Final Examination

9:00 – 12:00, Tuesday, August 2, 2016 Instructor: Alex Brodsky

Name: ____________________

Student Number: ____________________

Student Signature: ____________________

Duration: 180 minutes
Aids allowed: None

1. Place your student card on the table beside you, it will be checked during the exam.
2. This examination has 18 pages. Ensure that you have a complete paper.
3. The use of calculators, computers, books, papers, memoranda, cell phones, or any other electronic device is strictly prohibited.
4. Place your book-bags, coats, and books at the front of the room.
5. You may not reenter the examination once you leave.
6. You may not leave the examination in the last 15 minutes of the exam.
7. You must hand in the exam. You may not remove the exam from the room.
8. You may not ask questions of invigilators, except in cases of supposed errors or ambiguities in examination questions.
9. Write your answers in the space provided, if you need more space use the back of the **current** page and **indicate** this in the provided space.
10. **Your answers should be at most 3 sentences long!**
11. No smoking is permitted.
12. Write legibly and neatly.
13. Complete as much of the exam as you can.
14. Good Luck!

Question	Value
1	/30
2	/10
3	/20
4	/20
5	/15
6	/20
7	/20
8	/15
Total	/150

1. [30 Marks] Answer True or False for the following questions and **provide a brief (one or two sentence) justification:**

   (a) [3] ____________ A scanner generates a parse tree from stream of tokens.

   (b) [3] ____________ If L is a nonregular language then so is $\overline{L}$.

   (c) [3] ____________ Even if a grammar is LL(1) the language it specifies may still be inherently ambiguous.

   (d) [3] ____________ All local variables are bound at run-time.

   (e) [3] ____________ Tail recursion is not possible in any function that calls itself multiple times.

(f) [3] ______________ If DFAs M_1 and M_2 are different, then the languages $L(M_1)$ and $L(M_2)$ recognized by these DFAs must also be different.

(g) [3] ______________ If a grammar G is such that for any two productions with the same left-hand size, $A \to \alpha$ and $A \to \beta$ such that FIRST(α) is disjoint from FIRST(β), then the grammar is LL(1).

(h) [3] ______________ A synthetic attribute grammar can be used to enforce a rule requiring types to be declared before use.

(i) [3] ______________ A closure called with the same arguments, will always return the same value.

(j) [3] ______________ A generator can be implemented using a continuation.

2. [5 Marks] These questions deal with general programming languages concepts.

   (a) [5] For each of the following errors, state which phase of program translation would be responsible for catching the error. Assume all snippets of code are in Java.

   ```
   foo( 1 2 )
   ```

   ```
   3 = a :
   ```

   ```
   void foo() { return 3; }
   ```

   ```
   int i = 3K;
   ```

   ```
   if a == b return c;
   ```

   (b) [2] A compiler is typically separated into two parts: a front-end and a back-end. Why is this distinction important?

   (c) [3] Would it be easier to write an interpreter or a compiler for Scheme? Why?

3. [20 Marks] These questions deal with scanners and regular languages.

   (a) Give regular expressions for the following languages. You may use a dot (.) to denote any symbol in the alphabet and the notation $[a - b]$ to denote a range of symbols from an alphabet. E.g. $[a - d]$ is equivalent to $(a|b|c|d)$.

      i. [2] The language of all English words that do not have two consecutive vowels. Note: you may use $\mathcal{V}$ to denote the range of all vowels and $\mathcal{C}$ to denote the range of consonants.

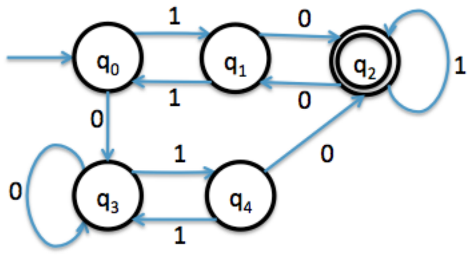
      ii. [2] The language of integers that are not divisible by 100. Note: integers may have leading 0's. I.e., 007 is a valid integer.

      iii. [2] The language, over the alphabet $\Sigma = \{X, Y, Z\}$, consisting of strings that have exactly three Xs or exactly three Ys, or exactly three Zs. E.g., $XYZXYZX$ and $ZZYZ$ are in the language, but $XYXYXYXY$ is not.

   (b) [5] Give a DFA that recognizes the language over the alphabet $\Sigma = \{a, b, c, d\}$ that consists of words where no two adjacent characters are the same. E.g., "abba" is not in the langauge, but "abcba" is.

(c) [3] Describe a deterministic approach to determine if two regular expressions specify the same language. I.e., you could write a program to do this.

(d) [6] Minimize the following DFA.



4. [20 marks] These questions deal with parsing and context free grammars.

   (a) Consider the grammar in Figure 1 (last page of this exam) and the following code.

```
struct outer {
  struct inner {
    int n;
  } s;
  int i;
  float h[42];
}
```

      i. [5] Using this grammar, derive a parse tree for the expression

      ii. [4] Suppose an LL(1) parser was parsing this snippet according to the grammar in Figure 1. Show how the parser's stack looks before and after it performs the first three derivations.

(b) [3] Describe the process for determining whether a grammar is LL(1).

(c) Consider the language $L = \{a^m b^n \mid m \neq n\}$.

i. [4] Give a context free grammar for this language. For part marks include a brief explanation of the grammar.

ii. [4] Give a DPDA for this language. Be sure to state method of acceptance. For part marks include a brief explanation of the DPDA.

