These questions are taken without modification from previous exams I have given. You may not be familiar with some of the languages mentioned, but you can expect the type of questions transfers quite easily to those you have seen.

1. C++ allows you to specify `const` reference parameters, e.g. `void foo(const Stack & s)`. These parameters are passed by reference but may not be modified.
   (a) (10 points) Why is this a useful feature for a language to have?
   (b) (10 points) Discuss how this feature can be type-checked, i.e. how can we make sure a constant reference parameter is not modified? Suggest the best algorithm you can along with possible problems it might have; make sure you indicate when this type-checking is done.

2. (10 points) For each of the following types, describe it in terms of the aggregate models of base types (cross products, mapping, sequences).
   (a) `ifstream X[10];`
   (b) `struct { int y[5]; ofstream os; };`

3. Consider the following grammar:
   
   ```
   <start> ::= <expression>
   <expression> ::= <item> 
                  | <expression> * <expression>
                  | <expression> ! <expression>
   <item> ::= a | b | c | d
   ```
   
   (a) (10 points) Prove that this grammar is ambiguous.
   (b) (5 points) Rewrite the grammar so that it is not ambiguous and gives ! a higher precedence than *.

4. (21 points) Consider the program below. For each of Name, Transitive Name, and Full Structural Equivalence, decide if the statements below are legal or illegal. (You could, for example, build a 3 x 7 table, one row per equivalence method, and list L for legal and I for illegal for each statement.)
Program ThreeTimesAType;

type
    T1 = array[1..10] of char;
    T2 = array[1..10] of T1;

var
    a,b : T1;
    c : array [1..10] of array[1..10] of char;
    d : T2;
    e : char;
    f : T2;

begin
    a := b; { statement 1 }
    d[2] := a; { statement 2 }
    c := d; { statement 3 }
    d := f; { statement 4 }
    c[2][1] := e; { statement 5 }
    c[5] := b; { statement 6 }
    f := c; { statement 7 }
end.

5. (15 points) Beginning programmer Suma Fuma produced the following code to sum the numbers between 1 and $n$.

#include <iostream>
void sum(int n)
{
    int sum = 0;
    for (int i = 0; i <= n; i++)
        sum = sum + i;
}
main()
{
    sum(10);
    cout << sum << endl;
“It doesn’t work!” Suma tells you.

The program compiles. Diagram the bindings applied to sum by indicating each point at which an instance of it is defined and drawing arrows to indicate which instance a reference to sum is bound to. Using the diagram, give an explanation of scope and why it causes Suma’s program to not work.

6. (15 points) Control flow Ruby provides iterators via the “each” method for any aggregate object, and counting methods for integers. Is there a need for any other iterative structure in the language?

7. (20 points) For each of the statements below, give one piece of evidence to support or discredit the claim.

   (a) ML is a simpler language than C++.
   (b) Ruby is more orthogonal than C++.
   (c) ML is more orthogonal than Ruby.
   (d) C++ permits more abstraction than ML.
   (e) Any language whose compiler or interpreter is written in C++ is just as portable as C++ itself.

8. (10 points) ML is a strongly-typed language and yet we can very easily write polymorphic functions like quicksort, which sorts any list of values for which we give a comparison predicate.

   (a) Explain what it means to be a strongly typed language.
   (b) Explain what polymorphism is and why this feature does not imply a contradiction to strong typing.