Computer Science 330 Language Implementation Test

6.30-8.00pm Thursday 10th April 2003

Start reading 6.20p.m. Write your name on all sheets of your answer book. Start writing your answers at 6.30pm. Stop writing at 8.00p.m.

Remove the staple fastening the question sheets to the answer book, but do not remove the staples from the answer book. Read the questions carefully. Hand in your answer book at the front of the class. Always show your working - most marks are for showing you know what you are doing, rather than just getting the right answer. Attempt all questions. Questions total 100 marks. The test counts for 20% of the total mark.

Question 1 — 15 Marks

Write regular expressions to match the following tokens. **Read the specifications carefully! They are not necessarily the same as in the assignment or lecture notes.**

(a) A character literal, where a character literal is composed of a single char literal character surrounded by single quotes; a char literal character is any character (except a single quote or backslash), or a \ followed by any character other than the letter u, or a unicode escape; and a unicode escape is \u followed by exactly four hexadecimal digits. For example `'a'`, `'\''`, `'\\'`, `'\u5b09'`. You might like to declare and use named regular expressions.

(5 marks)

(b) A floating point literal. For example, `3.141592654`, `2.99792458e8`, `6.6262e-34`. Assume either the fractional part or the exponent may be omitted, but not both. So `123` and `e63` are not floating point literals. Similarly, the integral part in front of the decimal point may not be omitted, so `.6e5` is not a floating point literal.

(5 marks)

(c) A possibly multi-line Java style comment, where a comment represents text enclosed in "/*…*/", and line breaks can be in UNIX, Windows, or Macintosh style. For example

```
/* Line 1
Line 2 /* and a nested comment */
Line 3 */
```

Assume nested comments are permitted. You will need multiple rules, states and actions to perform the match.

(5 marks)

Question 2 65 marks

Consider the following CUP grammar.

```
terminal LEFT, RIGHT, NEWLINE, MINUS, INC, TIMES, ASSIGN, ERROR;
terminal String NUMBER;
terminal String IDENT;

non terminal StmtList, Stmt;
non terminal Integer Expr, Term, Factor;
non terminal String Variable;

start with StmtList;

StmtList::=

      |
            StmtList Stmt
      ;

Stmt::=
            Variable ASSIGN Expr NEWLINE
      |
            Expr NEWLINE
      |
            NEWLINE
      ;

Expr::=
            Expr MINUS Term
      |
            Term
      ;

Term::=
            Term TIMES Factor
      |
            Factor
      ;

Factor::=
            LEFT Expr RIGHT
      |
            NUMBER
      |
            Variable
      |
            INC Variable
      |
            Variable INC
      |
            MINUS Factor
      ;

Variable::=
            IDENT
      ;
```

Assume that

- The terminal symbols LEFT, RIGHT, NEWLINE, MINUS, INC, TIMES, ASSIGN correspond to "`(`", "`)`", "`\n`", "`-`", "`++`", "`*`", "`=`".
- IDENT corresponds to an identifier, and NUMBER corresponds to a decimal integer.

(a) Using the information provided in the appendix, perform a shift-reduce LALR(1) parse of the input

```
b = 100 \n
a = ++ b * - 200 \n
```

Show both the symbols and states on the stack, the current token, and the action performed at each stage.

(20 marks)

(b) Draw the full parse tree, showing all rules used in the above shift-reduce LALR(1) parse. (7 marks)

(c) (i) Note that StmtList is nullable.

(ii) Draw the first graph, and compute the first sets for this grammar.

(iii) Draw the follow graph, and compute the follow sets for this grammar. (20 marks)

(d) Suppose we are writing a parser that does not build a parse tree, but (like the INTERP1 example) evaluates the construct corresponding to a rule when it reduces by that rule.

