RYERSON POLYTECHNICAL UNIVERSITY

DEPARTMENT OF MATH, PHYSICS, AND COMPUTER SCIENCE

CPS 710

FINAL EXAM

FALL 93

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.

CALCULATORS ARE ALLOWED. NO OTHER AIDS ARE ALLOWED.

Question A	25
Question B	25
Question C	25
Question D	25

This exam has 5 pages including the front page

This exam has 5 pages including the front page

Part A - General Concepts - 25 points

A1 (7 points)

Draw a transition diagram to recognise the regular expression: **a (a | b)* b**
Show all **a** and **b** edges from all states of your diagram.

A2 (8 points)

Draw a parse tree (NOT a syntax tree!) for the expression **a - b + c / d** given the grammar

DIFF -> {DIFF **-**}^{0} SUM
SUM -> QUOT {**+** SUM}^{0}
QUOT -> {QUOT /}^{0} LETTER
LETTER -> **a** | **b** | **c** | **d**

A3 (5 points)

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

A4 (5 points)

List 5 different types of semantic errors in programs

Part B - Internal Tables - 25 points

B1 (6 points)

Describe the interactions between the scanner and the symbol table manager

B2 (6 points)

Which part of the symbol table should be optimised for speed, and why?

B3 (7 points)

If you were writing a compiler for a language for which the average number of identifiers in a typical program is 75, how would you organise your name table? Why?

B4 (6 points)

List 3 stack frame fields

Part C - Evaluation - 25 points - This is easy

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. 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 points

In this question, non-terminals are in upper-case, and terminals in **lower-case bold**

D1 (6 points)

Left-factor the following set of productions fully
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:
S -> S ∩ S
S -> S ∪ S
S -> ∅ | **{1}** | **{2}**

D3 (6 points)

What is an **ambiguous** grammar?

D4 (6 points)

Why is the following set of productions **ambiguous**?
FN_CALL -> **id (** EXPRESSIONS **)**
EXPRESSIONS -> EXPRESSION EXPRESSIONS | **ε**
EXPRESSION -> **id** | FN_CALL | **(** EXPRESSION + EXPRESSION **)**