Computer Science 330 Language Implementation Test 2002 Thursday 18th April 6.30-8.00pm

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 10% of the total mark.

Question 1 — 15 Marks

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

(a) A binary number, where a binary number is the text "`0b`" or "`0B`", followed by one or more binary digits.

(3 marks)

(b) A single regular expression to match a line break for UNIX (linefeed), Windows (carriage return/linefeed), or Macintosh (carriage return).

(2 marks)

(c) A String literal, where a string literal is composed of zero or more string characters surrounded by double quotes; a string character is any character (except a double quote or backslash), or an octal escape; and an octal escape is a \ followed by exactly three octal digits, where the first octal digit must be either "0" or "1". For example `""`, `"abc"`, `"\042abc\042"`, `"\015\012\000"`. You might like to declare and use named regular expressions.

(5 marks)

(d) 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
Line 3 */
```

The internal text within a comment must not contain either "`/*`" (which should cause an error to be thrown) or "`*/`" (because that represents the end of the comment). You will need multiple rules, states and actions to perform the match.

(5 marks)

Question 2 55 marks

Consider the following CUP grammar.

```
terminal
       IF, THEN, ELSE, WHILE, DO, LEFTBRACE, RIGHTBRACE,
       LEFT, RIGHT, SEMICOLON, COMMA, IDENT;

non terminal
       Stmt, StmtSeq, Expr, ExprSeqOpt, ExprSeq;

start with Stmt;

Stmt::=
              Expr SEMICOLON
       |
              IF Expr THEN Stmt ELSE Stmt
       |
              IF Expr THEN Stmt
       |
              WHILE Expr DO Stmt
       |
              LEFTBRACE StmtSeq RIGHTBRACE
       ;

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

Expr::=
              IDENT
       |
              IDENT LEFT ExprSeqOpt RIGHT
       ;

ExprSeqOpt::=
              /* Empty */
       |
              ExprSeq
       ;

ExprSeq::=
              Expr
       |
              ExprSeq COMMA Expr
       ;
```

Assume that

- The terminal symbols IF, THEN, ELSE, WHILE, DO, LEFTBRACE, RIGHTBRACE, LEFT, RIGHT, SEMICOLON, COMMA correspond to “if”, “then”, “else”, “while”, “do”, “{”, “}”, “(”, “)”, “;”, “,”.
- IDENT corresponds to an identifier.

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

```
if a then { b( c ); }
```

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

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

(c) (i) Indicate which nonterminals are 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. (25 marks)

(d) Indicate how a parse tree could be built, and a treewalk of the parse tree performed for the above grammar, by adding in actions, etc, for the rule for the "while" statement. (3 marks)

Question 3 30 marks

Suppose that we have a computer language, with switch statements like the following example:

```
switch ( i ) {
      +2, +4:
            println( "small and even" );
      1..5:
            println( "small" );
      6..9:
            println( "big" );
      ..-1:
            println( "negative" );
      ..:
            println( "default" );
      }
```

The basic syntax is "switch (Expr) { CaseStmtList }". A case statement has a "," separated list of label ranges, then a ":", then a substatement. A label range can be a single integer, or a range of integers, with a ".." in between. The lower bound may be omitted, for -∞, and the upper bound omitted for +∞. An optional "+" or "-" sign is permitted as part of the integer. Assume the lexical analyser treats signed integer literals as two tokens, the sign and the unsigned literal, so the parser has to match two tokens. There is no "default", because ".." has an equivalent effect.

Write a grammar to parse a (general) switch statement, with the specified syntax. You may assume that grammar rules have been provided for expressions and statements. You do not have to write any actions.

Appendix

Tables for the CUP grammar

Grammar Rules (Productions)

```
13: ExprSeq ::= ExprSeq COMMA Expr
12: ExprSeq ::= Expr
11: ExprSeqOpt ::= ExprSeq
10: ExprSeqOpt ::=
9: Expr ::= IDENT LEFT ExprSeqOpt RIGHT
8: Expr ::= IDENT
7: StmtSeq ::= StmtSeq Stmt
6: StmtSeq ::=
5: Stmt ::= LEFTBRACE StmtSeq RIGHTBRACE
4: Stmt ::= WHILE Expr DO Stmt
3: Stmt ::= IF Expr THEN Stmt
2: Stmt ::= IF Expr THEN Stmt ELSE Stmt
1: Stmt ::= Expr SEMICOLON
0: $START ::= Stmt EOF
```

Action Table

