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.

```
charLit = [^\\\'] | \\[^u] | \\u[0-9a-fA-F]{4}
%%
\'  {charLit} \'
```

(5 marks)

(b) A floating point literal.

```
[0-9]+ \. [0-9]+ e [\+\-]? [0-9]+ | [0-9]+ e [\+\-]? [0-9]+ | [0-9]+ \. [0-9]+
```

(5 marks)

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

```
%{
int nest = 0;
%}
%state NORMAL, COMMENT;
%%
<NORMAL> {
        "/*"                    { yybegin( COMMENT ); nest++; }
        }
<COMMENT> {
        "/*"                    { nest++; }
        "*/"                    { --nest; if ( nest == 0 ) yybegin( NORMAL ); }
        \r|\n|\r\n              { }
        .                       { }
        }
```

(5 marks)

Question 2 — 55 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
```

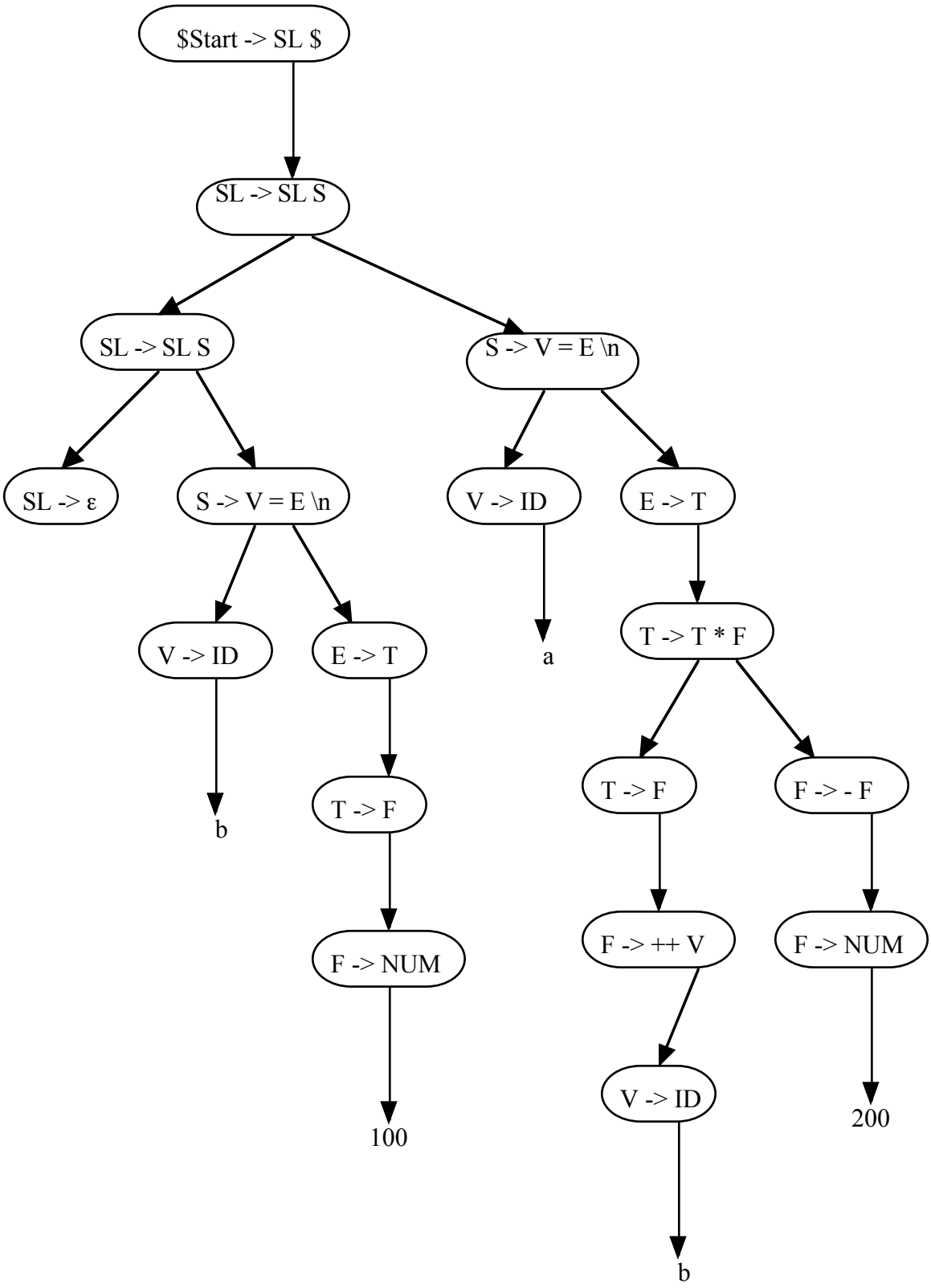
Show both the symbols and states on the stack, the current token, and the action performed at each stage.

Stack								Input	Action
$0								ID b	Red SL -> empty
$0	SL 1								Shift ID 4
$0	SL 1	ID 4						=	Red V -> ID
$0	SL 1	V 3							Shift = 24
$0	SL 1	V 3	= 24					NUM 100	Shift NUM 7
$0	SL 1	V 3	= 24	NUM 7				\n	Red F -> NUM
$0	SL 1	V 3	= 24	F 11					Red T -> F
$0	SL 1	V 3	= 24	T 10					Red E -> T
$0	SL 1	V 3	= 24	E 25					Shift \n 26
$0	SL 1	V 3	= 24	E 25	NL 26			ID a	Red S -> V = E \n
