |   ⟨Expr⟩ ⟨Operator⟩ ⟨Operand⟩
⟨Operand⟩      ::=  number
                |   id
                |   ⟨Operand⟩ '[' ⟨Expr⟩ ']'
                |   ⟨Operand⟩ '(' ⟨Actuals⟩ ')'
                |   '(' ⟨Expr⟩ ')'
⟨Operator⟩     ::=  '+' | '>' | '==' | 'AND'
⟨Actuals⟩      ::=  ⟨ActualList⟩ | ε
⟨ActualList⟩   ::=  ⟨Expr⟩
                |   ⟨Actuals⟩ ',' ⟨Expr⟩
```

Question 1 (7 Points)

In the table below you see pairs of types with a box ☐ in between. Write an X in the box if neither type is a subtype of the other, or draw a $<:$ or $:>$ (suitably) to indicate that one is a subtype of the other.

Use the same assumptions as in class, i.e., that (1) we are using an imperative language (updates are allowed), that (2) the type system enforces strong typing and (3) the type system permits any type to be a subtype of another if and only if doing so will not require dynamic checks.

(a) (3 Points) Fill in as indicated above:

`[5 TO 10]`	☐	`[0 TO 10]`
`[5 TO 10]`	☐	`[3 TO 7]`
`[5 TO 10]`	☐	`[5 TO 7]`
`ARRAY [0 TO 10] OF [5 TO 10]`	☐	`ARRAY [0 TO 10] OF [0 TO 10]`
`ARRAY [0 TO 10] OF [0 TO 10]`	☐	`ARRAY [5 TO 10] OF [0 TO 10]`

(b) (4 Points) Continue filling in. For the following, assume that `A` is a *supertype* of `B`, and that the type `C[X]` is *covariant* in type parameter `X`.

`A`	☐	`B`
`C[A]`	☐	`C[B]`
`A` $\to$ `A`	☐	`A` $\to$ `B`
`A` $\to$ `A`	☐	`B` $\to$ `A`
`A` $\to$ `A`	☐	`B` $\to$ `B`
`A` $\to$ `B`	☐	`B` $\to$ `A`

Question 2 (9 Points)

Consider the following MYSTERY program. Assume that Mystery is configured so that the program can execute without any errors.

```
VAR z : INTEGER;
PROCEDURE P(y : INTEGER) : INTEGER =
  BEGIN
    y := y + 1;
    PRINT z;
    RETURN y + 2
  END;
PROCEDURE Q(x : INTEGER) : INTEGER =
  BEGIN
    PRINT x;
    PRINT x
  END
BEGIN
  z := 0;
  Q(P(z))
END
```

(a) (3 Points) What will the program print under *by-value-result* parameter passing? *Explain.*

(b) (3 Points) What will the program print under *by-reference* parameter passing? *Explain.*

(c) (3 Points) What will the program print under *by-name* parameter passing? *Explain.*

Personal Identifier:

Question 3 (12 Points)

Answer the following four questions about programming language concepts.

(a) (3 Points) What is *short-circuit evaluation*? Explain with an example.

(b) (3 Points) What is a *widening conversion*? Explain with an example.

(c) (3 Points) What is a the difference between *discriminated (tagged) union types* and *free (untagged) union types*? Explain with an example.

(d) (3 Points) What is a the difference between *type inference* and *dynamic typing*? Explain with an example.

Question 4 (8 Points)

Consider one of the following class interfaces (the two are equivalent; one is in Java, and one in Scala). The constructor definitions are omitted for brevity, as they play no role in this discussion.

```java
// Java
class C<X> {
  int    statusCode()  { ... };
  X      select(int z) { ... };
  C<X>   copy()        { ... };
}
```

```scala
// Scala
class C[X] {
  def statusCode(): Int  = ...
  def select(x : Int): X = ...
  def copy(): C[X]       = ...
}
```

Assume that we are defining such a class in a language with *definition-site variance*. We now try to determine the variance of type parameter **X**.

(a) (4 Points) Can we safely mark type variable **X** as *covariant*? Explain.

(b) (4 Points) Can we safely mark type variable **X** as *contravariant*? Explain.

Personal Identifier:

Question 5 (7 Points)

Consider the following program in MYSTERY. Assume that the program is well-formed and executable, and uses by-value parameter passing, *static storage binding* for global variables, *stack-dynamic storage binding* for all other variables, and *static scoping*:

