

**CS 370**  
**Final Exam**  
**December 11, 2006**

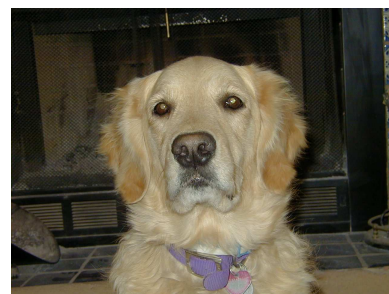
The following exam is open book and open notes. You may feel free to use whatever additional reference material you wish, but **no electronic aids** are allowed. Please note the following instructions. There will be a ten point deduction for failure to comply with them:

- start each problem on a new sheet of paper
- write your social security number, but not your name, on each sheet of paper you turn in
- show your work whenever appropriate. There can be no partial credit unless I see how answers were arrived
- be succinct. You may lose points for facts that, while true, are not relevant to the question at hand

You have until 10:00 to finish the exam.

1. (15 points) My dog Sandi is pretty well educated - she's passed the Advanced Obedience course at Karen's Kritters. Some of the verbal commands she knows are *heel*, *front*, *around*, and *side*

- Whenever I give her a verbal command, it starts with her name (*Sandi*). This is always followed by the command itself.
- When I give her the *heel* command (*i.e.* I say, "Sandi heel"), she walks beside me. When I stop walking, she sits facing forward (this requires no verbal command).
- While we're walking, I may give her the *front* command instead of just stopping. This command tells her to walk around in front of me, and sit facing me.
- After I give her the *front* command, I will always give her either the *side* command (in which case she'll go to my left, turn around, and sit facing forward waiting for a *heel* command) or the *around* command (in which case she'll walk around me to the right, and come up on my left facing forward, again ready for the *heel* command).



- (a) Write a regular expression describing this subset of the verbal commands that Sandi knows.
- (b) Draw a finite state machine for this language. Treat the words as symbols here (so you *don't* need to have a state transition for *S*, another for *a*, another for *n*, and so forth).
2. (10 points) We've spent quite a while using regular expressions to describe regular languages. Of course, the set of syntactically valid regular expressions is itself a language (notice that here we're talking about the regular expressions *themselves*, not the languages described by the regular expressions), described in the text on page 38. Is this language regular? Be sure to say why or why not.
3. (25 points) The programming language LISP is unique in never having really had its syntax "designed"; the intent was just to have an easy-to-translate notation for its internal data structures. As a result, its grammar is extremely simple – it just has two components:
- A symbol such as a number or an identifier is called an atom. An atom is a valid LISP expression. As a simplification, for purposes of this question, we'll assume the only valid atom is *a*.
  - A list of valid LISP expressions, surrounded by parentheses, is a valid LISP expression. The list may be empty.

So, for instance, *a*, (*a a a*), and ((*a a*) *a* (*a a* ())) are all valid LISP expressions.

Using { (, ), *a* } for your terminal symbols, write a context-free grammar for LISP.

4. (40 points) Suppose we wanted to add support for call-by-reference parameters to  $C^b$ .

- (a) How would identifier nodes in the syntax tree have to be modified to support this?
- (b) If a user program used a by-reference parameter in an expression, what code would have to be generated? For concreteness, assume the parameter is a reference to an `unsigned char`, and its offset in the activation record is 3. *Hint: you'll want to use the  $\Upsilon$  index register.*
- (c) If a user program made an assignment to a by-reference parameter, what code would have to be generated? Make the same assumptions as in part (b).

*Note: if you can't come up with specific HC11 code for parts (b) and (c), write a description of what has to happen for some partial credit.*

5. (25 points) Consider the following deterministic pushdown automaton:

$\Sigma = \{a, b, c\}$

$\Gamma = \{x\}$

$S = \{A, B, C\}$

$T = \{$

$(A, a, \epsilon) \Rightarrow (A, x)$

$(A, b, \epsilon) \Rightarrow (B, x)$

$(A, c, x) \Rightarrow (C, \epsilon)$

$(B, b, \epsilon) \Rightarrow (B, x)$

$(B, c, x) \Rightarrow (C, \epsilon)$

$(C, c, x) \Rightarrow (C, \epsilon)$

$\}$

$s_0 = A$

$A = \{A, B, C\}$

(a) Which of the following strings are accepted by this automaton?

i. `abcc`

ii. `acbc`

iii. `abbc`

(b) Give a one-sentence description of the strings accepted by the PDA.

6. (40 points) We never got around to implementing if-then-else statements this semester. Suppose we had. Remember that an if-then-else statement takes the form

```
if ( expr ) statement [else statement]
```

So, it's got some syntax that lets the parser recognize it (the keyword `if` and optional keyword `else`), an expression in parentheses, a statement, and an optional second statement. If the expression evaluates to non-zero the first statement is executed; if it evaluates to zero the second statement (if any) is executed.

- (a) What would a syntax tree node for an if-then-else have to contain? This includes both any fields containing data for the node itself, and any pointers it might need for children.
- (b) Sketch out how the node would be evaluated when generating code. What HC11 code would be produced by the expansion of the node itself, and what would we assume would be on the stack as a result of the code generated by the recursive calls handling the children of the node?

I don't really care how you give me your answers, just so it's clear what you mean. C code, pseudocode, and English (or any mix of the three) are equally acceptable.