OLLSCOIL NA hÉIREANN
THE NATIONAL UNIVERSITY OF IRELAND, CORK

COLÁISTE NA hOLLSCOILE, CORCAIGH
UNIVERSITY COLLEGE, CORK

Summer Examinations 2014

CS4150: Principles of Compilation

Professor I. Gent
Professor B. O'Sullivan
Dr K. T. Herley

Answer All Questions
Total marks 100%

1.5 Hours
About 1.1 percent per minute

PLEASE DO NOT TURN THIS PAGE UNTIL INSTRUCTED TO DO SO

ENSURE THAT YOU HAVE THE CORRECT EXAM PAPER

Question 1 *[30%]*

(i) Give both a regular expression and a deterministic finite automaton that captures all alphanumeric strings (*i.e.* consisting of letters and digits) that contain the string "cs4150" as a substring. (You may make use of the following shorthand: $\cdot$ (dot) for "any symbol" and $[\wedge x]$ for "any symbol other than x".) (*10%*)

(ii) Draw a deterministic finite automaton that captures the strings over the alphabet $\{a, b, c\}$ in the following set: All strings containing both an even number of as and an even number of bs. (*5%*)

(iii) Give a regular expression that captures the set indicated in Part (ii). Justify your answer. (For partial credit, give instead a regular expression over the same alphabet that contains an even number of as.) (*15%*)

Question 2 *[25%]*

(i) We often use context-free grammars (CFGs) as a framework within which to describe the syntax of some notation $\mathcal{N}$ such as a programming language. This question involves the use of a CFG to capture the syntax of a variant of the context-free grammar notation itself. The CFG variant is illustrated below.

⟨term⟩ ⇒ ⟨term⟩ ***** ⟨factor⟩ ‖ ⟨factor⟩ •
⟨factor⟩ ⇒ **id** ‖ **num** •

Notice that we have substituted symbols ⇒ and ‖ for the more customary symbols → and | and that we have terminated the right-hand side of each production with the • symbol.

Write a context-free grammar $\mathcal{G}$ that captures the syntax of such grammars. Assume each grammar has at least one production. Use the symbols **N** and **T** to denote the nonterminals and terminals of the grammar respectively (in the same way that the symbol **id** denotes any validly formatted identifier in an expression grammar of the kind featured in Question 3 below). (*20%*)

(ii) Give a complete parse tree based on your grammar $\mathcal{G}$ for the term-factor grammar fragment given above. (*5%*)

Question 3 *[30 %]*

(i) Consider the following context-free grammar for a simple programming language named PL0. The operators **div** and **mod** are division operators that correspond to the Java operators / and % and := is used to indicate assignment. Modify the grammar to remove any left recursion. (*7%*)

⟨*prog*⟩ → ⟨*list*⟩
⟨*list*⟩ → ⟨*assignment*⟩ **;** ⟨*list*⟩ | ϵ
⟨*assignment*⟩ → **id :=** ⟨*expr*⟩
⟨*expr*⟩ → ⟨*expr*⟩ **+** ⟨*term*⟩ | ⟨*expr*⟩ **-** ⟨*term*⟩ | ⟨*term*⟩
⟨*term*⟩ → ⟨*term*⟩ ***** ⟨*factor*⟩ | ⟨*term*⟩ **div** ⟨*factor*⟩ | ⟨*term*⟩ **mod** ⟨*factor*⟩ | ⟨*factor*⟩
⟨*factor*⟩ → **(** ⟨*expr*⟩ **)** | **id** | **num**

(ii) Analyze the grammar to calculate the FIRST sets for each of the nonterminals in the (modified) grammar. (*8%*)

(iii) Give a succinct description of a recursive descent parser for PL0. Give pseudocode for the parsing methods for nonterminals ⟨*prog*⟩, ⟨*list*⟩ and ⟨*assignment*⟩. Give a crisp summary (in words) of the behaviour of the parsing methods for ⟨*expr*⟩, ⟨*term*⟩ and ⟨*factor*⟩ (and any derivatives of same stemming from modifications made in Part (i) above) and summarise carefully any assumptions regarding the behaviour of the scanner or any ancillary methods you rely upon. (*15%*)

Question 4 *[15 %]*

(i) Suppose the we wished to augment the PL0 language, introduced in Question 3 above, with a for loop with the following syntax. (The template on the left illustrates the general syntax, the code snippet on the right is an example that sums the numbers from 1 to 100.)

```
for var := initval to finalval step incr do
    begin
    ⟨list⟩
    end
```

```
sum := 0;
for k := 1 to 100 step 1 do
    begin
    sum := sum + k
    end
```

The intention is that the variable *var* takes on the values $initval, initval + incr, initval + 2 \times incr \cdots$ up to but not exceeding *finalval* and that the loop body (delimited by **begin** and **end**) is executed in turn for each such value. Note also that *initval*, *finalval* and *incr* may be expressions ($\langle expr \rangle$). Give a translation of the for loop shown on the right into three-address code (TAC), showing clearly the correspondence between the PL0 code and the TAC. *(10%)*

(ii) Write a brief note on the role played by the runtime stack in respect of memory management and in managing method call and return during program execution. *(5%)*

Three-Address Code

x := y op z	assignment
x := op y	unary assignment
x := y	copy
goto L	unconditional jump
if x relop y goto L	conditional jump
param x	procedure call
call p n	procedure call
return y	procedure call
x := y[i]	indexed assignment
x[i] := y	indexed assignment