Computer Science 330 Language Implementation Test

6.20-8.00pm Thursday 8th April 2004

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 JFlex rules to match the following tokens, and where appropriate return a value, or perform some appropriate action.

(a) An octal integer. For example 077, 064, but not 0, 64, 069.

(2 marks)

(b) A hexadecimal integer. For example 0xff, 0xFF, 0XFf, 0x123456789abcdef, but not ff, 0ff, xff.

(2 marks)

(c) An identifier, possibly including underscores. For example, x1, hello, banana_boat, but not 1x.

(2 marks)

(d) A Windows, Macintosh or UNIX line break, with an action to increment a line count. Do not return a token.

(2 marks)

(e) Text enclosed in {:...:}. For example,

```
{:
RESULT = new NegateNode( expr );
:}
```

Add appropriate actions to increment the line count, when a line break occurs.

You may assume {:...:} may not be nested. Return the text enclosed inside the {:...:} as the value of a terminal symbol ACTION.

(7 marks)

Question 2 65 marks

Consider the following CUP grammar.

```
terminal IF, THEN, ELSE, WHILE, DO;
terminal LEFT, RIGHT, LEFTCURLY, RIGHTCURLY, SEMICOLON, ASSIGN, ERROR;
terminal String NUMBER;
terminal String IDENT;

non terminal StmtList;
non terminal Stmt;
non terminal Expr;
non terminal Variable;
non terminal Constant;

start with StmtList;

StmtList::=
            /* Empty */
      |
            StmtList Stmt
      ;

Stmt::=
            Variable ASSIGN Expr SEMICOLON
      |
            Expr SEMICOLON
      |
            SEMICOLON
      |
            LEFTCURLY StmtList RIGHTCURLY
      |
            IF Expr THEN Stmt
      |
            IF Expr THEN Stmt ELSE Stmt
      |
            WHILE Expr DO Stmt
      ;

Expr::=
            LEFT Expr RIGHT
      |
            Variable
      |
            Constant
      ;

Variable::=
            IDENT
      ;

Constant::=
            NUMBER
      ;
```

The grammar has a shift/reduce conflict related to the if statements, that is resolved by shifting rather than reducing.

Assume that

- The terminal symbols IF, THEN, ELSE, WHILE, DO, LEFT, RIGHT, LEFTCURLY, RIGHTCURLY, SEMICOLON, ASSIGN correspond to "`if`", "`then`", "`else`", "`while`", "`do`", "`(`", "`)`", "`{`", "`}`", "`;`", "`=`".
- 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

```
if a then while b do c = 55 ; else d ;
```

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. (8 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. (27 marks)

(d) State 1 is

```
lalr_state [1]: {
  [Constant ::= (*) NUMBER , {SEMICOLON }]
  [Expr ::= (*) Variable , {SEMICOLON }]
  [Stmt ::= (*) IF Expr THEN Stmt ELSE Stmt ,
      {EOF IF WHILE LEFT LEFTCURLY SEMICOLON NUMBER IDENT }]
  [StmtList ::= StmtList (*) Stmt ,
      {EOF IF WHILE LEFT LEFTCURLY SEMICOLON NUMBER IDENT }]
  [Stmt ::= (*) SEMICOLON ,
      {EOF IF WHILE LEFT LEFTCURLY SEMICOLON NUMBER IDENT }]
  [Variable ::= (*) IDENT , {SEMICOLON ASSIGN }]
  [Expr ::= (*) LEFT Expr RIGHT , {SEMICOLON }]
  [Stmt ::= (*) IF Expr THEN Stmt ,
      {EOF IF WHILE LEFT LEFTCURLY SEMICOLON NUMBER IDENT }]
  [Stmt ::= (*) Expr SEMICOLON ,
      {EOF IF WHILE LEFT LEFTCURLY SEMICOLON NUMBER IDENT }]
  [Expr ::= (*) Constant , {SEMICOLON }]
  [Stmt ::= (*) WHILE Expr DO Stmt ,
      {EOF IF WHILE LEFT LEFTCURLY SEMICOLON NUMBER IDENT }]
  [Stmt ::= (*) LEFTCURLY StmtList RIGHTCURLY ,
      {EOF IF WHILE LEFT LEFTCURLY SEMICOLON NUMBER IDENT }]
  [$START ::= StmtList (*) EOF , {EOF }]
  [Stmt ::= (*) Variable ASSIGN Expr SEMICOLON ,
      {EOF IF WHILE LEFT LEFTCURLY SEMICOLON NUMBER IDENT }]
}
transition on LEFTCURLY to state [13]
transition on SEMICOLON to state [12]
transition on Stmt to state [11]
transition on IF to state [10]
transition on EOF to state [9]
transition on Constant 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 WHILE to state [2]
]
```

