RYERSON UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

CPS 710

FINAL EXAM

FALL 2016

NAME: _____

STUDENT ID: _____

INSTRUCTIONS

Please answer directly on this exam.

This exam has 4 questions, and is worth 40% of the course mark. It has 8 pages including this one

NO AIDS ARE ALLOWED.

Throughout this exam, non-terminals in grammars are in UPPER-CASE and terminals are in a **shaded box**.

A - General Concepts	30
B - Parsing	20
C - Grammars	25
D - Evaluation	25

Part A - General Concepts - 30 marks

A1 (2 marks)

What is a **self-hosting** compiler or interpreter?

A2 (3 marks)

What is a **virtual machine** and why is it useful?

A3 (8 marks)

What would be the output of the following program fragment:

```
integer i = 1;
integer function f(integer n)
   {
   i += n;
   return i;
   }
integer function g(integer n)
   {
   integer i = n;
   return f(n);
   }
display g(2);
display f(1);
```

If the language is statically scoped?	If the language is dynamically scoped?

A4 (4 marks)

List at least 4 errors that can only be detected once the symbol table is implemented

A5 (13 marks)

As you know, Java integer literals are mostly strings of 1 or more digits. However, there is more to them:

1. Underscores ("`_`") can be embedded in integer literal strings as long as they are not the first or last digit of the integer literal string. (These underscores are just placeholders to make it easier to read integers with many digits)
2. Integers can be represented in four bases, which is indicated by the prefix, as follows:
   - Binary integers start with the string "`0b`". The only digits allowed are "`0`" and "`1`".
   - Hexadecimal integers start with the string "`0x`". The digits allowed are "`0`" to "`9`" and "`A`" to "`F`"
   - Octal integers start with the string "`0`". The digits allowed are "`0`" to "`7`".
   - Decimal integers start with a digit other than "`0`"`.` The digits allowed are "`0`" to "`9`".
3. Finally, the fact that an integer is a Long is indicated by adding the suffix "`L`" to the string

   So for example" `0b100_000_101L`" is the binary number "`100000101`" stored as a Long.

Write a regular expression that can be used to scan any integer literal. You can use regular definitions if you want. You can also use the standard regular expression notational shorthands: "+", "?", and [a-z] for character classes.

Part B - Parsing (20 marks)

B1 First and Follow (11 marks)

Fill out the tables calculating the first and follow sets based on the following LL(1) grammar for structs in the C language:

				First (RHS of production)
(1)	S	→	STRUCT STRUCTS `$`	
(2)	STRUCT	→	`struct` `id` `{` MEMBERS `}` `;`	
(3)	STRUCTS	→	STRUCT STRUCTS	
(4)	STRUCTS	→	ε	
(5)	MEMBERS	→	MEMBER MEMBERS	
(6)	MEMBERS	→	ε	
(7)	MEMBER	→	`int` `id` `;`	
(8)	MEMBER	→	`float` `id` `;`	

	S	STRUCT	STRUCTS	MEMBERS	MEMBER
First					
Follow					

B2 LL(1) table (9 Marks)

Fill in the LL(1) table for this grammar, by entering the number of the production that is applied (e.g. (3)) in each cell where a production is applied.

	`struct`	`id`	`{`	`}`	`;`	`int`	`float`	`$`
S								
STRUCT								
STRUCTS								
MEMBERS								
MEMBER								

Part C - Grammars - 25 marks

C1 (6 marks)

Left-factor the following set of productions fully. You can introduce new non-terminals if you wish.

DEFINITION → FUNCTIONDEF | VARIABLEDEFS

FUNCTIONDEF → `type` `id` `(` PARAMETERS `)`

VARIABLEDEFS → `type` `id` (`,` `id`)*

VARIABLEDEFS → `type` `id` `[` DIMENSIONS `]`

C2 (7 points)

Remove the **left recursion** from the following set of productions. You can assume that the two logical operations are left-associative and have the same precedence.

S → S `∧` S

S → S `∨` S

S → `true` | `false`

C3 (6 marks)

In this question `integer` refers to an integer token and not the word "integer".

Why is the following set of productions **ambiguous**?

TERM → `integer` | POLYN | TERM (`^` `integer`)?

POLYN → `integer` `X` (`^` `integer`)?

C4 (6 marks)

Rewrite the grammar in C3 to remove the ambiguity while describing the same language.

Part D - Evaluation - 25 points

Preliminary Explanation

In this question you will be evaluating vector comparisons in a programming language for vectors. In this language vectors are ordered lists of integers and vectors. The elements of a vector do not have to all be the same type. For example, `[[10,3], 2, -1, [[3],-4], 5]]` is a vector

The section of the grammar which deals with vector comparisons is:

COMPARISON → EXPRVEC **=** EXPRVEC
EXPRVEC → **[** EXPRLIST **]**
EXPRLIST → EXPRESSION (**,** EXPRLIST)*
EXPRLIST → ε

You have written a parser for this language using javacc and jjtree. This parser produces the following types of AST nodes to deal with comparison expressions:

- **ASTcomparison** has two ASTexprvec children
- **ASTexprvec** has n ASTexpression children, where n ≥ 0. (There is no ASTexprlist – the parser optimized the data structures to put the list of expressions directly into the ASTexprvec.).
- **ASTexpression** can be treated like a black box in this assignment. It represents combinations of operations on vectors and integers.
- **ASTinteger** stores integer literals (constants). It has the following special method:

  public int getvalue() //returns the integer value of the literal

- **ASTvector** stores vector literals. An ASTvector has 0 or more children. Each child is either an ASTinteger or an ASTvector.

You have written an evaluator visitor for the entire language except for the ASTcomparison node. In particular these visitors are already written:

- public Object visit(ASTexpression node, Object data)
  // returns either an ASTinteger or an ASTvector
- public Object visit(ASTvector node, Object data)
  // returns node (because no further evaluation is necessary)
- public Object visit(ASTinteger node, Object data)
  // returns node (because no further evaluation is necessary)

Exam Question

On the next page, write this visitor method in Java:

- public Object visit(ASTcomparison node, Object data)
  // Returns Boolean.TRUE if the two children of node evaluate to the same value
  // Returns Boolean.FALSE otherwise

Other requirements:

- **Your evaluator should be as efficient as possible, short-circuiting the evaluation when possible: i.e. expressions should only be evaluated if absolutely necessary.**
- You can assume that the nodes you are evaluating are error-free. In particular, you do not need to perform any type checking or catch any exceptions.

Useful AST node methods:

- public int getId()
  // Returns the ID of the node (i.e. a number representing its type)
  // The following two IDs will be useful:
  //HLTreeConstants.JJTINTEGER and HLTreeConstants.JJTVECTOR
- int jjtGetNumChildren();
  // Return the number of children the node has.
- public Node jjtGetChild(int i);
  // Returns the ith child node. The children are numbered from zero, left to right.

STUDENT ID: _____

```
public Object visit(ASTcomparison node, Object data)
        // Returns Boolean.TRUE if the two children of node evaluate to the same value
        // Returns Boolean.FALSE otherwise
        {
```