Indicate an appropriate action to evaluate the rule

```
Factor::= Variable:name INC ;
```

and return the value of the Factor. Assume Variable returns the name (identifier) of the variable, and that the values of variables are stored in a Hashtable. (8 marks)

(e) State 1 is

```
lalr_state [1]: {
  [Factor ::= (*) Variable INC , {NEWLINE MINUS TIMES }]
  [Factor ::= (*) NUMBER , {NEWLINE MINUS TIMES }]
  [Term ::= (*) Term TIMES Factor , {NEWLINE MINUS TIMES }]
  [StmtList ::= StmtList (*) Stmt , {EOF LEFT NEWLINE MINUS INC NUMBER IDENT }]
  [Stmt ::= (*) NEWLINE , {EOF LEFT NEWLINE MINUS INC NUMBER IDENT }]
  [Variable ::= (*) IDENT , {NEWLINE MINUS INC TIMES ASSIGN }]
  [Factor ::= (*) INC Variable , {NEWLINE MINUS TIMES }]
  [Factor ::= (*) LEFT Expr RIGHT , {NEWLINE MINUS TIMES }]
  [Expr ::= (*) Term , {NEWLINE MINUS }]
  [Stmt ::= (*) Expr NEWLINE , {EOF LEFT NEWLINE MINUS INC NUMBER IDENT }]
  [Factor ::= (*) MINUS Factor , {NEWLINE MINUS TIMES }]
  [Factor ::= (*) Variable , {NEWLINE MINUS TIMES }]
  [Term ::= (*) Factor , {NEWLINE MINUS TIMES }]
  [Expr ::= (*) Expr MINUS Term , {NEWLINE MINUS }]
  [$START ::= StmtList (*) EOF , {EOF }]
  [Stmt ::= (*) Variable ASSIGN Expr NEWLINE , {EOF LEFT NEWLINE MINUS INC
NUMBER IDENT }]
}
transition on NEWLINE to state [13]
transition on INC to state [12]
transition on Factor to state [11]
transition on Term to state [10]
transition on Stmt to state [9]
transition on EOF to state [8]
transition on NUMBER to state [7]
transition on LEFT to state [6]
transition on Expr to state [5]
transition on IDENT to state [4]
transition on Variable to state [3]
transition on MINUS to state [2]
```

(i) Write down the set of items for goto(state 1, Variable) (state 3).

(ii) Write down the set of items for goto(state 1, INC) (a variant of state 12, without merging it with other variants with the same core). Make sure you take the closure.

(10 marks)

Question 3 20 marks

Suppose that we have a computer language, with method declarations in the style of the following example:

```
int f( int[] a, b; char c, d ) {
      …
      }
```

The method starts with the return type (which can be either a Type or "void"), then the name of the method, then the parameter declarations enclosed in "(…)". It is possible to declare several parameters of the same Type, by writing the Type, then a "," separated list of the parameters being declared. Parameter declarations of different Types should be separated by ";"s. If the method has no parameters, it still has the "()"s, with nothing inside.

Write a grammar to parse a (general) method declaration, with the specified syntax. You may assume that grammar rules have been provided for Types and BlockStatements. You do not have to write any actions.

Appendix

Tables for the CUP grammar

Grammar Rules (Productions)

```
16: Variable ::= IDENT
15: Factor ::= MINUS Factor
14: Factor ::= Variable INC
13: Factor ::= INC Variable
12: Factor ::= Variable
11: Factor ::= NUMBER
10: Factor ::= LEFT Expr RIGHT
9: Term ::= Factor
8: Term ::= Term TIMES Factor
7: Expr ::= Term
6: Expr ::= Expr MINUS Term
5: Stmt ::= NEWLINE
4: Stmt ::= Expr NEWLINE
3: Stmt ::= Variable ASSIGN Expr NEWLINE
2: StmtList ::= StmtList Stmt
1: StmtList ::=
0: $START ::= StmtList EOF
```

