UNIVERSITY OF EDINBURGH

FACULTY OF SCIENCE AND ENGINEERING

COMPUTER SCIENCE THREE

COMPILING TECHNIQUES

Wednesday 18 September 2002

09:30 to 11:00

Examiners:

Prof. K. Turner (External)

Prof. C. Stirling (Chairman)

INSTRUCTIONS TO CANDIDATES

Answer any TWO questions.

All questions carry equal weight.

1. (a) Explain the concept of states in lexer generation tools such as JLex. Give an example of a useful application of states. *[3 marks]*

   (b) Extend the following JLex lexer definition so that arithmetic expressions with multi-digit integer numbers, the operators "+", "-", "*", and "/" as well as parentheses ("(" and ")") can be scanned. Return appropriate tokens and values for recognised input symbols. Skip white spaces and issue error messages for illegal characters. *[10 marks]*

```
%%

%class Scanner

%function nextToken
%type java_cup.runtime.Symbol

%char
%line
%ignorecase

%{
   private java_cup.runtime.Symbol tok(int kind,Object value)
   {
      return new java_cup.runtime.Symbol(kind,yychar,
                                       (yychar+yylength()),value);
   }
%}

%eofval{
   {
      return tok(sym.EOF,null);
   }
%eofval}

DIGIT=[0-9]
WHITE_SPACE=[\ \t\n\015]

%%

...
```

[QUESTION CONTINUED ON NEXT PAGE]

(c) Extend the following CUP parser definition so that arithmetic expressions containing integer numbers, the operators "+", (binary and unary) "-", "*", and "/" as well as parentheses ("(" and ")") can be parsed. Ensure that the operators have the common algebraic priorities ("*", "/" higher than "+", "-") and give unary "-" precedence over other operators. Use CUP precedence directives.

Also write semantic actions so that the total number of integer constants in the input is counted and the maximum nesting level is determined. [*12 marks*]

Example:

Expression	Max. nesting level	No. of constants
23+6	0	2
(12-7)+3	1	3
23+6+(34+2)*(23-(6-(5)))	3	7

Note: The maximum nesting level is returned as the value of "program", whereas the number of integer constants is stored in a variable ("constants") in the "Parser" class.

```
import java.io.*;

parser code {:
   // Counter for constants
   public int constants = 0;

   private Scanner lexer;

   public Parser(Scanner l) {
      lexer = l;
   }
:};

scan with {: return lexer.nextToken(); :};

...

start with program;

program ::= ...
```

2. (a) Consider the following Java program fragment: [*4 marks*]

```
1: class Test_1 {
2:    int j;
3:
4:    int f(int) {
5:       Test_2 t;
6:       t.j = 3x;
7:    }
8: ];
```

While compiling this program, a Java compiler gives the following error messages:

```
line 4: missing parameter name
line 5: Test_2 not declared
line 6: nondigits in number
line 8: syntax error at or near symbol ]
```

i. Which phase of the compilation detects the error reported by the first message?
ii. Which phase of the compilation detects the error reported by the second message?
iii. Which phase of the compilation detects the error reported by the third message?
iv. Which phase of the compilation detects the error reported by the fourth message?

(b) Briefly explain why the front-end of a compiler is usually divided into two stages: a lexer and a parser. Consider all important aspects. [*3 marks*]

(c) We want to extend PL/0 with a GOTO instruction and labels as possible targets of a jump. The labels define a class of tokens "LABEL". The CUP grammar is extended as follows:

```
statement ::= label statement
|             GOTO label
|             ...
```

[QUESTION CONTINUED ON NEXT PAGE]

i. What extensions of the symbol table are necessary? [*3 marks*]

ii. How would you deal with a GOTO whose label has not yet been processed during the semantic analysis? [*4 marks*]

iii. Explain briefly and informally the necessary actions of the semantic analysis stage for

  A. a GOTO instruction

  B. a label

  following your suggested strategy in (2.c.ii). [*4 marks*]

iv. A. GOTO jumps into a WHILE loops are illegal. What difficulties arise if one wants to check this during the semantic analysis? [*2 marks*]

  B. Suggest a solution to the problem in (2.c.iv.A). Note that GOTOs to a label in the same loop are permitted. [*5 marks*]

3. (a) Consider the context-free grammar

- $S \rightarrow aSbS$
- $S \rightarrow bSaS$
- $S \rightarrow \epsilon$

i. Show that this grammar is ambiguous either by
   - drawing two different parse trees for the string "aabbab".
   - writing down two right-most derivations of the string "aabbab".

   Note: You are required just to draw the parse trees OR to write down the derivations. You do not need to attempt both tasks. [*4 marks*]

ii. Compute FIRST(S) and FOLLOW(S). [*4 marks*]

iii. Is this grammar LL(1)? Justify your answer. [*5 marks*]

iv. A. What does the stack of a bottom-up parser contain on acceptance of the input? [*1 mark*]

   B. What does the stack of a top-down parser contain on acceptance of the input? [*1 mark*]

(b) Now consider the following program fragment:

```
for(i = 0; i < 100; i++)
{
   if (k < 50) {
      a[i] = b[i] + c;
      c[i] = c[i] + b[i] * c;
   }
   else {
      a[i] = sin(k);
      c[i] = c[i] + a[i];
   }
}
```

Apply appropriate optimising transformations to the given code fragment and write down the resulting program after each transformation. Explain briefly the advantages and disadvantages of the applied transformations. [*10 marks*]