SNS COLLEGE OF TECHNOLOGY

(An Autonomous Institution)
COIMBATORE – 35

DEPARTMENT OF COMPUTER SIENCE AND ENGINEERING (UG & PG)

Third Year Computer Science and Engineering, 5th Semester

2 Marks Question and Answer

Subject Code & Name: 19CSB301 / Automata Theory and Compiler Design

Prepared by: Dr.B.Vinodhini AP/CSE, Dr.M.Shobana ASP/CSE, Mrs.G.Swathi,AP/CSE

UNIT – I

1. Write any three applications of Automata Theory.

1. It is base for the formal languages and these formal languages are useful of the programming languages.
2. It plays an important role in complier design.
3. To prove the correctness of the program automata theory is used.
4. In switching theory and design and analysis of digital circuits automata theory is applied.
5. It deals with the design finite state machines.

2. Define Finite Automaton.

FA consists of a finite set of states and a set of transitions from state to state that Occur on input symbols chosen from an alphabet ∑.

A finite automata is a collection of 5 tuples (Q, Σ. δ, q_0, F) where Q is a finite set of states, which is non empty.

Σ is a input alphabet, indicates input set.

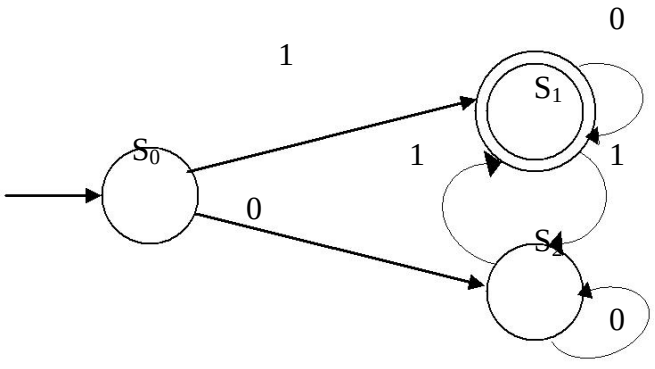
δ is a transition function or a function defined for going to next state. q_0 is an initial state (q_0 in Q)

F is a set of final states.

Two types:

Deterministic Finite Automation (DFA)

Non-Deterministic Finite Automation. (NFA)



3. Define Deterministic Finite Automation.

- The finite automata is called DFA if there is only one path for a specific input from current state to next state.
- A finite automata is a collection of 5 tuples (Q, Σ. δ, q_0, F)

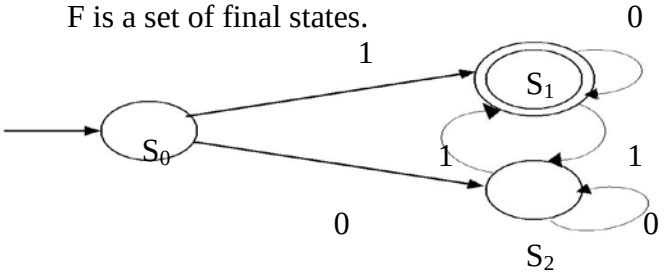
where Q is a finite set of states, which is non empty.

Σ is a input alphabet, indicates input set.

δ is a transition function or a function defined for going to next state.

q_0 is an initial state (q_0 in Q)

F is a set of final states.



4. Define Non-Deterministic Finite Automation.

The finite automaton is called NFA when there exists many paths for a specific input from current state to next state.

A finite automata is a collection of 5 tuples (Q, Σ. δ, q_0, F) where Q is a finite set of states, which is non empty.

Σ is a input alphabet, indicates input set.

δ is a transition function or a function defined for going to next state. q_0 is an initial state (q_0 in Q)

F is a set of final states.

5. Define NFA with ε transition.

The ε is a character used to indicate null string.

i.e the string which is used simply for transition from one state to other state without any input.