Action Table

```
From state #0
      EOF:REDUCE(1) LEFT:REDUCE(1) NEWLINE:REDUCE(1)
      MINUS:REDUCE(1) INC:REDUCE(1) NUMBER:REDUCE(1)
      IDENT:REDUCE(1)
From state #1
      EOF:SHIFT(8) LEFT:SHIFT(6) NEWLINE:SHIFT(13)
      MINUS:SHIFT(2) INC:SHIFT(12) NUMBER:SHIFT(7)
      IDENT:SHIFT(4)
From state #2
      LEFT:SHIFT(6) MINUS:SHIFT(2) INC:SHIFT(12)
      NUMBER:SHIFT(7) IDENT:SHIFT(4)
From state #3
      NEWLINE:REDUCE(12) MINUS:REDUCE(12) INC:SHIFT(18)
      TIMES:REDUCE(12) ASSIGN:SHIFT(24)
From state #4
      RIGHT:REDUCE(16) NEWLINE:REDUCE(16) MINUS:REDUCE(16)
      INC:REDUCE(16) TIMES:REDUCE(16) ASSIGN:REDUCE(16)
From state #5
      NEWLINE:SHIFT(23) MINUS:SHIFT(21)
From state #6
      LEFT:SHIFT(6) MINUS:SHIFT(2) INC:SHIFT(12)
      NUMBER:SHIFT(7) IDENT:SHIFT(4)
From state #7
      RIGHT:REDUCE(11) NEWLINE:REDUCE(11) MINUS:REDUCE(11)
      TIMES:REDUCE(11)
From state #8
      EOF:REDUCE(0)
From state #9
      EOF:REDUCE(2) LEFT:REDUCE(2) NEWLINE:REDUCE(2)
      MINUS:REDUCE(2) INC:REDUCE(2) NUMBER:REDUCE(2)
      IDENT:REDUCE(2)
From state #10
      RIGHT:REDUCE(7) NEWLINE:REDUCE(7) MINUS:REDUCE(7)
      TIMES:SHIFT(15)
From state #11
      RIGHT:REDUCE(9) NEWLINE:REDUCE(9) MINUS:REDUCE(9)
      TIMES:REDUCE(9)
```

```
From state #12
      IDENT:SHIFT(4)
From state #13
      EOF:REDUCE(5) LEFT:REDUCE(5) NEWLINE:REDUCE(5)
      MINUS:REDUCE(5) INC:REDUCE(5) NUMBER:REDUCE(5)
      IDENT:REDUCE(5)
From state #14
      RIGHT:REDUCE(13) NEWLINE:REDUCE(13) MINUS:REDUCE(13)
      TIMES:REDUCE(13)
From state #15
      LEFT:SHIFT(6) MINUS:SHIFT(2) INC:SHIFT(12)
      NUMBER:SHIFT(7) IDENT:SHIFT(4)
From state #16
      RIGHT:REDUCE(8) NEWLINE:REDUCE(8) MINUS:REDUCE(8)
      TIMES:REDUCE(8)
From state #17
      RIGHT:REDUCE(12) NEWLINE:REDUCE(12) MINUS:REDUCE(12)
      INC:SHIFT(18) TIMES:REDUCE(12)
From state #18
      RIGHT:REDUCE(14) NEWLINE:REDUCE(14) MINUS:REDUCE(14)
      TIMES:REDUCE(14)
From state #19
      RIGHT:SHIFT(20) MINUS:SHIFT(21)
From state #20
      RIGHT:REDUCE(10) NEWLINE:REDUCE(10) MINUS:REDUCE(10)
      TIMES:REDUCE(10)
From state #21
      LEFT:SHIFT(6) MINUS:SHIFT(2) INC:SHIFT(12)
      NUMBER:SHIFT(7) IDENT:SHIFT(4)
From state #22
      RIGHT:REDUCE(6) NEWLINE:REDUCE(6) MINUS:REDUCE(6)
      TIMES:SHIFT(15)
From state #23
      EOF:REDUCE(4) LEFT:REDUCE(4) NEWLINE:REDUCE(4)
      MINUS:REDUCE(4) INC:REDUCE(4) NUMBER:REDUCE(4)
      IDENT:REDUCE(4)
From state #24
      LEFT:SHIFT(6) MINUS:SHIFT(2) INC:SHIFT(12)
      NUMBER:SHIFT(7) IDENT:SHIFT(4)
From state #25
      NEWLINE:SHIFT(26) MINUS:SHIFT(21)
From state #26
      EOF:REDUCE(3) LEFT:REDUCE(3) NEWLINE:REDUCE(3)
      MINUS:REDUCE(3) INC:REDUCE(3) NUMBER:REDUCE(3)
      IDENT:REDUCE(3)
From state #27
      RIGHT:REDUCE(15) NEWLINE:REDUCE(15) MINUS:REDUCE(15)
      TIMES:REDUCE(15)
```