```
From state #0
       IF:SHIFT(6) WHILE:SHIFT(5) LEFTBRACE:SHIFT(1)
       IDENT:SHIFT(4)
From state #1
       IF:REDUCE(6) WHILE:REDUCE(6) LEFTBRACE:REDUCE(6)
       RIGHTBRACE:REDUCE(6) IDENT:REDUCE(6)
From state #2
       EOF:SHIFT(23)
From state #3
       SEMICOLON:SHIFT(22)
From state #4
       THEN:REDUCE(8) DO:REDUCE(8) LEFT:SHIFT(15)
       RIGHT:REDUCE(8) SEMICOLON:REDUCE(8) COMMA:REDUCE(8)
From state #5
       IDENT:SHIFT(4)
From state #6
       IDENT:SHIFT(4)
From state #7
       THEN:SHIFT(8)
From state #8
       IF:SHIFT(6) WHILE:SHIFT(5) LEFTBRACE:SHIFT(1)
       IDENT:SHIFT(4)
From state #9
       EOF:REDUCE(3) IF:REDUCE(3) ELSE:SHIFT(10)
       WHILE:REDUCE(3) LEFTBRACE:REDUCE(3) RIGHTBRACE:REDUCE(3)
       IDENT:REDUCE(3)
From state #10
       IF:SHIFT(6) WHILE:SHIFT(5) LEFTBRACE:SHIFT(1)
       IDENT:SHIFT(4)
From state #11
       EOF:REDUCE(2) IF:REDUCE(2) ELSE:REDUCE(2)
       WHILE:REDUCE(2) LEFTBRACE:REDUCE(2) RIGHTBRACE:REDUCE(2)
       IDENT:REDUCE(2)
From state #12
```

```
       DO:SHIFT(13)
From state #13
       IF:SHIFT(6) WHILE:SHIFT(5) LEFTBRACE:SHIFT(1)
       IDENT:SHIFT(4)
From state #14
       EOF:REDUCE(4) IF:REDUCE(4) ELSE:REDUCE(4)
       WHILE:REDUCE(4) LEFTBRACE:REDUCE(4) RIGHTBRACE:REDUCE(4)
       IDENT:REDUCE(4)
From state #15
       RIGHT:REDUCE(10) IDENT:SHIFT(4)
From state #16
       RIGHT:REDUCE(12) COMMA:REDUCE(12)
From state #17
       RIGHT:SHIFT(21)
From state #18
       RIGHT:REDUCE(11) COMMA:SHIFT(19)
From state #19
       IDENT:SHIFT(4)
From state #20
       RIGHT:REDUCE(13) COMMA:REDUCE(13)
From state #21
       THEN:REDUCE(9) DO:REDUCE(9) RIGHT:REDUCE(9)
       SEMICOLON:REDUCE(9) COMMA:REDUCE(9)
From state #22
       EOF:REDUCE(1) IF:REDUCE(1) ELSE:REDUCE(1)
       WHILE:REDUCE(1) LEFTBRACE:REDUCE(1) RIGHTBRACE:REDUCE(1)
       IDENT:REDUCE(1)
From state #23
       EOF:REDUCE(0)
From state #24
       IF:SHIFT(6) WHILE:SHIFT(5) LEFTBRACE:SHIFT(1)
       RIGHTBRACE:SHIFT(26) IDENT:SHIFT(4)
From state #25
       IF:REDUCE(7) WHILE:REDUCE(7) LEFTBRACE:REDUCE(7)
       RIGHTBRACE:REDUCE(7) IDENT:REDUCE(7)
From state #26
       EOF:REDUCE(5) IF:REDUCE(5) ELSE:REDUCE(5)
       WHILE:REDUCE(5) LEFTBRACE:REDUCE(5) RIGHTBRACE:REDUCE(5)
       IDENT:REDUCE(5)
```

Reduce (GoTo) Table

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

Computer Science 330

Language Implementation Test 2002

Thursday 18th April

Answer Booklet

Surname
Given Names
Student ID Number
Login Name
Normal Signature

1 _____/15

2(a) _____/20

2(b) _____/7

2(c) _____/25

2(d) _____/3

3 _____/30

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 the same as in the assignment or lecture notes.**

(a) A binary number.

(3 marks)

(b) A single regular expression to match a line break for UNIX (linefeed), Windows (carriage return/linefeed), or Macintosh (carriage return).

(2 marks)

(c) A String literal, subject to the conditions specified.

(5 marks)

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

(5 marks)

Question 2 55 marks

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