$0	SL 1	S 9							Red SL -> SL S
$0	SL 1								Shift ID 4
$0	SL 1	ID 4						=	Red V -> ID
$0	SL 1	V 3							Shift = 24
$0	SL 1	V 3	= 24					++	Shift ++ 12
$0	SL 1	V 3	= 24	++ 12				ID b	Shift ID 4
$0	SL 1	V 3	= 24	++ 12	ID 4			*	Red V -> ID
$0	SL 1	V 3	= 24	++ 12	V 14				Red F -> ++ V
$0	SL 1	V 3	= 24	F 11					Red T -> F
$0	SL 1	V 3	= 24	T 10					Shift * 15
$0	SL 1	V 3	= 24	T 10	* 15			-	Shift - 2
$0	SL 1	V 3	= 24	T 10	* 15	- 2		NUM 200	Shift NUM 7
$0	SL 1	V 3	= 24	T 10	* 15	- 2	NUM 7	\n	Red F -> NUM
$0	SL 1	V 3	= 24	T 10	* 15	- 2	F 27		Red F -> - F
$0	SL 1	V 3	= 24	T 10	* 15	F 16			Red T -> T * F
$0	SL 1	V 3	= 24	T 10					Red E -> T
$0	SL 1	V 3	= 24	E 25					Shift \n 26
$0	SL 1	V 3	= 24	E 25	NL 26			$	Red S -> V = E \n
$0	SL 1	S 9							Red SL -> SL S
$0	SL 1								Shift $ 8
$0	SL 1	$ 8						$	Red $Start -> SL $
$0	$Start	-1							Accept

(20 marks)

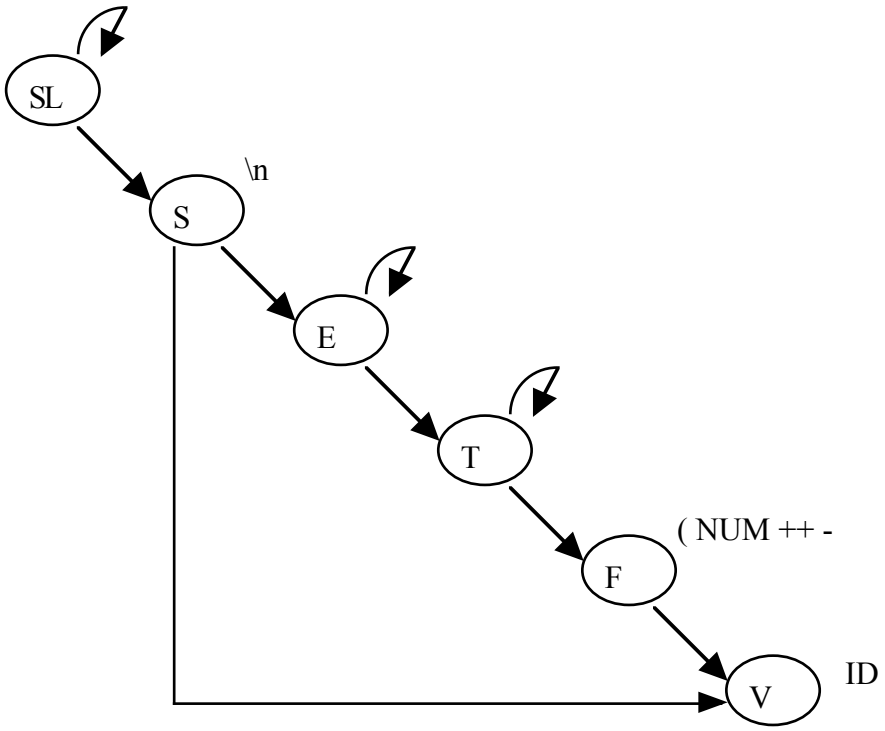
Print your login name _____

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

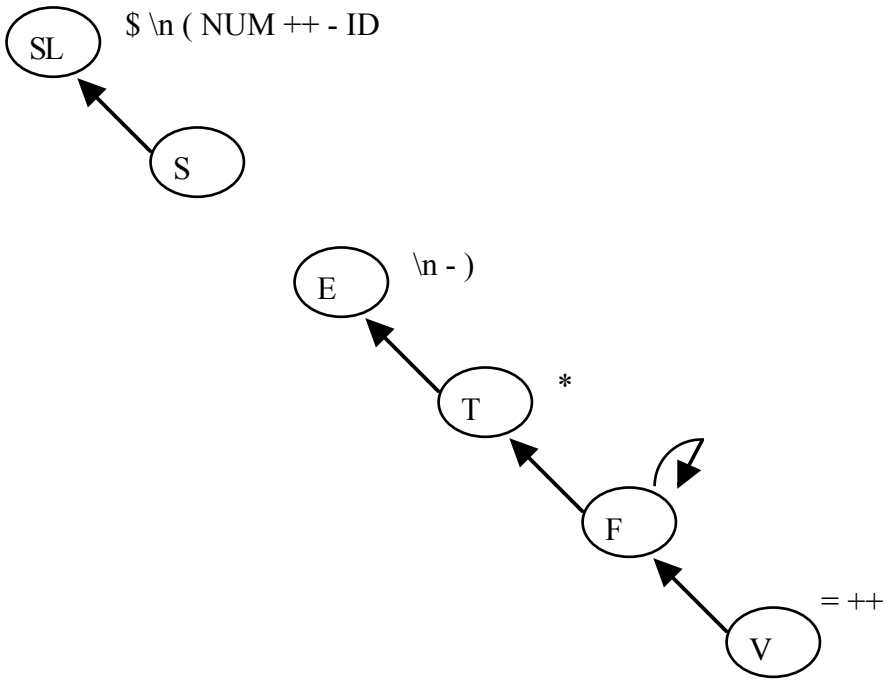


(c) (25 marks)

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	\n (NUM ++ - ID	$ \n (NUM ++ - ID
Stmt	\n (NUM ++ - ID	$ \n (NUM ++ - ID
Expr	(NUM ++ - ID	\n -)
Term	(NUM ++ - ID	\n -) *
Factor	(NUM ++ - ID	\n -) *
Variable	ID	\n -) * ++ =

(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)

