Computer Science 330 Language Implementation

2002 Test Solution

Question 1 — 15 Marks

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

(a) A binary number.

```
0[bB][01]+
```

(3 marks)

(b) A line break, for UNIX (linefeed), Windows (carriage return/linefeed), or Macintosh (carriage return).

```
\r|\n|\r\n
```

(2 marks)

(c) A String literal.

```
\"{SChar}*\"
where
SChar  = ([^[\"\\]|\\[01][0-7][0-7])
```

(5 marks)

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

```
<NORMAL>"/*"  {yybegin( COMMENT ); }
<COMMENT>"*/" {yybegin( NORMAL ); }
<COMMENT>"/*" {throw new Error( "Nested comment" ); }
<COMMENT>\r|\n|\r\n { }
<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 ); }
```

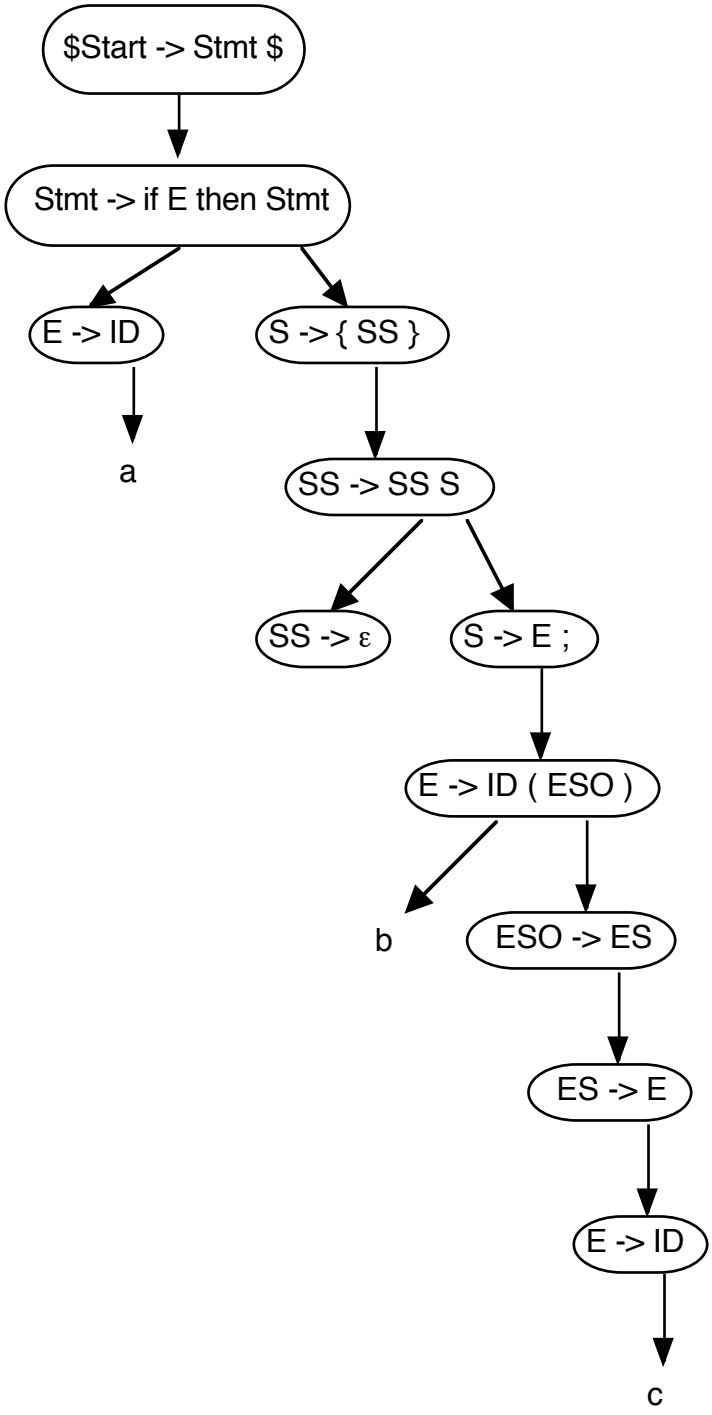
Show both the symbols and states on the stack, the current token, and the action performed at each stage.

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	E 7									Shift then 8
$0	if 6	E 7	then 8							{	Shift { 1
$0	if 6	E 7	then 8	{ 1							Red SS -> ε
$0	if 6	E 7	then 8	{ 1	SS 24					ID b	Shift ID 4
$0	if 6	E 7	then 8	{ 1	SS 24	ID 4				(	Shift (15
$0	if 6	E 7	then 8	{ 1	SS 24	ID 4	(15			ID c	Shift ID 4
$0	if 6	E 7	then 8	{ 1	SS 24	ID 4	(15	ID 4		)	Red E -> ID
$0	if 6	E 7	then 8	{ 1	SS 24	ID 4	(15	E 16			Red ES -> E
$0	if 6	E 7	then 8	{ 1	SS 24	ID 4	(15	ES 18			Red ESO -> ES
$0	if 6	E 7	then 8	{ 1	SS 24	ID 4	(15	ESO 17			Shift) 20
$0	if 6	E 7	then 8	{ 1	SS 24	ID 4	(15	ESO 17	) 20	;	Red E -> ID (ESO)
$0	if 6	E 7	then 8	{ 1	SS 24	E 3					Shift ; 22
$0	if 6	E 7	then 8	{ 1	SS 24	E 3	; 22			}	Red S -> E ;
$0	if 6	E 7	then 8	{ 1	SS 24	S 25					Red SS -> SS S
$0	if 6	E 7	then 8	{ 1	SS 24						Shift } 26
$0	if 6	E 7	then 8	{ 1	SS 24	} 26				$	Red S -> { SS }
$0	if 6	E 7	then 8	S 9							Red S -> if E then S
$0	S 2										Shift $ 23
$0	S 2	$23								$	Red $Start -> S $