5. [15 marks] These questions deal with semantic analysis and attribute grammars.

   (a) [2] What is the difference between synthetic and inherited attributes?

   (b) [3] What two things does an attribute grammar consist of and what is their purpose?

   (c) Consider the grammar in Figure 1 on the last page of this final. In C, all fields in a struct must have different names. That is, it is an error to have two fields with the same name in one struct. Suppose you had access to a *Set* data type with the following operations:

      `Set()` returns a new empty set.
      `Insert(S,i)` inserts item i into set S.
      `Contains(S,i)` return *true* if and only if item i is in set S.

   For the grammar in Figure 1 you will be asked to create an attribute grammar that uses the above Set abstract data type to detect duplicate fields.

      i. [3] Although either S-grammar or L-grammar can be used to to detect if duplicate fields, why would an L-grammar be more appropriate in this case?

ii. [7] For the grammar in Figure 1 give an attribute grammar that uses the above Set abstract data type to detect duplicate fields. (Hint: to save time and space use the number of each production instead of rewriting the production.)

6. [20 marks] These questions deal with naming and binding.

   (a) [3] Define the following terms:

   Binding :

   Scope of a binding :

   Reference environment :

   (b) [2] Why are objects that are allocated from the heap bound at runtime rather than compile time or load time?

   (c) [2] What is the difference between lexical and dynamic scoping?

   (d) [2] What is the difference between deep vs shallow binding?

(e) [2] What is a closure?

(f) [5] With the help of a diagram, describe how the reference environment for a closure can be implement so that the closure can be invoked from anywhere in the program, even outside of the scope where it was created.

(g) [4] The C language uses lexical scoping, does not allow nested functions, and allows static local variables. It can be argued that C supports closures. Justify this.

7. [20 marks] These questions deal with control flow and computation abstractions.

   (a) [2] What is the difference between an expression and a statement?

   (b) [4] Why are short-circuit and non-short-circuit evaluations of boolean expressions considered equivalent in a purely functional language, but are not considered equivalent in an imperative language?

   (c) [5] The Fibonacci sequence is defined as $1, 1, 2, 3, 5, 8, \ldots$ where the next number in the sequence is the sum of the previous two. Using pseudocode (or a language of your choice) write a generator that yields the next Fibonacci number in the sequence, starting with 1.

(d) Consider the following function, which computes the maximum of a list of numbers:

```
def max( L ):
  if len( L ) < 2:
    return L[0]
  else:
    m = max( L[1:] )  # call max() with rest of list
    if m < L[0]:
      m = L[0]
    return m
```

i. [1] Why is this **not** a tail-recursive function?

ii. [4] Rewrite this function (in pseudocode or language of your choice in a tail-recursive fashion. (You may use helper functions if needed.)

(e) [2] What happens when an exception is raised? Be specific.

(f) [4] Suppose you were using a language that had continuations but not exceptions. That is, the language had a `call-cc( f )` function that created a continuation c, called function f and passed c as a parameter to f. How could you implement exceptions using continuations? I.e., how would you implement the `guard` and `raise` mechanisms, like you did in your last assignment? (This is a hard question.)

8. [15 marks] These questions deal with types, data types and memory management.

   (a) [2] What is a type system?

   (b) [2] Why is type analysis typically done at the semantic analysis stage and not earlier?

   (c) [2] Most languages support some form of type coercion and casting. What is the difference between these two operations?

   (d) [2] Which of the two memory layout schemes for arrays, contiguous layout or row pointer layout, is more memory efficient? Why?

(e) [1] Why are dope vectors not needed if the language uses row-pointer layout for multidimensional arrays?

(f) [2] What is the fundamental problem with languages that require the programmer to manage their own memory?

(g) One problem with the reference counting approach for garbage collection is that circular references cannot be garbage collected, e.g., a linked list where the tail points to the head of the list.

i. [2] Why are circular references a problem for reference counting? .

ii. [2] Why does mark and sweep garbage collection not suffer from the same problem?

$$\begin{align}
\mathit{Type} &\rightarrow \mathit{Primitive} \tag{1}\\
\mathit{Type} &\rightarrow \mathit{Struct} \tag{2}\\
\mathit{Struct} &\rightarrow \text{`}\texttt{struct}\text{'}\ \texttt{id}\ \text{`}\{\text{'}\ \mathit{Field}\ \mathit{Fields}\ \text{`}\}\text{'} \tag{3}\\
\mathit{Fields} &\rightarrow \mathit{Field}\ \mathit{Fields} \tag{4}\\
\mathit{Fields} &\rightarrow \epsilon \tag{5}\\
\mathit{Field} &\rightarrow \mathit{Type}\ \texttt{id}\ \text{`;'} \tag{6}\\
\mathit{Field} &\rightarrow \mathit{Type}\ \texttt{id}\ \mathit{Array}\ \text{`;'} \tag{7}\\
\mathit{Array} &\rightarrow \epsilon \tag{8}\\
\mathit{Array} &\rightarrow \text{`[' }\texttt{int}\text{ `]'}\ \mathit{Array} \tag{9}\\
\mathit{Primitive} &\rightarrow \text{`}\texttt{int}\text{'} \tag{10}\\
\mathit{Primitive} &\rightarrow \text{`}\texttt{float}\text{'} \tag{11}\\
\mathit{Primitive} &\rightarrow \text{`}\texttt{char}\text{'} \tag{12}
\end{align}$$

Figure 1: A simple grammar for C types, with start symbol *Type*.