Write down the set of items for goto(state 1, IF) (state 10). Make sure you take the closure.

(10 marks)

Question 3 20 marks

When writing a CUP grammar rule, it is possible to write such things as:

```
Stmt::=
            Variable:variable ASSIGN Expr:expr SEMICOLON
        |
            Expr:expr SEMICOLON
        |
            SEMICOLON
        |
            LEFTCURLY StmtList:stmtList RIGHTCURLY
        |
            IF Expr:cond THEN Stmt:stmt1
        |
            IF Expr:cond THEN Stmt:stmt1 ELSE Stmt:stmt2
        |
            WHILE Expr:cond DO Stmt:stmt1
        ;
```

A grammar rule has an identifier on the left hand side, then "::=", then one or more alternative right hand sides, then a ";". Each alternative right hand side corresponds to zero or more symbols. A symbol is represented by an identifier, with an optional ":" and a label identifier. In fact it is also possible to write grammar rules such as

```
Expr::=
            Expr:expr1 MINUS Expr:expr2
            {:
            RESULT = new MinusNode( expr1, expr2 );
            :}
        |
            MINUS Expr:expr
            {:
            RESULT = new NegateNode( expr );
            :}
            %prec PREFIX
        |
            INCR Expr:expr
            {:
            RESULT = new PreIncrNode( expr );
            :}
            %prec PREFIX
        ;
```

with an optional action and precedence specification at the end of each alternative right hand side. (Don't allow actions in the middle of an alternative right hand side.)

Write a grammar to parse a CUP style grammar rule. You do not have to write any actions. Assume the lexical analyser returns a single ACTION terminal symbol for an action such as

```
            {:
            RESULT = new MinusNode( expr1, expr2 );
            :}
```

and that you do not have to analyse the action.

Appendix

Tables for the CUP grammar

Grammar Rules (Productions)

```
14: Constant ::= NUMBER
13: Variable ::= IDENT
12: Expr ::= Constant
11: Expr ::= Variable
10: Expr ::= LEFT Expr RIGHT
9: Stmt ::= WHILE Expr DO Stmt
8: Stmt ::= IF Expr THEN Stmt ELSE Stmt
7: Stmt ::= IF Expr THEN Stmt
6: Stmt ::= LEFTCURLY StmtList RIGHTCURLY
5: Stmt ::= SEMICOLON
4: Stmt ::= Expr SEMICOLON
3: Stmt ::= Variable ASSIGN Expr SEMICOLON
2: StmtList ::= StmtList Stmt
1: StmtList ::=
0: $START ::= StmtList EOF
```

