4203 Compiler Theory (Final Exam)- Solution

Answer all of the following questions:

Question 1 (20 points)

1.1 Given the following LL (1) Grammar, where A is a non-terminal, a and b are terminals:

$A \rightarrow a\ A\ b \mid \varepsilon$

a) Construct First and Follow sets for the non-terminal A.

First(*A*)= {a, ε} Follow(*A*)= {$,b)

b) Construct the LL (1) parsing table .

M[N,T]	a	b	$
A	$A \rightarrow a\ A\ b$	$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$

c) Show the action of corresponding LL(1) parser given the input string aabb.

Step	Parsing Stack	Input	Action
1	*A*$	aabb$	$A \rightarrow a\ A\ b$
2	a *A* b$	aabb$	match
3	*A* b$	abb$	$A \rightarrow a\ A\ b$
4	a *A* b b$	abb$	match
5	*A* b b$	bb$	$A \rightarrow \varepsilon$
6	b b$	bb$	match
7	b$	b$	match
8	$	$	accept

d) For the above grammar, is it possible to have an equivalent regular expression? If yes, write the corresponding regular expression? If no, justify why

No RE cannot count

1.2 Draw a picture to illustrate the general organization of an activation record.

Space for arguments (parameters)
Space for bookkeeping information, including return address
Space for local data
Space for local temporaries

1.3 What happens on the stack at function call? What happens on the stack at function return?

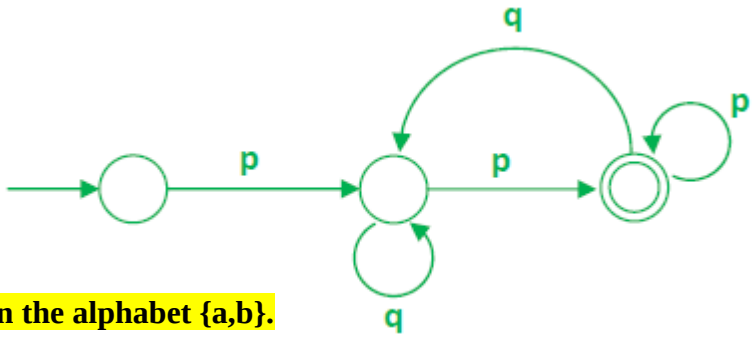
Calling Sequence

- Sequence of operations that must be done for procedure calls
  - Call sequence
    - Sequence of operations performed during procedure calls
      - Find the arguments and pass them to the callee.
      - Save the caller environment, i.e. local variables in activation records, return address.
      - Create the callee environment, i.e. local variables in activation records, callee's entry point.
  - Return sequence
    - Sequence of operations performed when return from procedure calls
      - Find the arguments and pass them back to the caller.
      - Free the callee environment.
      - Restore the caller environment, including PC.

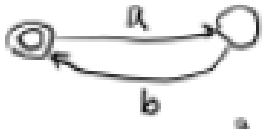
Question 2 (20 points)

2.1 Draw a DFA that accepts the following:

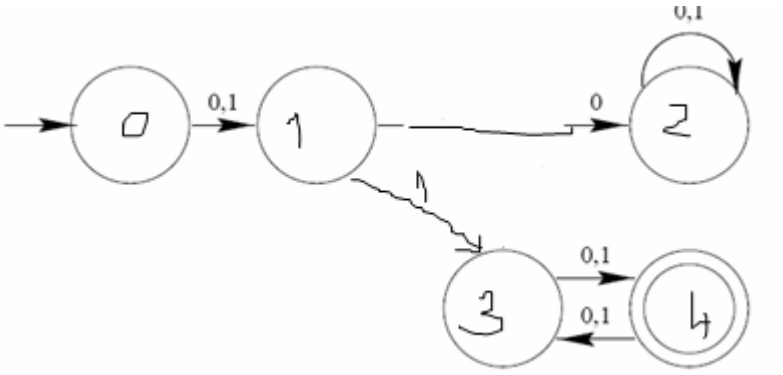
a) The regular expression p(p|q)*p, given the alphabet {p,q}.



b) The regular expression (ab)*, given the alphabet {a,b}.



c) Odd length strings of digits over the alphabet {0,1}, where the second digit is 1,and cannot contain strings of length less than 3



