CS 571
Midterm

Oct 24
100 points **Open book, open notes**
Time: 85 minutes **No Electronic Devices**

Important Reminder: As per the course Academic Honesty Statement, cheating of any kind will minimally result in receiving an F letter grade for the entire course.
Justify all answers
Please write your answers only within the provided exam booklets.

There are a total of 6 questions.

1. A Unix *path* is defined in stages as follows:

   Path Component A *path component* is a sequence of one-or-more characters which does not contain any occurrences of the `/` or `NUL` characters.

   Relative Path A *relative path* is a sequence of one-or-more path components separated by a single `/` character.

   Absolute Path An *absolute path* consists of the `/` character optionally followed by a relative path.

   Unix Path A Unix path is either an absolute or a relative path.

   Provide a regex for Unix paths. You may use `\/` to represent the regex matching `/` and `\0` to represent the regex matching the `NUL` character. You should factor (using intermediate named regex's) or format your answer to ensure that it is readable and understandable. *10-points*

2. An *X-expression* is either an atom, or two X-expressions surrounded by parentheses and separated by `.` (period), or a sequence of one-or-more X-expressions surrounded by parentheses.

   (a) Give a grammar for *X-expressions*. You should use the set of terminals { `ATOM`, `'('`, `')'`, `'.'`}.

   (b) Use your grammar to provide a *parse tree* for the X-expression `( (1 . 2) 3)`, where the integers will be scanned as `ATOM` terminals.

   20-points

3. Given the following program in a language which supports nested functions as well as both lexically-scoped (indicated using a `lex` declaration) and dynamically-scoped variables (indicated using a `dyn` declaration):

```
lex lex1 = 1;  //lexically scoped var lex1
dyn dyn1 = 2;  //dynamically scoped var dyn1

f(param_f) { //define function f with single parameter param_f
  lex lex1 = 3;
  dyn dyn1 = 4;

  g(param_g) { //define function g with single parameter param_g

    return lambda(x) { return x + param_f*param_g + lex1*dyn1; };
  }

  return g(dyn1);

}

print f(6)(7);
```

   What will be printed by the above program. Please remember to justify your answer. *20-points*

4. Describe how you would represent a CFG using basic S-expressions.

   (a) Specifically, describe how you would use S-expressions to represent *terminals*, *non-terminals*, *rules* and *grammars*.

   (b) Show your representation for the example CFG:

```
s : a
  | b
  ;
a : A
  ;
b : B
  ;
```

   (c) Describe how you would hide the details of your representation from users of your representation.

15-points

5. Write a Scheme function `(count-atoms` *s-exp*`)` which counts the number of atoms (non-pairs) in a maximally simplified representation of S-expression *s-exp* (i.e., the `'()` terminating proper lists should be ignored).

   Example log:

```
> (count-atoms 'a)
1
> (count-atoms '(a ()))
2
> (count-atoms '(a . ()))
1
> (count-atoms '(a b . ()))
2
> (count-atoms '(a b ()))
3
>
```

20-points

6. Discuss the validity of the following statements:

   (a) All evaluable Scheme expressions are S-expressions.

   (b) All S-expressions are evaluable Scheme expressions.

   (c) Assuming that a stack grows towards high memory, then within a stack frame for a function, the parameters to the function will be located at higher addresses than the local variables of the function.

   (d) Languages which allow recursive functions **must** use stack allocation for function parameters and local variables.

   (e) If a language requires a left-associative binary operator $\oplus$ to have the same precedence as a right-associative binary operator $\otimes$, then the language is ambiguous.

15-points