```
1 VAR x : INTEGER;
2 PROCEDURE P(z : INTEGER) : INTEGER =
3   PROCEDURE Q(w : INTEGER) : INTEGER =
4     BEGIN
5       RETURN w + x
6     END;
7   PROCEDURE R(x : INTEGER, y : INTEGER) : INTEGER =
8     BEGIN
9       RETURN Q(z)
10    END
11  BEGIN
12    RETURN R(2, z)
13  END
14 BEGIN
15  x := 0;
16  PRINT P(1);
17  PRINT 0
18 END
```

(a) (4 Points) What is the scope of the variables listed below? List the number of all lines during whose execution the variable is in scope. You can use range notation (e.g., “5–12”).

x (line 1)	
z (line 2)	
y (line 7)	

(b) (3 Points) What is the difference between the *scope* and the *lifetime* of a variable? Use the code above as an example.

Personal Identifier:

Question 6 (7 Points)

Consider the language L0 whose syntax we define via the nonterminal ⟨*expr*⟩ in the following grammar:

$$
\begin{array}{lcl}
\langle \mathit{expr} \rangle & ::= & \mathit{nat} \\
& | & \langle \mathit{ltv} \rangle \\
& | & \langle \mathit{expr} \rangle \ \text{‘@’} \ \langle \mathit{expr} \rangle \\
& | & \langle \mathit{expr} \rangle \ \text{‘+’} \ \langle \mathit{expr} \rangle \\
& | & \text{‘[’} \ \langle \mathit{cont} \rangle \\
\langle \mathit{ltv} \rangle & ::= & \text{‘U’} \\
& | & \text{‘D’} \\
\langle \mathit{cont} \rangle & ::= & \langle \mathit{ltv} \rangle \ \langle \mathit{lock} \rangle \\
& | & \langle \mathit{expr} \rangle \ \text{‘,’} \ \langle \mathit{cont} \rangle \\
\langle \mathit{lock} \rangle & ::= & \text{‘L’} \ \langle \mathit{lock} \rangle \\
& | & \text{‘]’}
\end{array}
$$

where *nat* describes the natural numbers (ℕ).

(a) (4 Points) For each of the following token sequences, mark whether they are productions of the L0 grammar:

`[U]`	
`[ 1 , [ D L ]`	
`[ 1 , 3 , L L ]`	
`[ 3 , U L ]`	

(b) (3 Points) Assume that the operators ‘+’ and ‘@’ are left-associative, and that ‘@’ has a higher precedence than ‘+’. Draw the parse tree for the following expression: `1 + 2 @ 3 + 4`

Question 7 (12 Points)

Consider the following program in Java (on the left) or Scala (on the right); both programs are equivalent. Assume that we run this program once.

```java
1 class A {
2   void f()    { }
3   void g(A a) { a.f(); }
4   }
5
6 class B extends A {
7   @Override
8   void f()    { }
9 }
10
11 class C extends A {
12   @Override
13   void f() { h(); }
14   void h() { }
15 }
16
17 class D extends C {
18   @Override
19   void h() { }
20 }
21 A v = new B();
22 A z = new D();
23 v.f();
24 z.g(v);
25 v.g(z);
```