A Non Deterministic finite automata is a collection of 5 tuples (Q, Σ. δ, q_0, F)

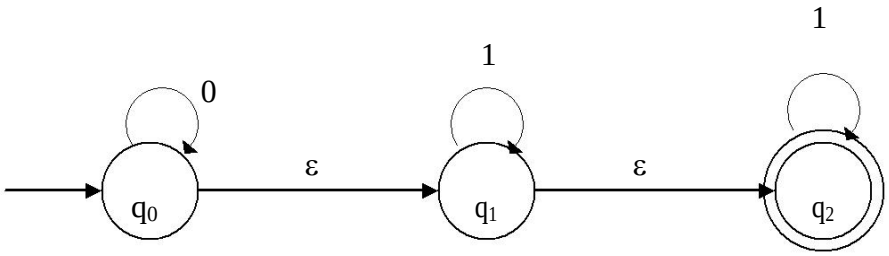
where Q is a finite set of states, which is non empty.

Σ is a input alphabet, indicates input set.

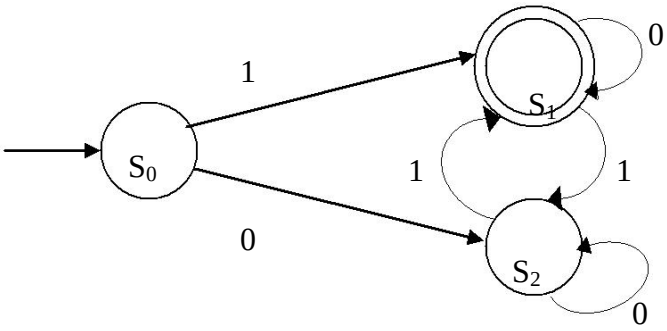
δ is a transition function or a function defined for going to next state.

q_0 is an initial state (q_0 in Q)

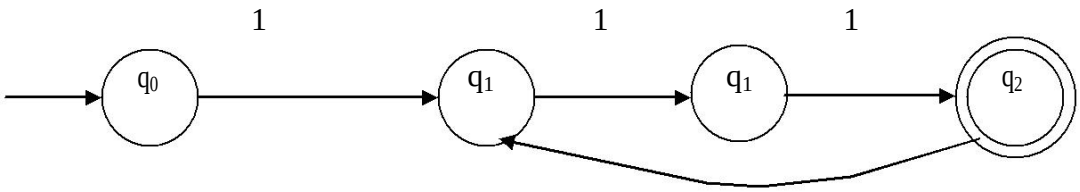
F is a set of final states.



