RYERSON POLYTECHNIC UNIVERSITY

DEPARTMENT OF MATH, PHYSICS, AND COMPUTER SCIENCE

CPS 710

FINAL EXAM

FALL 96

STUDENT ID: ____

INSTRUCTIONS

Please write your student ID on this page. Do not write it or your name on any other pages.

Please answer directly on this exam.

This exam has 4 questions, and is worth 30% of the course mark.

NO AIDS ARE ALLOWED.

A - General Concepts	40
B - Shift-Reduce parsing	15
C - Evaluation	20
D - Grammars	25

This exam has 7 pages including the front page

This exam has 7 pages including the front page

Part A - General Concepts - 40 marks

<u>A1 (4 marks)</u>

What is a scanner?

<u>A3 (5 marks)</u>

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

<u>A3 (5 marks)</u>

Draw a syntax tree for the expression **p and q or not r**

<u>A4 (6 marks)</u>

Given the grammar
BOOLEAN $\rightarrow$ AND-EXPR {**or** AND_EXPR}^{0}
AND_EXPR $\rightarrow$ NOT_EXPR {**and** NOT_EXPR}^{0}
NOT_EXPR $\rightarrow$ {**not**}^{0} EXPR
EXPR $\rightarrow$ **p** | **q** | **r**

Give the rightmost derivation of the expression **p and q or not r**

A5 (6 marks)

Explain the difference between syntax and semantic errors. Give 2 examples of each type of error.

A6 (6 marks)

Why is is better to organise the name table of a symbol table as self-organising list (i.e. list of identifiers where last used identifier is moved to the front) rather than as a simple list?

A7 (10 marks)

Explain 2 different ways of organising parsing and symbol table activities in order to implement lexical scoping in an interpretor.

Part B - Shift-Reduce Parsing (15 marks)

Given the LR(1) grammar with the following productions augmented with states (shown subscripted and outlined):

(1) IF $\rightarrow {}_{1}$ **if** ${}_{2}$ **e** ${}_{3}$ STATEMENTS ${}_{6}$ **else** ${}_{7}$ STATEMENTS ${}_{9}$

(2) IF $\rightarrow {}_{1}$ **if** ${}_{2}$ **e** ${}_{3}$ STATEMENTS ${}_{6}$

(3) STATEMENTS $\rightarrow {}_{3,7}$ STATEMENTS ${}_{6}$ STATEMENT ${}_{8}$

(4) STATEMENTS $\rightarrow {}_{3,7}$ STATEMENT ${}_{5}$

(5) STATEMENT $\rightarrow {}_{3,6,7}$ **s** ${}_{4}$

Fill in the LR table for this grammar:

	IF	STATEMENTS	STATEMENT	**if**	**e**	**else**	**s**	**$**
1								
2								
3								
4								
5								
6								
7								
8								
9								

There are 4 possible entries for each slot in the table:

- S state: shift token onto symbol stack and change state
- R production: reduce the production by popping n symbols off the symbol stack and n symbols off the state stack, and inserting the left-hand side of the production into the input stream
- HALT: to halt process
- nothing: to indicate a syntax error

Part C - Evaluation - 20 marks - This has a short answer

In this question you will be evaluating CASE statements in a new programming language.

- The section of the grammar which deals with CASE statements is:

```
CASE            -> case EXPRESSION  COMPARISONS  OTHERWISE ;
COMPARISONS     -> {COMPARISON COMPARISONS}^0
COMPARISON      -> EXPRESSION : EXPRESSION
OTHERWISE       -> otherwise : EXPRESSION
```

  The non-terminals COMPARISON and COMPARISONS do not appear in any other part of the grammar.

- A syntax tree node produced by the parser has the structure:

```
typedef struct node {
    int type;
    struct node *v1, *v2, *v3;
    } node;
```

- a CASE structure is organised as follows:
  - type = CASE
  - v1 points to an expression which will be compared against others
  - v2 points to a COMPARISON, or an expression corresponding to the otherwise clause of the case statement.
- a COMPARISON structure is organised as follows
  - type = COMPARISON
  - v1 points to an expression that will be compared to the value of v1 in the CASE structure.
  - v2 points to an expression that will be the value of the case structure if the CASE v1 is equal to the COMPARISON v1.
  - v3 points to the next COMPARISON structure, or an expression corresponding to the otherwise clause of the CASE statement.
- The CASE structure evaluates to the value of the first v2 in the list of comparisons for which the associated v1 is equal to the v1 in the CASE structure. If there is no such value, it evaluates to the value of the otherwise clause.
- You are writing int eval(node *tree), a function which evaluates a syntax tree. Eval returns an int which is the value of the syntax tree (you can assume that syntax trees all evaluate to integers). Eval is completely written except for the part which handles CASE and COMPARISON structures.
- Write C code for the section of eval which handles CASE structures, and whatever C code is necessary to handle COMPARISON structures.
- You do not need to do any memory management in your code. You can assume that the CASE expression you are evaluating is error-free.

Part D - Grammars - 25 marks

In this question, non-terminals are in UPPER-CASE and terminals are <u>underlined</u>**.**

<u>D1 (6 marks)</u>

What is a **left-recursive** grammar? Why would you not want a grammar to be left-recursive?

<u>D2 (6 marks)</u>

Remove the **left recursion** from the following set of productions. You may need to introduce new non-terminals.

$S \rightarrow S \ \underline{\cap} \ S$

$S \rightarrow S \ \underline{\cup} \ S$

$S \rightarrow \underline{-} \ S$

$S \rightarrow \underline{\emptyset} \quad | \quad \underline{\{} \ \underline{1} \ \underline{\}} \quad | \quad \underline{\{} \ \underline{2} \ \underline{\}}$

D3 (6 marks)

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

S → IF | o

IF → IF1 | IF2

IF1 → if c then S

IF2 → if c then S else S

D4 (7 marks)

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

WHILE → while e do STATS od

STATS → STAT STATS^0

STAT → WHILE | ASSIGNMENT | EXPR

ASSIGNMENT → a := EXPR

EXPR → TERM {(+ | −) EXPR}^{0}

TERM → (+ | −)^0 a