CSCE 531 Fall 2001
FINAL EXAM
Friday 01/12/14—Closed Book

The sum of all points is 170, but the maximum number of points attainable is 150.

1 Short Questions–1 point each; 14 points total

1. What is the difference between a translator and a compiler?
2. What is the name of the machine ("target machine") whose language is the target language of the compiler that we are building in CSCE 531?
3. What is the difference between the von Neumann architecture and the Harvard architecture?
4. The TAM has a Harvard architecture. True or false?
5. The TAM is a stack-based machine. True or false?
6. Why is it easier to write a compiler for a target machine with no registers?
7. What are the three components of the state in the denotational semantics approach[1]?
8. What is a cross-compiler?
9. BNF production rules may describe contextual languages. True or false?
10. BNF production rules and regular expressions have the same expressive power. True or false?
11. An emulator is an interpreter of source code. True or false?
12. McCarthy's `eval` is an example of a recursive interpreter. True or false?
13. What is the first argument of `eval`?
14. What is the second argument of `eval`?

[1] Every question on denotational semantics refers to the simple language described in class.

2 Semantics–5 points

1. (5 points) Describe (very briefly) the semantic difference between commands, expressions, and declarations.

3 BNF–6 points

1. (2 points) In Pascal, an identifier is a non-empty sequence of letters and numbers that starts with a letter. Provide BNF production(s) to describe Pascal identifiers.
2. (2 points) Provide a regular expression that describes Pascal identifiers.
3. (2 points) Provide a deterministic finite state automaton that recognizes Pascal identifiers.

4 Compilation–12 points

1. (2 points) List the three phases of compilation.
2. (2 points) What are the two kinds of constraints that are checked in contextual analysis?
3. (4 points) Briefly contrast static and dynamic scope rules on the following program:

```
int x = 4; // a global variable
main
 {
 foo  // a parameterless function
  {
  int x = 5;
  ...
  }
 bar  // another parameterless function
  {
  call foo;
  write x;  // which x?
  ...
  }
```

4. (2 points) Why is it impossible to write a one-pass compiler for Java?
5. (2 points) The argument you gave in answering the previous question does not hold for Pascal. Why?

5 Lexical Analysis–20 points

(Parts of this question are from `http://www.cs.purdue.edu/homes/hosking/502/`) Certain assemblers form their integer literals in the following way. *Binary* literals consist of one or more binary digits (`0, 1`) followed by the letter `B`; e.g., `10110B`. *Octal* literals consist of one or more octal digits `0` through `7` followed by the letter `Q` (since `O` looks too much like `0`; e.g., `1234567Q`). *Hexadecimal* literals consist of at least one decimal digit (`0` through `9`) followed by zero or more hexadecimal digits (`0` through `9` and `A` through `F`) followed by the letter `H`; e.g., `0ABCDEFH`. *Decimal* literals consist of at least one decimal digit *optionally* followed by the letter `D`; e.g., 1234.

1. (15 points) Draw the state diagram of an NFA (not a DFA) for these literal forms; you may use ϵ-transitions.

2. (5 points) Give a regular expression for the literals; you may use ϵ.

6 Contextual Analysis–13 points

1. (8 points) *Briefly* describe the difference between monolithic, flat, and nested block structures and explain how this affects management of the identification table.

2. (5 points) There are two common scope rules for the standard environment. (One is used in C.) Contrast them briefly.

7 Code Generation–90 points

1. (30 points) Some imperative programming languages (and most, if not all, functional languages) have a conditional *expression* construct. Do not confuse this with a conditional *statement* construct. For example, in C, the conditional expression is `<condition> ? <trueExpression> : <falseExpression>`. The conditional expression evaluates to either `<trueExpression>` or `<falseExpression>`, depending on the value of `<condition>`.

   There are two possibilities for the evaluation of this template.

   (a) The condition is evaluated before the expressions. Then, only the appropriate expression is evaluated. (This is an instance of *lazy* evaluation).

   (b) The condition and both expressions are evaluated at the same time. The value of the appropriate expression is then returned. This is an instance of *eager* (also known as *strict*) evaluation.

   Most (maybe all) programming languages implement the conditional expression using lazy evaluation. Discuss the problems of eager evaluation in this case. (Consider the possibility of side effects in expressions and expressions that give rise to errors.)

   Write a code template for Mini-Triangle for the conditional expression with lazy evaluation.

2. (20 points) Write a code template for Mini-Triangle for the *block expression*

   ```
   let D in E
   ```

   The declaration `D` is elaborated, and the resulting bindings are used in the evaluation of the expression `E`. The value of `E` is the value of the whole block expression.

3. (20 points) Write a code template for Mini-Triangle for the *mixed expression*

   ```
   begin C; yield E; end
   ```

   Here the command `C` is executed (including side effects), and then `E` is evaluated.

4. (20 points) Consider the following TAM assembly code.

```
0:  PUSH        1
1:  PUSH        1
2:  LOADL       0
3:  STORE (1)   0[SB]
4:  LOADL       1
5:  STORE (1)   1[SB]
6:  JUMP        15[CB]
7:  LOAD  (1)   0[SB]
8:  LOAD  (1)   1[SB]
9:  CALL        add
10:  STORE (1)   0[SB]
11:  LOAD  (1)   1[SB]
12:  LOADL       1
13:  CALL        add
14:  STORE (1)   1[SB]
15:  LOAD  (1)   1[SB]
16:  LOADL       5
17:  CALL        lt
18:  JUMPIF(1)   7[CB]
19:  LOAD  (1)   0[SB]
20:  CALL        putint
21:  POP   (0)   2
22:  HALT
```

The TAM assembly code is obtained by disassembling a Triangle program, parts of which are given below. Complete the program.

```
let
 var sum : Integer;
 var i : Integer
in
 begin
 ....
 putint(sum)
end
```

8 Interpretation—10 points

1. (2 points) Sketch the *iterative interpretation scheme*.
2. (2 points) List three kinds of languages for which iterative interpretation works well.
3. (6 points) Consider the iterative interpretation of Mini-Basic.
   (a) What are three reasonable choices for the representation of Mini-Basic commands in the code store?
   (b) Describe the trade-off involved in this choice of representation.