Action Table

```
From state #0
      EOF:REDUCE(1) IF:REDUCE(1) WHILE:REDUCE(1)
      LEFT:REDUCE(1) LEFTCURLY:REDUCE(1) SEMICOLON:REDUCE(1)
      NUMBER:REDUCE(1) IDENT:REDUCE(1)
From state #1
      EOF:SHIFT(9) IF:SHIFT(10) WHILE:SHIFT(2)
      LEFT:SHIFT(6) LEFTCURLY:SHIFT(13) SEMICOLON:SHIFT(12)
      NUMBER:SHIFT(7) IDENT:SHIFT(4)
From state #2
      LEFT:SHIFT(6) NUMBER:SHIFT(7) IDENT:SHIFT(4)
From state #3
      SEMICOLON:REDUCE(11) ASSIGN:SHIFT(25)
From state #4
      THEN:REDUCE(13) DO:REDUCE(13) RIGHT:REDUCE(13)
      SEMICOLON:REDUCE(13) ASSIGN:REDUCE(13)
From state #5
      SEMICOLON:SHIFT(24)
From state #6
      LEFT:SHIFT(6) NUMBER:SHIFT(7) IDENT:SHIFT(4)
From state #7
      THEN:REDUCE(14) DO:REDUCE(14) RIGHT:REDUCE(14)
      SEMICOLON:REDUCE(14)
From state #8
      THEN:REDUCE(12) DO:REDUCE(12) RIGHT:REDUCE(12)
      SEMICOLON:REDUCE(12)
From state #9
      EOF:REDUCE(0)
From state #10
      LEFT:SHIFT(6) NUMBER:SHIFT(7) IDENT:SHIFT(4)
From state #11
      EOF:REDUCE(2) IF:REDUCE(2) WHILE:REDUCE(2)
      LEFT:REDUCE(2) LEFTCURLY:REDUCE(2) RIGHTCURLY:REDUCE(2)
      SEMICOLON:REDUCE(2) NUMBER:REDUCE(2) IDENT:REDUCE(2)
From state #12
      EOF:REDUCE(5) IF:REDUCE(5) ELSE:REDUCE(5)
      WHILE:REDUCE(5) LEFT:REDUCE(5) LEFTCURLY:REDUCE(5)
      RIGHTCURLY:REDUCE(5) SEMICOLON:REDUCE(5) NUMBER:REDUCE(5)
      IDENT:REDUCE(5)
```

```
From state #13
      IF:REDUCE(1) WHILE:REDUCE(1) LEFT:REDUCE(1)
      LEFTCURLY:REDUCE(1) RIGHTCURLY:REDUCE(1) SEMICOLON:REDUCE(1)
      NUMBER:REDUCE(1) IDENT:REDUCE(1)
From state #14
      IF:SHIFT(10) WHILE:SHIFT(2) LEFT:SHIFT(6)
      LEFTCURLY:SHIFT(13) RIGHTCURLY:SHIFT(15) SEMICOLON:SHIFT(12)
      NUMBER:SHIFT(7) IDENT:SHIFT(4)
From state #15
      EOF:REDUCE(6) IF:REDUCE(6) ELSE:REDUCE(6)
      WHILE:REDUCE(6) LEFT:REDUCE(6) LEFTCURLY:REDUCE(6)
      RIGHTCURLY:REDUCE(6) SEMICOLON:REDUCE(6) NUMBER:REDUCE(6)
      IDENT:REDUCE(6)
From state #16
      THEN:REDUCE(11) DO:REDUCE(11) RIGHT:REDUCE(11)
      SEMICOLON:REDUCE(11)
From state #17
      THEN:SHIFT(18)
From state #18
      IF:SHIFT(10) WHILE:SHIFT(2) LEFT:SHIFT(6)
      LEFTCURLY:SHIFT(13) SEMICOLON:SHIFT(12) NUMBER:SHIFT(7)
      IDENT:SHIFT(4)
From state #19
      EOF:REDUCE(7) IF:REDUCE(7) ELSE:SHIFT(20)
      WHILE:REDUCE(7) LEFT:REDUCE(7) LEFTCURLY:REDUCE(7)
      RIGHTCURLY:REDUCE(7) SEMICOLON:REDUCE(7) NUMBER:REDUCE(7)
      IDENT:REDUCE(7)
From state #20
      IF:SHIFT(10) WHILE:SHIFT(2) LEFT:SHIFT(6)
      LEFTCURLY:SHIFT(13) SEMICOLON:SHIFT(12) NUMBER:SHIFT(7)
      IDENT:SHIFT(4)
From state #21
      EOF:REDUCE(8) IF:REDUCE(8) ELSE:REDUCE(8)
      WHILE:REDUCE(8) LEFT:REDUCE(8) LEFTCURLY:REDUCE(8)
      RIGHTCURLY:REDUCE(8) SEMICOLON:REDUCE(8) NUMBER:REDUCE(8)
      IDENT:REDUCE(8)
From state #22
      RIGHT:SHIFT(23)
From state #23
      THEN:REDUCE(10) DO:REDUCE(10) RIGHT:REDUCE(10)
      SEMICOLON:REDUCE(10)
From state #24
      EOF:REDUCE(4) IF:REDUCE(4) ELSE:REDUCE(4)
      WHILE:REDUCE(4) LEFT:REDUCE(4) LEFTCURLY:REDUCE(4)
      RIGHTCURLY:REDUCE(4) SEMICOLON:REDUCE(4) NUMBER:REDUCE(4)
      IDENT:REDUCE(4)
From state #25
      LEFT:SHIFT(6) NUMBER:SHIFT(7) IDENT:SHIFT(4)
From state #26
      SEMICOLON:SHIFT(27)
From state #27
      EOF:REDUCE(3) IF:REDUCE(3) ELSE:REDUCE(3)
      WHILE:REDUCE(3) LEFT:REDUCE(3) LEFTCURLY:REDUCE(3)
      RIGHTCURLY:REDUCE(3) SEMICOLON:REDUCE(3) NUMBER:REDUCE(3)
      IDENT:REDUCE(3)
From state #28
      DO:SHIFT(29)
From state #29
      IF:SHIFT(10) WHILE:SHIFT(2) LEFT:SHIFT(6)
      LEFTCURLY:SHIFT(13) SEMICOLON:SHIFT(12) NUMBER:SHIFT(7)
      IDENT:SHIFT(4)
From state #30
      EOF:REDUCE(9) IF:REDUCE(9) ELSE:REDUCE(9)
      WHILE:REDUCE(9) LEFT:REDUCE(9) LEFTCURLY:REDUCE(9)
      RIGHTCURLY:REDUCE(9) SEMICOLON:REDUCE(9) NUMBER:REDUCE(9)
      IDENT:REDUCE(9)
```

