RYERSON UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

CPS 710

FINAL EXAM

FALL 2006

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.

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

Part A - General Concepts - 30 marks

A1 (4 marks)

What is the primary difference between a compiler and an interpreter?

A2 (4 marks)

Why would you design a compiler to generate intermediate code instead of directly generating machine code?

A3 (6 marks)

Explain the difference between parse trees and syntax trees. Use an example to support your explanation.

A4 (5 marks)

Give the definition of a **rightmost** derivation. Use an example to support your definition.

A5 (5 marks)

List 5 different types of semantic errors in programs.

A6 (6 marks)

What would be the output of the following program fragment:

```
integer a = 1;
integer b = 2;
function f( integer a)
    {
    b := a + 5;
    return b;
    }
function g(integer a)
    {
    integer b;
    return f(a);
    }
display g(6);
display b;
```

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

Part B - Parsing (20 marks)

In this question, non-terminals are in UPPER-CASE and terminals are **shaded.**

Given the following LL(1) grammar:

(1) S → CASE **$**
(2) CASE → **case** **id** CLAUSES FINAL **end**
(3) CLAUSES → CLAUSE CLAUSES
(4) CLAUSES → ε
(5) CLAUSE → **const** **:** **id** **=** **const**
(6) FINAL → **default** **:** **id** **=** **const**
(7) FINAL → ε

B1 First and Follow (11 marks)

Calculate the first and follow sets of the following strings of terminals and non-terminals:

Production :	(1)	(2)	(3)	(4)	(5)	(6)	(7)
First (RHS of production)							

	S	CASE	CLAUSES	CLAUSE	FINAL
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.

	case	**id**	**end**	**const**	**default**	**:**	**=**	**$**
S								
CASE								
CLAUSES								
CLAUSE								
FINAL								

Part C - Evaluation - 25 points

Preliminary Explanation

In this question, non-terminals are in UPPER-CASE and terminals are **shaded.**

In this question you will be evaluating case expression in a new programming language.

The section of the grammar which deals with case expressions is:

CASE	→	**case** **(** EXPRESSION **)** COMPARISONS OTHERWISE **;**
COMPARISONS	→	{ COMPARISON **,** }*
COMPARISON	→	EXPRESSION **:** EXPRESSION
OTHERWISE	→	**otherwise** **:** EXPRESSION

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

ASTcase has 3 children: ASTexpression, ASTcomparisons, and ASTexpression:

1. The first ASTexpression child contains the *switch expression* whose value, called the *switch value*, will be compared with comparison values to decide which comparison clause applies.
2. ASTcomparisons is a list of comparisons (see below).
3. The last ASTexpression child contains the *otherwise expression* parsed in the otherwise clause. The value of this expression, called the *otherwise value*, will be the value of the entire case expression is the switch value is not equal to any of the comparison values.

ASTcomparisons has one ASTcomparison child for each comparison in the case expression.

ASTcomparison has 2 ASTexpression children:

1. The *comparison expression* whose value, called the *comparison value*, will be compared to the switch value.
2. The *result expression* whose value, called the *result value*, will be the value of the entire case expression if the switch value is the same as the comparison value.

Exam Question

You have written an evaluator visitor for the entire language except for the 3 visit method for ASTcase, ASTcomparisons, and ASTcomparison node.

On the next page, write these 3 visitor methods in Java:

- public Object visit(ASTcase node, Object data)
- public Object visit(ASTcomparisons node, Object data)
- public Object visit(ASTcomparison node, Object data)

Other requirements:

- Your evaluator should be as efficient as possible: expressions should only be evaluated if absolutely necessary.
- You can assume that the case expression you are evaluating is error-free. In particular, you do not need to perform any type checking or catch any exceptions.

Useful AST node methods:

- int jjtGetNumChildren();
  Return the number of children the node has.
- public Node jjtGetChild(int i);
  This method returns a child node. The children are numbered from zero, left to right.
- public Object jjtAccept(Visitor visitor, Object data)
  This method accepts the visitor and returns the evaluated value.

Part D - Grammars - 25 marks

In this question, non-terminals are in UPPER-CASE and terminals are **shaded.**

D1 (6 marks)

Left-factor the following set of productions fully. You may need to introduce new non-terminals.

B → C | **t** | **f**

C → (**t** | **f**) ∨ C

C → (**t** | **f**) ∧ C

D2 (7 points)

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

S → S ∩ S

S → S ∪ S

S → ∅ | **{1}** | **{2}**

D3 (6 marks)

What is an **ambiguous** grammar? Why would you not want a grammar to be ambiguous?

D4 (6 marks)

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

FN_CALL → **id** **(** EXPRESSIONS **)**

EXPRESSIONS → EXPRESSION EXPRESSIONS | ε

EXPRESSION → **id** | FN_CALL | **(** EXPRESSION EXPRESSION **)**