```
if a then { b( c ); }
```

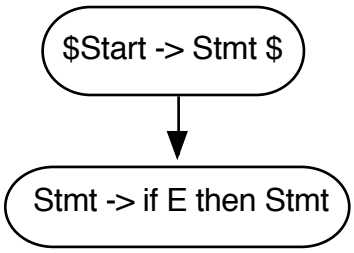
Show both the symbols and states on the stack, the current token, and the action performed at each stage.

Stack										Input	Action
$0										if	Shift if 6
$0	if 6									ID a	Shift ID 4
$0	if 6	ID 4								then	Red E->ID
$0	if 6										
$0	if 6										
$0	if 6										
$0	if 6										
$0	if 6										
$0	if 6										
$0	if 6										
$0	if 6										
$0	if 6										
$0	if 6										
$0	if 6										
$0	if 6										
$0	if 6										
$0	if 6										
$0	if 6										
$0	if 6										
$0	if 6										
$0	S 2										Shift $ 23
$0	S 2	$23								$	Red $Start -> S $
$0	$Start	-1									Accept

(20 marks)

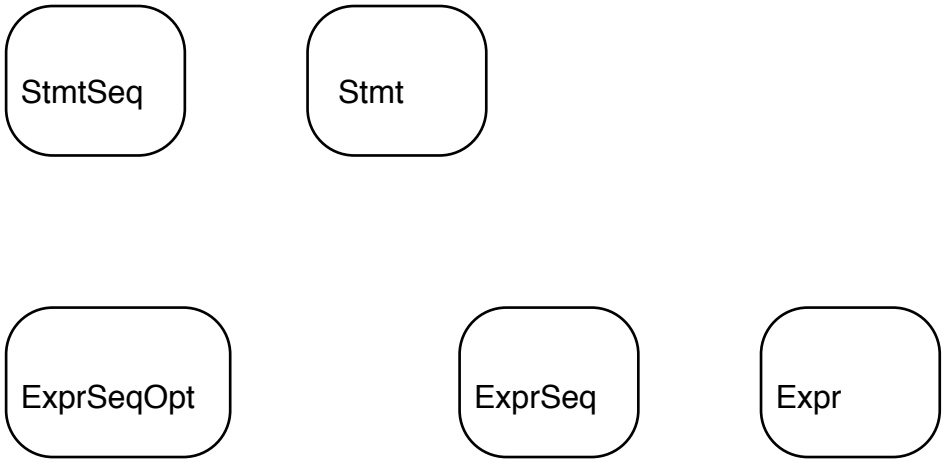
Print your login name _____

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

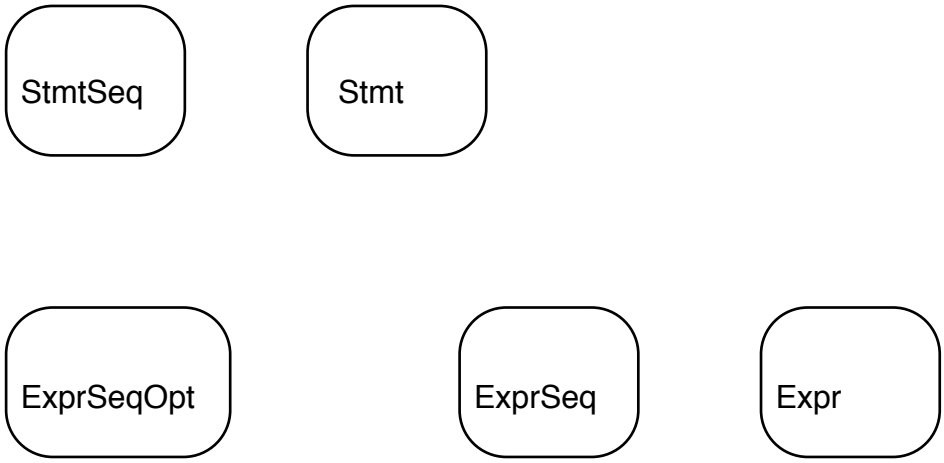


(c) (25 marks)

Draw the first graph for this grammar.



Draw the follow graph for this grammar.



Indicate which nonterminals are nullable, and the first and follow sets for the grammar

Symbol	Nullable	First Set	Follow Set
StmtSeq			
Stmt			
ExprSeqOpt			
ExprSeq			

Expr			

Print your login name _____

(d) Indicate how a parse tree could be built, and a treewalk of the parse tree performed for the above grammar, by adding in actions, etc, for the rule for the "while" statement. (3 marks)

Question 3 30 marks

Write a grammar to parse a (general) switch statement, with the specified syntax. You may assume that grammar rules have been provided for expressions and statements. You do not have to write any actions.

Print your login name _____