Reduce (GoTo) Table

```
From state #0:
      StmtList:GOTO(1)
From state #1:
      Stmt:GOTO(11)
      Expr:GOTO(5)
      Variable:GOTO(3)
      Constant:GOTO(8)
From state #2:
      Expr:GOTO(28)
      Variable:GOTO(16)
      Constant:GOTO(8)
From state #3:
From state #4:
From state #5:
From state #6:
      Expr:GOTO(22)
      Variable:GOTO(16)
      Constant:GOTO(8)
From state #7:
From state #8:
From state #9:
From state #10:
      Expr:GOTO(17)
      Variable:GOTO(16)
      Constant:GOTO(8)
From state #11:
From state #12:
From state #13:
      StmtList:GOTO(14)
From state #14:
      Stmt:GOTO(11)
      Expr:GOTO(5)
      Variable:GOTO(3)
      Constant:GOTO(8)
From state #15:
From state #16:
From state #17:
From state #18:
      Stmt:GOTO(19)
      Expr:GOTO(5)
      Variable:GOTO(3)
      Constant:GOTO(8)
From state #19:
From state #20:
      Stmt:GOTO(21)
      Expr:GOTO(5)
      Variable:GOTO(3)
      Constant:GOTO(8)
From state #21:
From state #22:
From state #23:
From state #24:
From state #25:
      Expr:GOTO(26)
      Variable:GOTO(16)
      Constant:GOTO(8)
From state #26:
From state #27:
From state #28:
From state #29:
      Stmt:GOTO(30)
      Expr:GOTO(5)
      Variable:GOTO(3)
      Constant:GOTO(8)
From state #30:
```

This page left blank for formatting purposes

Computer Science 330

Language Implementation Test

Thursday 8th April 2004

Answer Booklet

Surname
Given Names
Student ID Number
Login Name
Normal Signature

1 _____ /15

2(a)_____ /20

2(b) _____ /8

2(c)_____ /27

2(d) _____ /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 JFlex rules to match the following tokens, and where appropriate return a value, or perform some appropriate action.