6. Design FA which accepts odd number of 1's and any number of 0's.

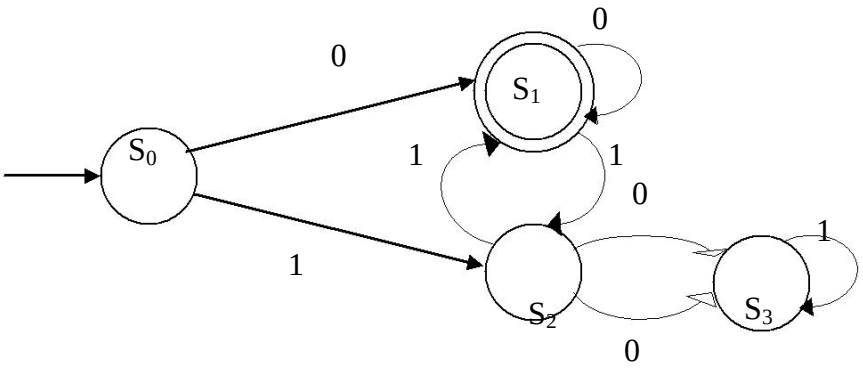


7. Design FA to check whether given unary number is divisible by three.

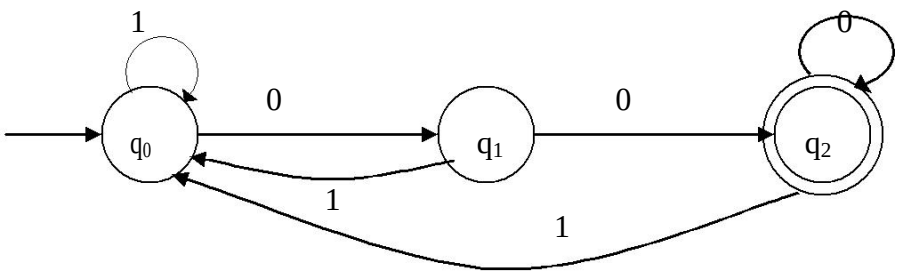


1

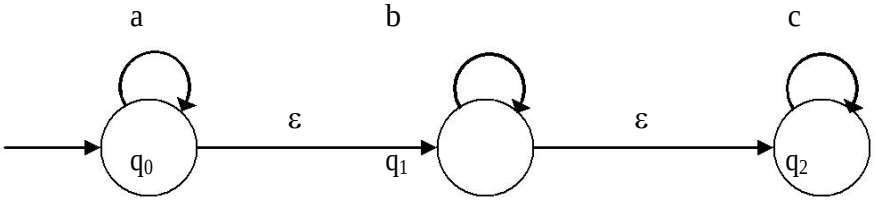
8. Design FA to check whether given binary number is divisible by three.



9. Design FA to accept the string that always ends with 00.



10. Obtain the ε-closure of states q0 and q1 in the following NFA with ε transition.



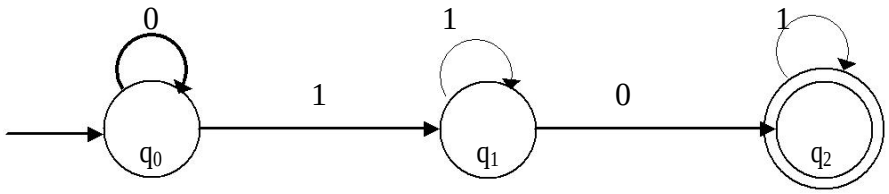
Solution:

ε-CLOSURE {q0} = {q0, q1,q2}
ε-CLOSURE {q1} = {q1,q2}

11. Explain a transition diagram.

It is a 5-tuple graph used state and edges represent the transitions from one state to other state.

Eg.:



12. Differentiate DFA and NFA?

Sl.No	DFA	NFA
1.	DFA is Deterministic Finite Automata	NFA is Non-Deterministic Finite Automata
2.	For given state, on a given input we reach to deterministic and unique state.	For given state, on a given input we reach to more than one state.
3.	DFA is a subset of NFA	Need to convert NFA to DFA in the design of complier.

13. Write short notes on Minimization of DFA?

a. Reducing the number of states from given FA
b. First find out which two states are equivalent we than replace those two states by

one representative state.

c. For finding the equivalent states we will apply the following rule

i. The two states S1 & S2 are equivalent if and only if both the states are final or non-final states.

14. What is automata theory?

It is the study of abstract computing devices or machines.

15. Define alphabet.

An alphabet is a finite, non-empty set of symbols, denoted by ∑.

Eg. ∑= {0, 1}, the binary alphabet

16. What do you mean by string?

A finite sequence of symbols chosen from some alphabet is called string. Eg. 01101 is a string from the binary alphabet ∑= {0, 1}

17. What is a empty string?

The empty string is the string with zero occurrences of symbols. It is denoted by ε.

18. Define length of a string.

Length of the string is "the number of symbols" in the string. |w|

E.g.; |101|=3, | ε|=0

19. Define language.

The language can be defined as a collection of strings or words over certain input set ∑*

20. Define set.

A set is a collection of objects.

E.g.: The collection of the four letters.

S= { a,b,c,d}

21. Define function.

A function is a rule that assigns to elements of one set a unique element of another set.

22. Define relation.

A relation is a set of pairs. The first component of each pair is chosen from a set called the domain and second component of each pair is chosen from a set called the range.

23. Define Abstract machines with e.g.

The computer which performs computation are not actual computers they are abstract machine.

E.g.: Finite Automata, Push down automata and Turing machine.

24. Give the examples/applications designed as finite state system.

. Text editors and lexical analyzers are designed as finite state systems. A lexical analyzer scans the symbols of a program to locate strings corresponding to identifiers, constants etc, and it

has to remember limited amount of information.

