Name: ______________________

Midterm Exam
CS 314, Fall 2018
November 7
Section 01-03

DO NOT OPEN THE EXAM
UNTIL YOU ARE TOLD TO DO SO

Name: ______________________

Student ID: ________________

Section: __________

WRITE YOUR NAME ON EACH PAGE IN THE UPPER RIGHT CORNER.

Instructions

We have tried to provide enough information to allow you to answer each of the questions. If you need additional information. make a *reasonable* assumption, write down the assumption with your answer, and answer the question. There are **6** problems, and the exam has **9** pages. Make sure that you have all pages. The exam is worth **250** points. You have **1 hour and 20 minutes** to answer the questions. Good luck!

This table is for grading purposes only.

	Score	Points
1		70
2		45
3		55
4		20
5		30
6		30
Total		250

Name: ______________________

1 LL(1) Parsing (70) Points

Consider the language consisting of all strings of properly balanced parentheses. The context free grammar G is defined as follows:

1. `Goal ::= List`
2. `List ::= Pair List`
3. `List ::=` ϵ
4. `Pair ::= ( List )`

Note that `EOF` is the token that immediately follows the start symbol `Goal` and it means *end of file* (`EOF`), representing the sentence has reached its end. Therefore, `FOLLOW{Goal} = {EOF}`.

(a) Prove that the grammar is LL(1) using `PREDICT` sets. Please also give `FOLLOW` and `FIRST` sets and show the process of constructing `PREDICT` sets.

(b) Please provide the parse table below. **Insert the rule number or leave an entry empty**.

	(	)	`EOF`
Goal			
List			
Pair			

(c) Show the parse tree for `((())())`.

(d) Write a recursive descent parser using pseudo code. **Extra-credit Question (10 pts)**: Modify your recursive descent parser such that it outputs the maximum nesting level of a given expression. For example, for `(( )(( )))( )(())`, your program should output "Maximum nesting level: 3".

Name: ______________________

2 Regular Expression and Context Free Grammars (45 pts)

(a) Write an unambiguous context-free grammar that generates the same language as regular expression $ab*|c^+$. Describe the four components of your context-free grammar, including start symbol (S), terminals (T), non-terminals (NT), and the set of production rules (P).

(b) Using your grammar from part (a), give a right-most derivation of the string a b b b.

Name: ____________________

3 Name Binding and Scoping (55 pts)

Assume variable names written as **capital** letters use **dynamic** scoping and variable names written as **lower case** letters use **static** (lexical) scoping. Assume that procedures return when execution reaches their last statement. Assume that all procedure names are resolved using static (lexical) scoping.

```
program main()
{  int A, b;
   procedure f()
   {  int c;
      procedure g()
      {  int c;
         c = 35;
         ... = ...b...  // <<<--------- (*A*)
         print A,b,c;   // <<<--------- (*1*)
         end g;
      }
      print A,b;  // <<<--------- (*2*)
      A = 0; b = 0; c = 0;
      call g();
      print c;    // <<<--------- (*3*)
      end f;
   }
   procedure g()
   {  int A,b;
      A = 3; b = 8;
      call f();
      print A,b;  // <<<--------- (*4*)
      end g;
   }
   A = 1; b = 2;
   print A,b;     // <<<--------- (*5*)
   call g();
   print A,b;     // <<<--------- (*6*)
   end main;
}
```

(a) Show the output of the entire program execution. Label the output with the location of the print statement (e.g.: `(*2*): ...`).

(b) (Extra Credit 10pt) Show the program with all lexically scoped variable names (**lower case**) replaced by their `(level,offset)` representation. In this question, you **do** need to annotate procedure names to distinguish the two definitions of procedure `g()`.