(a) An octal integer. For example 077, 064, but not 0, 64, 069.

(2 marks)

(b) A hexadecimal integer. For example 0xff, 0xFF, 0XFf, 0x123456789abcdef, but not ff, 0ff, xff.

(2 marks)

(c) An identifier, possibly including underscores. For example, x1, hello, banana_boat, but not 1x.

(2 marks)

(d) A Windows, Macintosh or UNIX line break, with an action to increment a line count. Do not return a token.

(2 marks)

(e) Text enclosed in {:...:}, with actions to increment line breaks.

(7 marks)

Question 2 65 marks

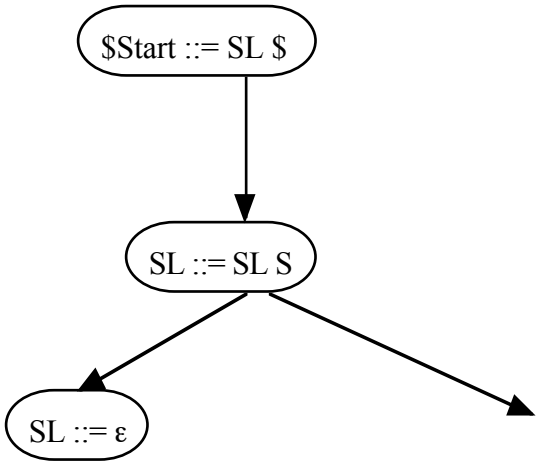
(a) Using the information provided in the appendix, perform a shift-reduce LALR(1) parse of the input

```
if a then while b do c = 55 ; else d ;
```

Show both the symbols and states on the stack, the current token, and the action performed at each stage. (20 marks)

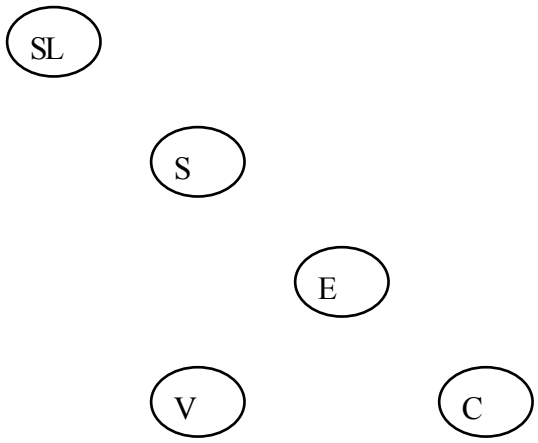
Print your login name _____

(b) Draw the full parse tree, showing all rules used in the above shift-reduce LALR(1) parse. (8 marks)

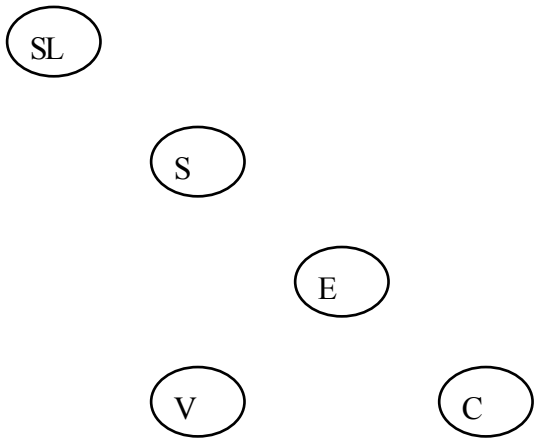


(c) (27 marks)

Draw the first graph for this grammar.



Draw the follow graph for this grammar.



Indicate the first and follow sets for the grammar

Symbol	First Set	Follow Set
StmtList		
Stmt		
Expr		
Variable		
Constant		

Print your login name _____

(d) State 1 is … (10 marks)

Write down the set of items for goto(state 1, IF) (state 10).

Print your login name _____

Question 3 20 marks

Write a grammar to parse a CUP style grammar rule. You do not have to write any actions.