2.2 Given the following BNF grammar and associated semantic rules:

BNF grammar rules

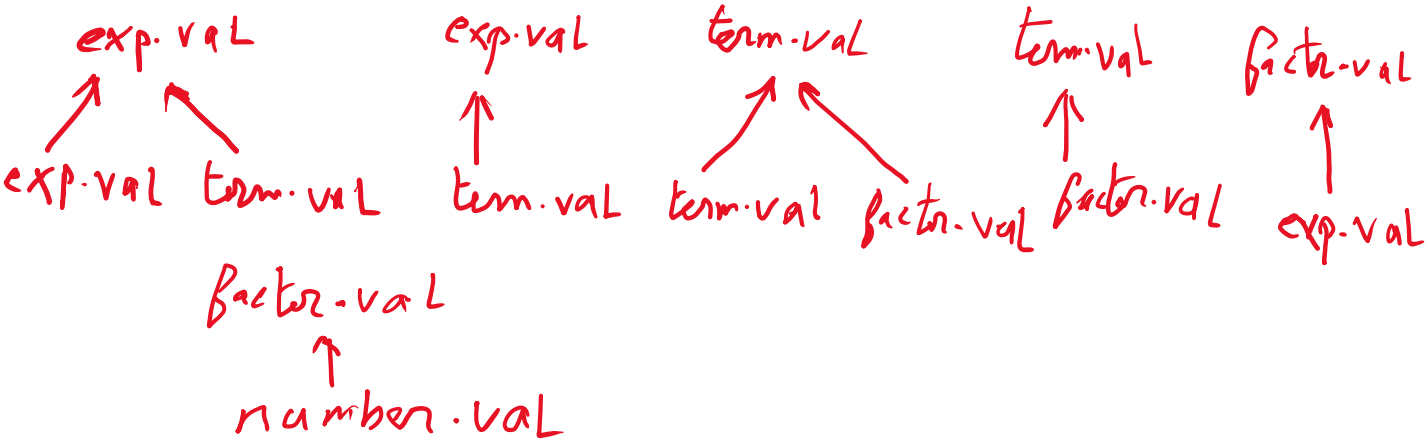
$exp \rightarrow exp + term \mid exp - term \mid term$
$term \rightarrow term * factor \mid factor$
$factor \rightarrow (exp) \mid \mathbf{number}$

Attribute Equations for the val attribute

Grammar Rule	Semantic Rules
$exp_1 \rightarrow exp_2 + term$	$exp_1.val = exp_2.val + term.val$
$exp_1 \rightarrow exp_2 - term$	$exp_1.val = exp_2.val - term.val$
$exp \rightarrow term$	$exp.val = term.val$
$term_1 \rightarrow term_2 * factor$	$term_1.val = term_2.val * factor.val$
$term \rightarrow factor$	$term.val = factor.val$
$factor \rightarrow (exp)$	$factor.val = exp.val$
$factor \rightarrow \mathbf{number}$	$factor.val = \mathbf{number}.val$

a) Draw the dependency graphs for the semantic rules

exp1=exp2.val + term.val
exp1= exp2.val-term.val
exp.val=term.val
term1.val=term2.val*factor.val
term.val=factor.val
factor.val= exp.val
factor.val=number.val



b) Draw the dependency graph for the string (34-3)*42

2.3 For the following piece of source code:

```
if (a < b + c)
      a = a - c;
c = b * c;
```

a) Write down the corresponding three-address code.

t1=b+c
t2= a < t1
if false t2 goto L1
t3=a-c
a=t3
Label L1
t4= b*c
c= t4

b) Show the quadruple representation for this code.

(0) (add, b , c)
(1) (lt, a, 0)
(2) (if_f, (1), (5))
(3) (sub, a,c)
(4) (asn, (3), a)
(5) (mul, b, c)
(6) (asn, (5), c)
(7) -------

Question 3 (20 points)

3.1 Construct regular expressions for:

a) An integral number is a non-zero sequence of digits, optionally followed by a letter denoting the base class (b for binary and o for octal).

***digit* = [0-9]**
***base* = b|o**
***integral_number* = *digit*+ *base*?**

b) All the strings that contain at least two 0's over the alphabet {0, 1}

(0|1)*0 (0|1)*0(0|1)*

3.2 State whether each of the following is true or false. If false, explain why.

a) S →A B
A → ε | aA
B →ε | bB
FIRST(S) contains 3 elements

FIRST(S) contains 3 elements.(T)
First(S)= { a,b, ε}

b) The following grammar is ambiguous(T).

