RYERSON UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

CPS 710

FINAL EXAM

FALL 2007

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	25
B - Parsing	25
C - Evaluation	25
D - Grammars	25

Part A - General Concepts - 25 marks

A1 (4 marks)

Explain the difference between a **native compiler** and **cross-compiler**.

A2 (3 marks)

Generally speaking, compilers and interpreters are faster if they have **fewer passes**. Is this more important for compilers or for interpreters? Explain your answer.

A3 (6 marks)

Give examples of the following types of errors (1 different example for each cell in the table)

Type of error:	Example
Lexical error	
Syntax error	
Syntax error	
Semantic error	
Semantic error	
Run-time error	

A4 (2 marks)

What is the **activation record** of a function call in a compiled run-time environment?
(Note that this question is continued in the next question)

A5 (4 marks)

List 4 types of entries that must be found in an activation record.

A6 (6 marks)

When discussing programming languages and compilers and interpreters, the two characterizations, "**static**" and "**dynamic**", are applied to various concepts: scoping, typing, error detection. Explain what these two words, "**static**" and "**dynamic**," mean in the context of compilers and interpreter theory.

Part B - Parsing (25 marks)

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

B1 First and Follow (14 marks)

Fill out the tables calculating the first and follow sets based on the following LL(1) grammar:

				First (RHS of production)
(1)	S	→	POLYN **$**	
(2)	POLYN	→	MONOM POLYNTAIL	
(3)	POLYNTAIL	→	+ MONOM POLYNTAIL	
(4)	POLYNTAIL	→	**-** MONOM POLYNTAIL	
(5)	POLYNTAIL	→	ε	
(6)	MONOM	→	COEFF **x** EXP	
(7)	COEFF	→	**const**	
(8)	COEFF	→	ε	
(9)	EXP	→	^ const	
(10)	EXP	→	ε	

Non-terminal	S	POLYN	POLYNTAIL	MONOM	COEFF	EXP
First						
Follow						

B2 LL(1) table (11 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.

	+	**-**	**const**	**x**	^	**$**
S						
POLYN						
POLYNTAIL						
MONOM						
COEFF						
EXP						

Part C - Evaluation - 25 points

Preliminary Explanation

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

In this question you will be writing evaluation visitors for polynomial evaluation expressions in a programming language that works with polynomials. This language is more complex than the one you have been using in class as it allows other names than "x" to be polynomial variables.

The section of the grammar which deals with polynomial evaluation is:

EVAL	→	POLYN [ASSIGNMENTS]
ASSIGNMENTS	→	ASSIGNMENT { , ASSIGNMENT }*
ASSIGNMENT	→	**identifier** = 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 polynomial evaluation:

ASTeval has 2 children:
- The first child is an ASTpolyn which contains the polynomial that will be evaluated
- The second child is an ASTassignments which corresponds to the list of variable assignments for that polynomial.

ASTpolyn is structured as follows:
- The first child is an Identifier containing the name of that polynomial's variable.
- The other children describe the terms of the polynomial.

ASTassignments has 1 or more children, each of which is an ASTassignment.
ASTassignment has 2 children:
- The first child is an Identifier containing a variable name.
- The second child is the AST of an expression which should evaluate to a Number.

The rules for the evaluation of a polynomial evaluation are as follows:
- If none of the variables in the assignments are the same as the polynomial variable, then the result of the polynomial evaluation is the original ASTpolyn structure.
  e.g. $(3x^2-5x)$[y=1+5, z=3-2] evaluates to $3x^2-5x$ since none of the assignment variables are x
- If at least one of the variables in the assignments is the same as the polynomial variable, then the result of the polynomial evaluation is a Number calculated by replacing the variable in the polynomial by the **last** value of that variable in the assignments.
  e.g. $(3x^2-5x)$[y=1/10, x=2, z=3, x=1*1, k=5] evaluates to -2 because the last value of x is 1;

Exam Question

You have written an evaluator visitor for the entire language except for the 3 visit method for ASTeval, ASTassignments, and ASTassignment nodes.

The evaluator visitor for ASTpolyn, which is already implemented, works as follows:
- public Object visit(ASTpolyn node, Object data)
  // If data is null, the visit returns node
  // If data is a Number value, visit returns a Number calculated by replacing the variable in the polynomial by data.

On the next page, write these 3 visitor methods in Java:
- public Object visit(ASTeval node, Object data)
- public Object visit(ASTassignments node, Object data)
- public Object visit(ASTassignment 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 expressions 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:

- 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.

Other useful methods and operators

- You can assume that the Identifier class has the following method:
  boolean equals(Identifier id)
  // This method returns true if and only if and only if id and this are the same identifier
- Don't forget that Java has an **instanceof** operator which checks whether an object is an instance of a class.

Part D - Grammars - 25 marks

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

D1 (6 marks)

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

S	→	FNCALL | ASSIGN
FNCALL	→	**identifier** **(** E **)**
ASSIGN	→	LHS **=** E
LHS	→	**identifier** { **[** E **]** }^{0}

D2 (7 points)

Remove the **left recursion** from the following set of productions. Do not worry about associativity and precedence.

C	→	C **or** N	|	C **and** N	|	N
N	→	**not** **(** C **)**	|	**true**	|	**false**

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**?

SUM	$\rightarrow$	TERM { (+ | -) TERM }*
TERM	$\rightarrow$	POLYN | **integer**
POLYN	$\rightarrow$	MONOM { (+ | -) MONOM }*
MONOM	$\rightarrow$	**integer** **X** ^ **integer**