Name: ______________________

(c) Show the runtime stack when execution reaches the point marked `(*2*)` in the code. Assume that program `main` has its own frame on the stack. Make sure you label all the stack frames with the corresponding program/procedure names and include the allocated stack variables (and their particular values) within the frame. Include all control links and access links between the activation records (stack frames), and value of the frame pointer `FP` by drawing an arrow to the corresponding location within the stack. **Use the frame layout in the figure below**.

parameter
return value
return address
access link
caller FP
local variables

(d) Give the RISC machine code for the non-local access to variable `b` at program point `(*A*)`. The access will need to load the content of variable `b` into a register. Use the RISC machine instructions `LOAD`, `STORE`, `LOADI`, `ADD`, `SUB` as described in the table below.

instr. format	description	semantics
memory instructions		
`LOADI` R_x `#<const>`	load constant value #<const> into register R_x	$R_x \leftarrow$ <const>
`LOAD` R_x `<id>`	load value of variable <id> into register R_x	$R_x \leftarrow$ <id>
`STORE` `<id>` R_x	store value of register R_x into variable <id>	<id> $\leftarrow R_x$
arithmetic instructions		
`ADD` R_x R_y R_z	add contents of registers R_y and R_z , and store result into register R_x	$R_x \leftarrow R_y + R_z$
`SUB` R_x R_y R_z	subtract contents of register R_z from register R_y , and store result into register R_x	$R_x \leftarrow R_y - R_z$
`MUL` R_x R_y R_z	multiply contents of registers R_y and R_z , and store result into register R_x	$R_x \leftarrow R_y * R_z$

4 Quick Identifications (20 pts)

Answer the following questions with either **yes** or **no** by marking the appropriate box as your selected answer (4 pts each).

(a) Every language that can be defined by a regular expression can also be expressed through a context-free grammar.

☐ Yes ☐ No

(b) It is easier to reason about a program's execution behavior if the program language uses dynamic scoping instead of static scoping.

☐ Yes ☐ No

(c) Garbage collection is a run-time technique to reduce the memory requirements of programs by detecting regions on the heap that can no longer be accessed by the program.

☐ Yes ☐ No

(d) Formal parameter is the information passed from caller to callee. Actual parameter is the local variable whose value is received from the caller.

☐ Yes ☐ No

(e) Regular expression and finite state automaton are equivalently powerful.

☐ Yes ☐ No

Name: ____________________

5 Compilers V.S. Interpreters. (30 pts)

To answer this question, please use the following definitions.

- A compiler maps an input program to a semantically equivalent output program. Note that the input and output language may be the same.
- An interpreter maps an input program to the answers it computes; In other words, it executes the program.

As part of the C project, we used and/or wrote the following programs/commands:

- `gcc` *usage*: `gcc <program>`
- `compile` *usage*: `compile <program>`
- `optimize` *usage*: `optimize <program>`
- `run` *usage*: `run <program>`

Under Unix/Linux, commands can be entered on a single command line if they are seperated by a semicolon. For instance, saying `cd foo; ls` will change the current directory to subdirectory `foo`, and will list its files. In addition, output to standard-out from one program can be "piped" into the standard-in of another program. This is called a Unix/Linux pipe, with the two programs seperated by `|` (e.g.: `prog1 | prog2`).

In the project, we used several languages, namely C, tinyL, RISC machine code, and ilab machines object code (executables). Classify the following commands or sequence of commands as a single unit, i.e., as a single composed command. If the single or composed command is a compiler, give its input and output language (e.g.: input language: C, output language: tinyL). For an interpreter, just give its input language.

(a) `compile test1`

Answer (mark one): ☐ compiler ☐ interpreter

Input language: ________, Output language: ________

(b) `compile test1; optimize < tinyL.out | optimize > opt.out`

Answer (mark one): ☐ compiler ☐ interpreter

Input language: ________, Output language: ________

(c) `compile test1; run tinyL.out`

Answer (mark one): ☐ compiler ☐ interpreter

Input language: ________, Output language: ________

(d) `gcc Interpreter.c`

Answer (mark one): ☐ compiler ☐ interpreter

Input language: ________, Output language: ________

6 Parameter Passing (30 pts)

```
program MAIN {
   int a := 1;
   int b := 2;
   int c := 5;
   procedure X(int z) {
      z := z + a;
      a := 2 * b * c;
   }
   /* MAIN */
   call X(b);
   call X(c);
   print(a, b, c);
}
```

Assume we use lexical scoping, given the program above, what values does the program print? Please fill in the values of variables a and b assuming different parameter passing styles. Recall that in *pass value-result* mode, the actual parameter supplied by the caller is first copied into the callee's formal parameter, and after the function completes, the (potentially modified) formal parameter of the callee is then copied back to the caller's actual parameter.

Pass by value	Pass by reference	Pass by value-result
a =	a =	a =
b =	b =	b =
c =	c =	c =

Please justify your answer for the output of each parameter passing mode.