S → ***S*a**
S*→b*S
***S*→c**

Leftmost: S → *S*a
→ b*S*a
→ bca
Another left most: S → b*S*
→ b*S*a
→ bca

b) The regular expressions a+a and a*a describe the same set of strings

The regular expressions a+a and a*a describe the same set of strings .(F)

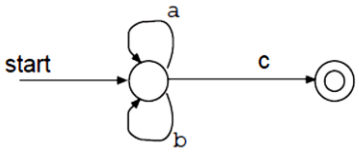
a+a; set of strings {aa,aaa,aaaa,…..}
a*a; set of strings {a,aaa,aaaa,…..}

d) A parser transforms a stream of characters into a stream of tokens (F)

A Scanner transforms a stream of characters into a stream of tokens

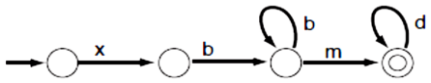
3.3 Write regular expressions that define the strings recognized by the DFAs shown below:

a)



(a|b)* c

b)



xb+ md*

Question 4 (20 points)

4.1 Given the following BNF grammar:

stmt-sequence → *stmt* ; *stmt-sequence* | *stmt*

stmt → s

a) Translate this Grammar into EBNF by using the Special Notation for *Optional Constructs*.

stmt-sequence →***stmt*** **[;*stmt-sequence*]**
stmt → **s**

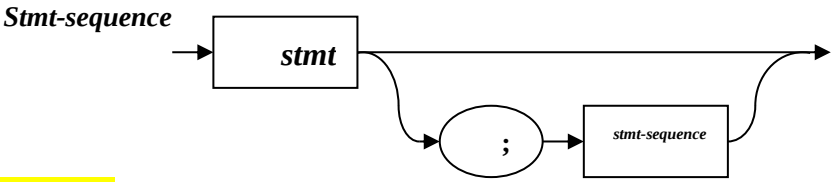
b) Write pseudo-code to parse EBNF grammar of part a) by recursive descent.

```
procedure stmt-sequence;
    begin
      stmt;
      if token = ; then
            match(;);
            stmt-sequence;
      end if;
    end stmt-sequence;
```

```
procedure stmt;
    begin
            match(s);
    end stmt;
```

```
procedure match( expectedToken);
begin
  if token = expectedToken then
    getToken;
  else
    error;
  end if;
end match
```

c) Draw the syntax diagrams of EBNF grammar of part a).



4.2 Given the following BNF grammar

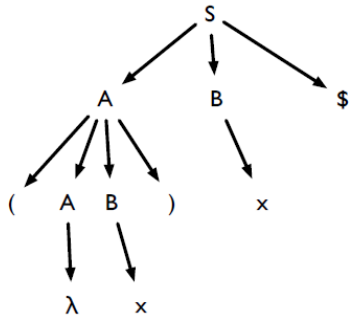
$A \rightarrow (AB)$
$A \rightarrow \varepsilon$
$B \rightarrow (A)$
$B \rightarrow x$

a) What are the terminals and non-terminals of this grammar?

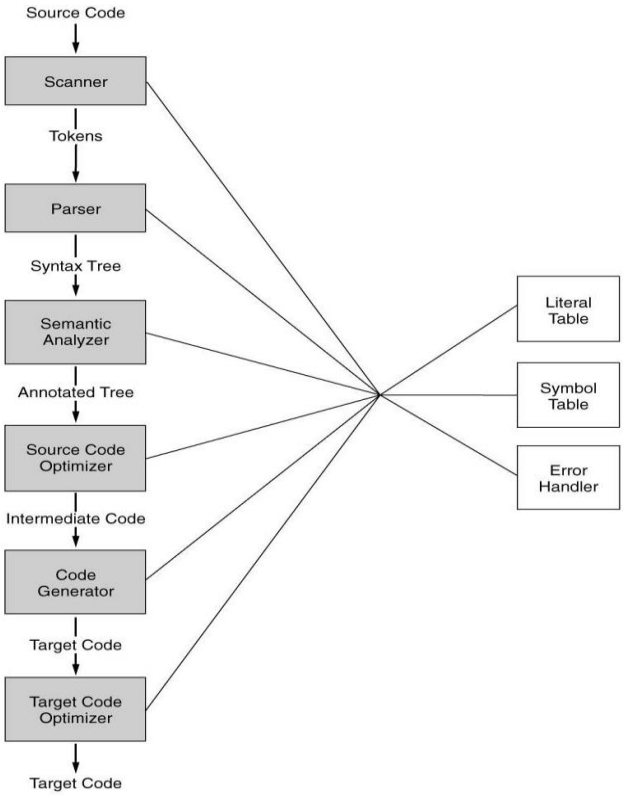
Terminals: ( ) x
Nonterminals: *A B*
Start: *A*

b) Construct a leftmost derivation and draw the parse tree for the following string: ((x)x).

***A* -> (*AB*) -> ((*AB*)*B*) -> ((*B*)*B*) -> ((x)*B*) -> ((x)x)**



4.3 Describe what phases normally are found in a compiler; what is their purpose; how they are connected; and what is their input and output.



4.4 Rewrite the following rule to avoid left recursion as required by LL(1) parser

$S \rightarrow Sx \mid y$
$S \rightarrow yS'$
$S' \rightarrow xS \mid \varepsilon$