Reduce (GoTo) Table

```
From state #0:
      StmtList:GOTO(1)
From state #1:
      Stmt:GOTO(9)
      Expr:GOTO(5)
      Term:GOTO(10)
      Factor:GOTO(11)
      Variable:GOTO(3)
From state #2:
      Factor:GOTO(27)
      Variable:GOTO(17)
From state #3:
From state #4:
From state #5:
From state #6:
      Expr:GOTO(19)
      Term:GOTO(10)
      Factor:GOTO(11)
      Variable:GOTO(17)
From state #7:
From state #8:
From state #9:
From state #10:
From state #11:
From state #12:
      Variable:GOTO(14)
From state #13:
From state #14:
From state #15:
      Factor:GOTO(16)
      Variable:GOTO(17)
From state #16:
From state #17:
From state #18:
From state #19:
From state #20:
From state #21:
      Term:GOTO(22)
      Factor:GOTO(11)
      Variable:GOTO(17)
From state #22:
From state #23:
From state #24:
      Expr:GOTO(25)
      Term:GOTO(10)
      Factor:GOTO(11)
      Variable:GOTO(17)
From state #25:
From state #26:
From state #27:
```

This page left blank for formatting purposes

Computer Science 330

Language Implementation Test

Thursday 10th April 2003

Answer Booklet

Surname
Given Names
Student ID Number
Login Name
Normal Signature

1 _____ /15

2(a)_____ /20

2(b) _____ /7

2(c)_____ /20

2(d) _____ /8

2(e)_____ /10

3 _____ /20

Total _____ /100

Do not write on this page.

It will not be returned to you.

Print your login name _____

Question 1 15 Marks

Write regular expressions to match the following tokens. **Read the specifications carefully! They are not necessarily the same as in the assignment or lecture notes.**

(a) A character literal.

(5 marks)

(b) A floating point literal.

(5 marks)

(d) A possibly multi-line Java style comment.

(5 marks)

Question 2 **65 marks**

(a) Using the information provided in the appendix, perform a shift-reduce LALR(1) parse of the input

```
b = 100 \n
a = ++ b * - 200 \n
```

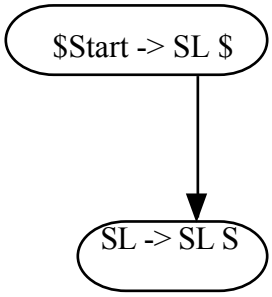
Show both the symbols and states on the stack, the current token, and the action performed at each stage.

Stack								Input	Action
$0								ID b	Red SL -> □
$0	SL 1								Shift ID 4
$0	SL 1	ID 4						=	
$0	SL 1								
$0	SL 1							100	
$0	SL 1							\n	
$0	SL 1								
$0	SL 1								
$0	SL 1								
$0	SL 1							ID a	
$0	SL 1								
$0	SL 1								
$0	SL 1								
$0	SL 1								
$0	SL 1								
$0	SL 1								
$0	SL 1								
$0	SL 1								
$0	SL 1								
$0	SL 1								
$0	SL 1								
$0	SL 1								
$0	SL 1								
$0	SL 1								
$0	SL 1								
$0	SL 1								
$0	SL 1								
$0	SL 1							$	
$0	SL 1								
$0	SL 1								Shift $ 8
$0	SL 1	$ 8						$	Red $Start -> SL $
$0	$Start	-1							Accept

(20 marks)