```scala
1 class A {
2   def f()       { }
3   def g(a : A) { a.f(); }
4 }
5
6 class B extends A {
7   override
8   def f() { }
9 }
10
11 class C extends A {
12   override
13   def f() { h(); }
14   def h() { }
15 }
16
17 class D extends C {
18   override
19   def h() { }
20 }
21 var v : A = new B()
22 var z : A = new D()
23 v.f();
24 z.g(v);
25 v.g(z);
```

(a) (3 Points) What static type(s) is variable `a` (line 3) bound to? *Explain.*

(b) (3 Points) What dynamic type(s) is variable `a` (line 3) bound to? *Explain.*

(c) (6 Points) What methods (e.g., `A.f`, `A.g`) will this program call, and in which order?

Question 8 (15 Points)

Consider the language defined below:

$$\begin{array}{lcl} \langle expr \rangle & ::= & num \\ & | & \langle pol \rangle \\ & | & \langle expr \rangle \text{ `*' } \langle expr \rangle \\ \langle pol \rangle & ::= & \text{`ID'} \mid \text{`ZERO'} \mid \text{`NEG'} \end{array}$$

where *num* describes the integers ($\mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\}$).

To define the type system and the natural semantics, we use the following metavariables:

n_1, n_2	Integer values (from *num*)
p_1, p_2	Productions of $\langle pol \rangle$
e_1, e_2	Productions of $\langle expr \rangle$
τ_1, τ_2	Types; must be either `Int` or `Pol`.

The **Type System** below assigns one of the two types `Int` or `Pol`:

$$\frac{}{n_1 : \texttt{Int}} (Tn) \qquad \frac{}{p_1 : \texttt{Pol}} (Tp) \qquad \frac{e_1 : \tau_1 \quad e_2 : \texttt{Int}}{e_1 \texttt{ * } e_2 : \texttt{Int}} (Tm1) \qquad \frac{e_1 : \texttt{Int} \quad e_2 : \tau_2}{e_1 \texttt{ * } e_2 : \texttt{Int}} (Tm2)$$

The **Natural Semantics** are:

$$\frac{}{n_1 \Downarrow n_1} (num) \qquad \frac{}{p_1 \Downarrow p_1} (pol) \qquad \frac{e_1 \Downarrow n_1 \quad e_2 \Downarrow n_2}{e_1 \texttt{ * } e_2 \Downarrow n_1 \cdot n_2} (mul)$$

$$\frac{e_1 \Downarrow \texttt{ID} \quad e_2 \Downarrow n_2}{e_1 \texttt{ * } e_2 \Downarrow n_2} (mpI) \qquad \frac{e_1 \Downarrow \texttt{ZERO} \quad e_2 \Downarrow n_2}{e_1 \texttt{ * } e_2 \Downarrow 0} (mpZ) \qquad \frac{e_1 \Downarrow \texttt{NEG} \quad e_2 \Downarrow n_2}{e_1 \texttt{ * } e_2 \Downarrow -n_2} (mpN)$$

where $n_1 \cdot n_2$ stands for arithmetic multiplication of n_1 and n_2.

(a) (4 Points) Are any parts of the language's semantics undefined? *Explain.*

(b) (3 Points) What does the expression '`NEG * 2 * 3`' evaluate to? Explain which rules you used to arrive at your conclusion.

(c) (6 Points) Extend the natural semantics such that the operator `*` can combine two `Pol` symbols, e.g., `ID * ZERO`. Ignore the type system for now. Your semantics should ensure that evaluating $(p_1 * p_2) * n_1$ gives the same result as evaluating $p_1 * (p_2 * n_1)$.

(d) (2 Points) Are any changes to the type system necessary to allow the operator `*` to combine two `Pol` symbols? If *yes*, describe the necessary changes on a high level. If *no*, explain your answer and name the applicable type rules.

Question 9 (12 Points)

Consider the following piece of Standard ML code:

```
datatype tree = N
              | B of tree   (* left child *)
                   * int    (* value *)
                   * tree   (* right child *)
```

This binary tree stores `int` values in each `B` node.

In the following, you do not have to get the syntax exactly right, as long as it is unambiguous what you are doing. Add explanations whenever you are in doubt about your syntax.

(a) (4 Points) Write a function `contains : (tree, int) -> bool` that checks whether a `tree` contains the `int` parameter that is passed in. Assume that the tree is sorted so that for any `B` node b, the left child of b only contains values less than the value stored in b and the right child of b only contains values greater than the value stored in b.

(b) (5 Points) Write a function `map : (int -> int) -> tree -> tree` that works analogously to the list `map` function that we discussed in class. In other words, a call to `map` with parameter `f : int -> int` and parameter `tr : tree` should replace all values v in `tr` by `f(`v`)`. Your function should otherwise leave the tree structure unchanged.

(c) (3 Points) Assume that you have `t : tree`. Call `map : (int -> int) -> tree -> tree` on `t` to increment all values in `t` by `2`.

Question 10 (8 Points)

Answer the following question about abstract datatypes.

(a) (4 Points) What is an *abstract datatype*? Explain with a short code example in a language of your choice. Make sure to state which language you are using.

(b) (4 Points) We discussed two concepts that programming languages use for supporting abstract datatypes in the type system: *subtyping* and *typeclasses*. Give one difference between these two concepts that affects *expressivity* or *reliability*.

Question 11 (3 Points)

Which of these following three kinds of storage location binding can give rise to the Dangling Pointer Problem in a language with manual (explicit) memory management? *Explain your answers.*

(a) (1 Point) Variables with static memory binding.

(b) (1 Point) Variables with stack-dynamic memory binding.

(c) (1 Point) Variables with explicit heap-dynamic memory binding.

Personal Identifier: