1. A programming language can be defined as:

a. the set of symbols (or alphabet) that can be used to construct correct programs
b. the set of all correct programs
c. the 'meaning' of all correct programs
d. all of the above

---

2. If we use following notation: L0 > L1 > L2 > L3 , which language is indexed with 0:

a. unrestricted
b. context-sensitive
c. context-free
d. regular

---

3. Using the given notation and denoting the string of terminals as a sentence, then for a more general, so called 'sentential form', composed both from terminals and non-terminals will be used the symbol:

a. c
b. X
c. alpha
d. u

---

4. Grammar:

S	**->**	**SaBC|aBC**
Ca	**->**	**aC**
BC	**->**	**CB**
CB	**->**	**BC**
aB	**->**	**Ba**
Ba	**->**	**aB**
aC	**->**	**Ca**
B	**->**	**b**
C	**->**	**c**

is:

a. unrestricted
b. context-sensitive
c. context-free
d. regular

---

5. The same grammar as previously produces:

a. odd number of a's and b's, even number of c's
b. even number of a's and b's, odd number of c's
c. the same number of a's and c's, even number of b's
d. the same number of a,b and c's

---

6. Automaton with the transition table:

```
[{state}:       {input<a>}      |{input<b>}]
-----------------------------------------------------
[q0:            {q1,q2}         |{q0}]
[q1:            {q0,q1}         |{-}]
[{q0,q1}:       {q0,q1,q2}      |{q0}]
[{q1,q2}:       {q0,q1,q2}      |{q1}]
[{q0,q1,q2}:    {q0,q1,q2}      |{q0,g1}]
```

is:

a. not a finite state machine
b. non-deterministic (NFA)
c. deterministic (DFA)
d. both NFA and DFA

---

7. User's tokens are typically defined as:

a. text strings
b. integers > 255
c. pointers to the table of tokens
d. chars

---

8. Finite state automaton based lexical analyzer can be realized by:

a. single-state description (expansion) model
b. table-driven analyzer
c. LEX constructor
d. all of the above

---

9. Token returning function for lexical analysis using LEX is:
a. yytext
b. yylval
c. yylex
d. yyin

---

10. Finding an lexically ambiguous expression, LEX

a. does not worry about it, respectively handle it in an own way
b. reports an error and stops immediately
c. pass an ERROR token to the parser
d. special error routine must be implemented

---

11. Grammar:

S -> if b then S else S

S -> if b then S

S -> a

is:

a. empty
b. unambiguous
c. conditionally ambiguous
d. ambiguous

---

12. Which of the given sentences is generated by the following grammar:

S->aaB

B->RSb

R->SB|b

a. none
b. aa
c. aab
d. aabb

---

13. Replacing the rule

A->AA

with the rules

A->AB
B->A

the ambiguity from such grammar:

a. can be removed (depending on the rest of productions)
b. is removed definitely
c. cannot be removed
d. we cannot use this approach, because final languages will be different

---

14. Grammar with ε-rules is from the syntax analysis point of view:

a. completely inapplicable
b. always ambiguous
c. applicable only if ε-rule is connected with the grammar starting symbol
d. applicable only if ε-rule is not connected with the grammar starting symbol

---

15 Left-recursive rules:

a. must be always removed from the grammar before any parsing
b. must be always removed from the grammar before top-down parsing
c. cause significant parsing slow down, but are acceptable for every type of grammar
d. must be always removed from the grammar before bottom-up parsing

---

16 Top-down parses perform following operations:

a. expansion and comparison
b. shift and comparison
c. shift and reduction
d. expansion and reduction

---

17 Every LR(k) grammar is unambiguous.

a. true
b. false

---

18 Every unambiguous grammar is LR(k) for some k
a. true
b. false

---

19 In the following questions replace LX and/or LY with the appropriate type of parser(s). LX parser scans the input from left to right, seeing only the next token, without backtracking.

a. LL
b. LR
c. Both LR and LR
d. None of our known parsers

---

20 LX parsing is top-down, starting with the initial symbol and replacing single nonterminals subsequently.

a. LL
b. LR
c. Both LR and LR
d. None of our known parsers

---

21 LX parsing is bottom-up matching a sequence of tokens with the right hand side of a production rule and, when applicable, replacing them with the corresponding left hand side.

a. LL
b. LR
c. Both LR and LR
d. None of our known parsers

---

22 LX parse tables are constructed from lookahead sets indicating which production rule to apply for a non-terminal for a particular input token.

a. LL
b. LR
c. Both LR and LR
d. None of our known parsers

---

23 LX parse tables are constructed based on a corresponding finite analyzer of viable prefixes, indicating whether to shift the next token on to the stack or to reduce by a particular production rule.

a. LL
b. LR
c. Both LR and LR
d. None of our known parsers

---

24 LX parsers tend to have smaller parse tables.

a. LL
b. LR
c. Both LR and LR

---

25 LX parsers tend to be more efficient.

a. LL
b. LR
c. Both LR and LR
d. None of our known parsers

---

26 An LX parser recognizes a grammatical element as soon as it sees its first token. An LY parser recognizes the same based on the input token and stack history. LY parsers therefore cover wider class of grammars than LX parsers.

a. LX= LR, LY=LL
b. LX= LL, LY=LR

---

27 LX parsers cannot handle the left recursion.

a. LL
b. LR
c. Both LR and LR
d. None of our known parsers