```
{:
Integer value = ( Integer ) table.get( variable );
if ( value == null ) {
      parser.report_error( "Undeclared Variable " + variable,
      new Symbol( sym.IDENT, variableleft, variableright, variable ) );
      value = new Integer( 0 );
      }
RESULT = value;
value = new Integer( value.intValue() + 1 );
table.put( variable, value );
:}
```

(e) State 1 is …

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

lalr_state [3]: {

[Factor ::= Variable (*) , {NEWLINE MINUS TIMES }]

[Stmt ::= Variable (*) ASSIGN Expr NEWLINE , {EOF LEFT NEWLINE MINUS INC NUMBER IDENT }]

[Factor ::= Variable (*) INC , {NEWLINE MINUS TIMES }]

}

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

lalr_state [12]: {

[Variable ::= (*) IDENT , {NEWLINE MINUS TIMES }]

[Factor ::= INC (*) Variable , {NEWLINE MINUS TIMES }]

}

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.

```
MethodDecl ::=
            Type IDENT LEFT FormalParamDeclListOpt RIGHT
            LEFTBRACE BlockStmtList RIGHTBRACE
      |
            VOID IDENT LEFT FormalParamDeclListOpt RIGHT
            LEFTBRACE BlockStmtList RIGHTBRACE
      ;

FormalParamDeclListOpt::=
            FormalParamDeclList
      |
            /* Empty */
      ;

FormalParamDeclList::=
            FormalParamDecl
      |
            FormalParamDeclList
            SEMICOLON
            FormalParamDecl
      ;

FormalParamDecl::=
            Type IdentList
      ;

IdentList::=
            IDENT
      |
            IdentList COMMA IDENT
      ;
```

_____