Question 1: (3 pts)
Write an example call to the OCaml function defined by:

```
let rec a b c = b (a b) c
```

Question 2: (5 pts)
Suppose tokenizers were generous instead of greedy. That is, suppose that, instead of taking the longest sequence of characters that would be a valid token, they take the shortest sequence. Explain what would go wrong. Use OCaml as an example.

Question 3: (10 pts)
In the Awk programming language, the expression ‘E1 E2’, where E1 and E2 are expressions, stands for string concatenation. For example, the Awk expression ‘"abc" "de"’ evaluates to the string "abcde". Suppose we extend Homework 2’s awkish_grammar to support string concatenation, by replacing ‘Binop -> [[T"+"]; [T"-"]]’ with ‘Binop -> [[]; [T"+"]; [T"-"]]’, so that the resulting grammar looks like this:

```
let ambigrammar =
  (Expr,
   function
     | Expr ->
         [[N Term; N Binop; N Expr];
          [N Term]]
     | Term ->
         [[N Num];
          [N Lvalue];
          [N Incrop; N Lvalue];
          [N Lvalue; N Incrop];
          [T"("; N Expr; T")"]]
     | Lvalue ->
         [[T"$"; N Expr]]
     | Incrop ->
         [[T"++"]; [T"--"]]
     | Binop ->
         [[]; [T"+"]; [T"-"]]
     | Num ->
         [[T"0"]; [T"1"]; [T"2"]; [T"3"]; [T"4"];
          [T"5"]; [T"6"]; [T"7"]; [T"8"]; [T"9"]])
```

Convert ambigrammar to ISO EBNF.

Question 4: (10 pts)
Prove that ambigrammar is ambiguous.

Question 5: (15 pts)
Suppose you have a working solution to Homework 2, and apply it to ambigrammar. What will your resulting parser do when given ambiguous input? Explain with an example.

Question 6: (15 pts)
Consider the DNA fragment analyzer in the hint code at the end of the old version of Homework 2. This code defines the type 'matcher' via 'type matcher = fragment -> acceptor -> fragment option'. Suppose we change the calling convention for matchers by having their argument being the acceptor, not the fragment, so that we define the type via 'type matcher = acceptor -> fragment -> fragment option'. Modify the hint code to use this new calling convention. Your modified version should be simple and elegant (as opposed to being as close to the original as possible).

Question 7: (10 pts)
Suppose we want to extend C++ to support Java-style generics, using a slightly different syntax when using generics (e.g., List<*String*> instead of List<String>). Conversely, suppose we also want to extend Java to support C++-style templates, using a slightly different syntax when using templates (e.g., List</String/> instead of List<String>).

Which of these two language extensions would be harder to implement and document, and why?

Question 8: (10 pts)
Does C++ support duck typing (as described in class)? Briefly explain why or why not

Question 9: (10 pts)
If a program contains an exit monitor operation A followed by a normal store B, the Java Memory model allows the thread's implementation to reorder operations so that it does B before A. However, the reverse is not true: if the program contains a normal store B followed by an exit monitor A, the JMM does not allow the thread's implementation to do A followed by B. Briefly explain why the former is correct but the latter is not.

Question 10: (12 pts)
The Java skeleton code in Homework 3's jmm.jar has a bug: its output line "Average swap time S ns real" overestimates the actual real average swap time, because the code assumes that each thread consumes real time equal to the difference between the last ending time of any thread and the first starting time of any thread. In theory the overestimate error could be large, as some threads could consume considerably more real time than others. Fix the bug by modifying the code to measure each thread's real time independently. Be careful to avoid race conditions in your fix.