25. Define Finite State Systems.

The finite automaton is a mathematical model of a system, with discrete inputs and outputs and a finite number of memory configuration called states.

26.What are the types of Finite automata?

Deterministic finite automata (DFA)
Non deterministic finite automata (NFA)

27.Define DFA.

A DFA consists of A=(Q,∑,δ,q0,F)
Where, Q is the finite set of states,
∑ is a finite input alphabet,
q0 in Q is the initial state,
F is the set of final states and δ is the transition mapping function
Q * Σ to Q.

28. Define NFA.

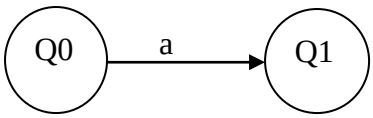
A NFA consists of A=(Q,∑,δ,q0,F)
Where, Q is the finite set of states ,
∑ is a finite input alphabet,
q0 in Q is the initial state,
F is the set of final states and δ is the transition mapping function
δ: Q x (∑U{ ε}) = 2^Q

29.What are the notations for DFA/NFA?

Transition diagram
Transition table

30. Define Transition diagram.

Transition diagram is a directed graph in which the vertices of the graph Correspond to the states of FA. If there is a transition from state q to state p on input, then there is an arc labeled ‘ a ‘ from q to p in the transition diagram.



31. Define transition table.

It is a conventional tabular representation of a function. The rows of the table correspond to the states and the columns correspond to the inputs.

32. What are the applications of automata theory?

- In compiler construction.
- In switching theory and design of digital circuits.
- To verify the correctness of a program.
- Design and analysis of complex software and hardware systems.
- To design finite state machines such as Moore and mealy machines.

33. What are the components of Finite automaton model?

The components of FA model are Input tape, Read control and finite control.

(a)The input tape is divided into number of cells. Each cell can hold one i/p symbol.

(b)The read head reads one symbol at a time and moves ahead.

(c)Finite control acts like a CPU. Depending on the current state and input Symbol read from the input tape it changes state.

34. Define extended transition function for DFA and NFA.

DFA: If δ is the transition function then the extended transition function is δ'.

It takes a state q and a string w and returns a state p.

i.e $\delta'(q,w)= \delta(\delta'(q,x),a)$

NFA: If δ is the transition function then the extended transition function is δ'.

It takes a state q and a string w and returns a set of states.

$$\bigcup_{i=1}^{k} \delta(p_i ,a)=\{r1,r2,r3\ldots..\}$$

35. Define language of DFA and NFA.

The language of a DFA A= (Q, ∑, δ,q0,F). This language is denoted by L (A) and defined as L (A) = {w/ δ' (q0,w) is in F}

The language of a NFA A=(Q, ∑, δ,q0,F). This language is denoted by L(A) and defined as L(A) = {w/ δ'(q0,w) n F≠Φ}

36. What is ε -closure of a state q0?

ε -closure (q0) denotes a set of all vertices p such that there is a path from q0 to Labeled ε.

37. What is a Regular Language?

The language accepted by M is L(M) is the set {x | δ(q0,x) is in F}. A language is regular if it is accepted by some finite automaton.

38. Define star closure and positive closure.

Star closure –zero or more Occurrence

Positive closure – one or more occurrence

(eg) a* = { ε,a,aa,….}

a+ = {a,aa,….}

L+ = L* - {ε}

39. Define NFA with ε-transition. Is the NFA's with ε-transitions are more powerful than The NFA's without ε -transition?

The NFA with €- moves defined by 5tuple or quintuple as similarly as NFA, Except δ (Q,∑, δ,qo,F) with all components as before and δ: $Qx(\sum U\{ \varepsilon\}) = 2^Q$ No, NFA with ε-transition and NFA without ε-transition have the same power.

40. Is it true the language accepted by NFA is different from the regular language?

Justify. Your answer.

No, it is false. For every regular expression r there exists a NFA with ε-transition that accepts L(r).