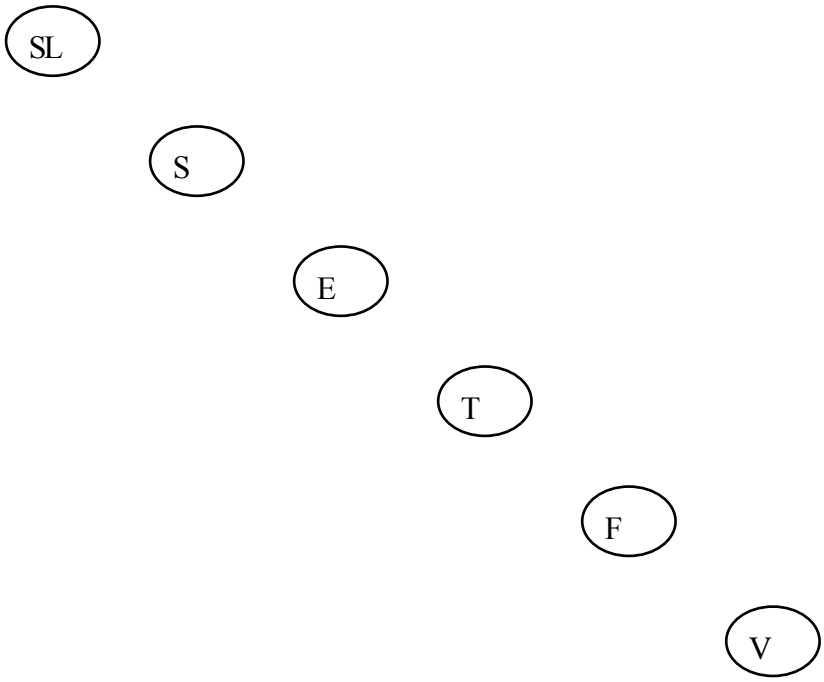
Print your login name _____

(b) Draw the full parse tree, showing all rules used in the above shift-reduce LALR(1) parse. (7 marks)

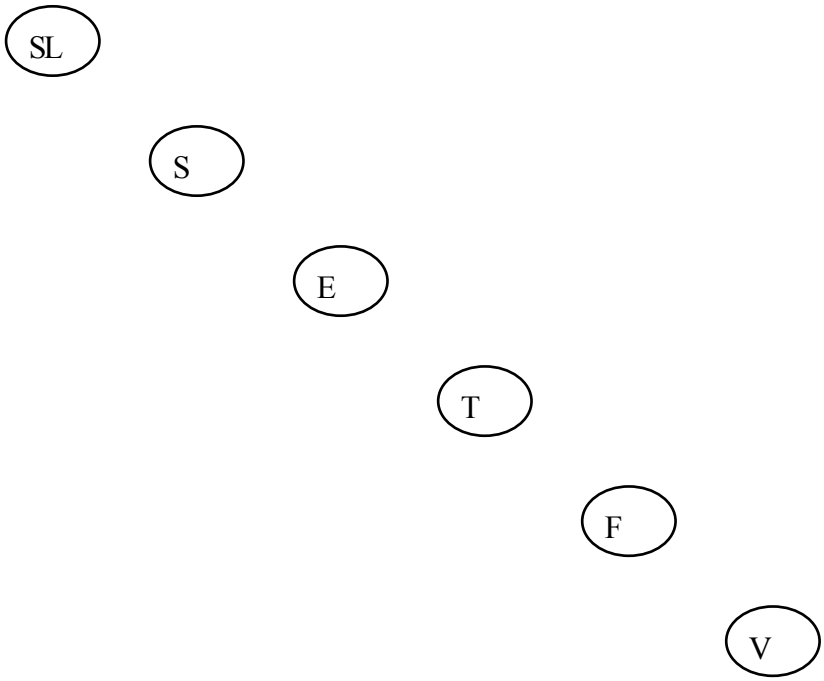


(c) (20 marks)

Draw the first graph for this grammar.



Draw the follow graph for this grammar.



Print your login name _____

Indicate the first and follow sets for the grammar

Symbol	First Set	Follow Set
StmtList		
Stmt		
Expr		
Term		
Factor		
Variable		

(d) Indicate an appropriate action to evaluate the rule

```
Factor::= Variable:name INC ;
```

and return the value of the Factor. Assume Variable returns the name (identifier) of the variable, and that the values of variables are stored in a Hashtable. (8 marks)

(e) State 1 is …

(i) Write down the set of items for goto(state 1, Variable). (5 marks)

(ii) Write down the set of items for goto(state 1, INC). (5 marks)

Print your login name _____

Question 3 20 marks

Write a grammar to parse a (general) method declaration, with the specified syntax. You may assume that grammar rules have been provided for Types and BlockStatements. You do not have to write any actions.