$0	$Start	-1									Accept

(20 marks)

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



(c)

Nullable symbols (1 mark)

Append terminal symbols for first graph (4 marks)

Arrows for first graph (4 marks).

Compute first set from first graph (2 marks).

Append terminal symbols for follow graph (6 marks)

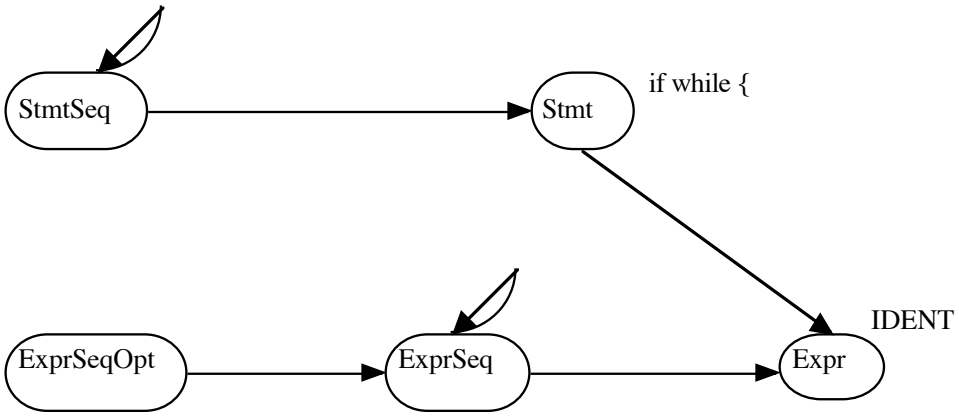
Arrows for follow graph (6 marks).

Compute follow set from follow graph (2 marks).

Base marking on whether shows understanding, not how many symbols/arrows are correct. Take into account the effects of previous errors. Take into account the extent to which their answer makes intuitive sense. For example, they should not have first or follow sets for a nonterminal that are empty.

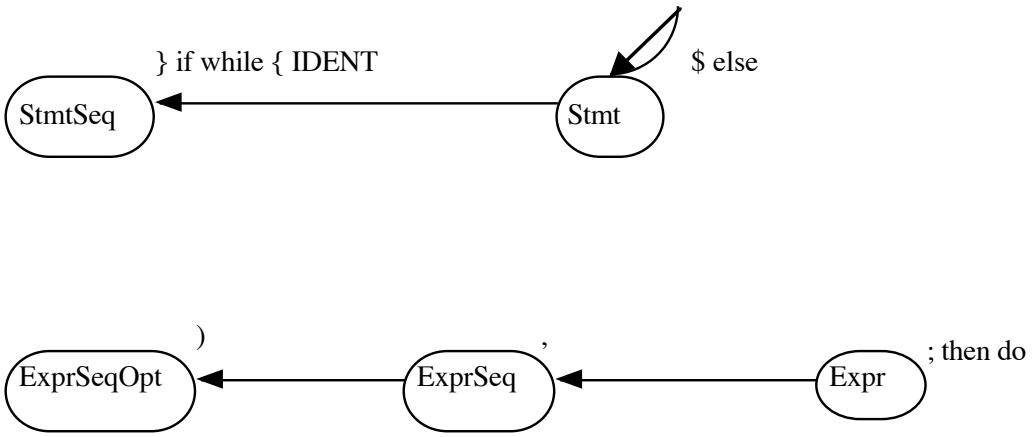
Draw the first graph for this grammar.

First Graph



Draw the follow graph for this grammar.

Follow Graph



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

Symbol	Nullable	First Set	Follow Set
StmtSeq	true	if while { ID	} if while { ID
Stmt	false	if while { ID	$ } if while { ID else
ExprSeqOpt	true	ID	)
ExprSeq	false	ID	) ,
Expr	false	ID	) , ; then do

Question 3 30 marks

Write a grammar to parse a switch statement. You may assume that grammar rules have been provided for expressions and statements. You do not have to write any actions.

```
Stmt::=
        SWITCH LEFT Expr RIGHT
        LEFTBRACE
        CaseStmtList
        RIGHTBRACE
    ;

CaseStmtList::=
        CaseStmt
    |
        CaseStmtList
        CaseStmt
    ;

CaseStmt::=
        LabelRangeList COLON Stmt
    ;

LabelRangeList::=
        LabelRange
    |
        LabelRangeList COMMA LabelRange
    ;

LabelRange::=
        Label
    |
        DOTDOT Label
    |
        Label DOTDOT Label
    |
        Label DOTDOT
    |
        DOTDOT
    ;

Label::=
        INTVALUE
    |
        MINUS INTVALUE
    |
        PLUS INTVALUE
    ;
```