UNIT II

1. Define compiler?

A compiler is a program that reads a program written in one language (source language) and translates it into an equivalent program in another language (target language) and the compiler reports to its user the presence of errors in the source program.

2. What are the classifications of compiler?

i) Single pass compiler
ii) Multi pass compiler
iii) Load-and-go compiler
iv) Debugging or optimizing compiler

3. What are the phases of compiler?

i) Lexical analyzer
ii) Syntax analyzer
iii) Semantic analyzer
iv) Intermediate code generation
v) Code generation
vi) Code optimization
vii) Symbol table manager.

4. Define preprocessor & what are the functions of preprocessor?

Preprocessor produce input to the compilers (i.e.) the program will be divided in to the modules. They may perform the following functions.

i) Macro processing
ii) File inclusion
iii) Rational preprocessor
iv) Language extension

5. What are the tools available in analysis phase?

i) Structure editors
ii) Pretty printer
iii) Static checkers
iv) Interpreters.

6. Define pretty printers?

A pretty printer analyzes a program and prints it in such a way that the structure of the program becomes clearly visible. For the comments may appear with an amount of indentation proportional to the depth of their nesting in the hierarchical organization of the statements.

7. Define assembler and its types?

It is defined by the low level language is assembly language and high level language is machine language is called assembler.

- One pass assembler
- Two pass assembler

8. Give the types of a language processing system?

a) Preprocessors
b) Compilers
c) Assembler
d) Loaders and link editors

9. What are the functions performed in analysis phase?

a. Lexical analysis or Linear analysis
b. Syntax analysis or hierarchical analysis
c. Semantic analysis

10. **What are the functions performed in synthesis phase?**

i) Intermediate code generation
ii) Code generation
iii) Code optimization

11. **Give the classification of processing performed by the semantic analysis?**

a) Processing of declarative statements.
b) Processing of executable statements.

12. Give the properties of intermediate representation?

a) It should be easy to produce.
b) It should be easy to translate into the target program.

13. **What are the two different parts of compilation?**

a) Analysis phases of compilation
b) Synthesis phases of compilation

14. **What is meant by lexical analysis?**
It reads the characters in the program and groups them into tokens that are sequences of characters having a collective meaning. Such as an identifier, a keyword, a punctuation, character or a multi-character operator like ++.

15. What is meant by syntax analysis?
It processes the string of descriptors, synthesized by the lexical analyzer, to determine the syntactic structure of an input statement. This process is known as parsing. Output of the parsing step is a representation of the syntactic structure of a statement. It representation in the form of syntax tree.

16. **What is meant by intermediate code generation?**
After syntax and semantic analysis, some compilers generate an explicit intermediate representation of the source program. It can have a variety of forms. This form called three-address code. It consists of sequence of instructions, each of which has at most three operands.

17. **What is meant by semantic analysis?**
This phase checks the source program for semantic errors and gathers type of information for the subsequent phase.

18. **What do you meant by interpreter?**
Certain other translators transform a programming language into a simplified language called intermediate code, which can directly executed using a program called an interpreter.

19. **What do you meant by phases?**
Each of which transforms the source program one representation to another. A phase is a logically cohesive operation that takes as input one representation of the source program and produces as output another representation

20. **Write short notes on symbol table manager?**
The table management or bookkeeping portion of the compiler keeps track of the names used by program and records essential information about each, such as its type (int,real etc.,) the data structure used to record this information is called a symbol table manger.
21. **Write short notes on error handler?**
The error handler is invoked when a flaw in the source program is detected. It must warn the programmer by issuing a diagnostic, and adjust the information being passed from phase to phase so that each phase can proceed. So that as many errors as possible can be detected in one compilation.

22. Mention some of the cousins of the compiler?

i) Preprocessors
ii) Assemblers
iii) Two pass assembly
iv) Loaders and Linker-editors.

23. **What is front end and back end?**

The phases are collected into a front end and a back end. The front end consists of those phases or parts of phases, that depends primarily on the source language and is largely independent of the target machine. The back ends that depend on the target machine and generally these portions do not depend on the source language.

24. **What do you meant by passes?**

A pass reads the source program or the output of the previous pass, makes the transformations specified by its phases and writes output into an intermediate file, which may then be read by a subsequent pass. In an implementation of a compiler, portions of one or more phases are combined into a module called pass.

25. **List some compiler construction tools?**

i) Parser generators
ii) Scanner generators
iii) Syntax-directed translation engine
iv) Automatic code generators
v) Data-flow engine.

26. Explain any one compiler construction tool?

Scanner generators, these automatically generate lexical analyzers normally from a specification based on regular expressions. The resulting of lexical analyzer is in effect of finite automata.

27. What are issues available in lexical analysis?

i) Simpler design
ii) Compiler efficiency is improved by specialized buffering techniques for reading input characters and processing tokens and significantly speeds up the performance of a compiler.
iii) Compiler portability is enhanced.

28. **Define patterns/lexeme/tokens?**

A set of strings in the input for which the same token is produced as output. This set of strings described by a rule called **pattern** associated with the token.

A lexeme is a sequence of characters in the source program that is matched by the pattern for a token.
Token is a sequence of character that can be treated as a single logical entity.

29. Give the algebraic properties of regular expression?

	AXIOM	DESCRIPTION
i)	r/s = s/r	/ is commutative
ii)	r/(s/t)=(r/s)/t	/ is associative
iii)	(rs)t=r(st)	concatenation is associative
iv)	r(s/t)=rs/rt	concatenation distributes over /
v)	r**=r*	* is idempotent

30. **What are the systems referred to data flow engine?**

i) Compiler-compilers
ii) Compiler-generators
iii) Translator writing systems.

31. **Give the error recovery actions in lexical errors?**

i) Deleting an extraneous character
ii) Inserting a missing character
iii) Replacing an incorrect character by a correct character.

UNIT III

1. What do u meant by parser and its types?

A parser for grammar G is a program that takes as input a string ‘w’ and produces as output either a parse tree for’w’, if ‘w’ is a sentence of G, or an error message indicating that w is not a sentence of G. it obtains a string of tokens from the lexical analyzer, verifies that the string generated by the grammar for the source language.

a) Top down parsing
b) Bottom up parsing

2. What are the different levels of syntax error handler?

a) Lexical, such as misspelling an identifier, keyword, or operator.
b) Syntactic, such as an arithmetic expression with unbalanced parentheses
c) Semantic, such as operator applied to an incompatible operand
d) Logical, such as an infinitely recursive call.

3. What are the goals of error handler in a parser?

i) It should report the presence of errors clearly and accurately

ii) It should recover from each error quickly enough to be able to detect subsequent errors

iii) It should not significantly slow down the processing of correct programs.

4. What are error recovery strategies in parser?

a) Panic mode
b) Phrase level
c) Error productions
d) Global corrections

5. Define CFG?

Many programming language has rules that prescribe the syntactic structure of well-formed programs. The syntax of programming language constructs can be described by CFG. Conditional statement defined by a rule such as; If S1 and S2 are statements and E is an expression, then

"If E then S1 else S2" is a statement.

6. Define derivations. Give an example and its types?

We apply the productions of a CFG to infer that certain strings are in the language of a certain variable. There are two approaches (a) derivations (b) recursive inference or reduction. Then derivation uses the production from head to body. A replacement according to a production is known as as derivation

i) Left most derivation

ii) Right most derivation or canonical derivations

E→ E+E→id+E→id+(E)
→ id+(E+E)→id+(id*E)
→ id+(id*id)

7. Define ambiguity?

A grammar that produces more than one parse tree for some sentences is said to be ambiguous.

8. Define sentential form?

If G = (V, T, P, S) is a CFG, then any string 'α' in (VUT)* such that S→* α is a sentential form.

9. Define yield of the string?

A string that is derived from the root variable is called the yield of the tree.

10. Give the several reasons for writing a grammar?

a) The lexical rules of a language are frequently quite simple and to describe them we do not need a notation as powerful as grammars.
b) R.E generally provides a more concise and easier to understand notation for token than grammars.
c) More efficient lexical analyzers can be constructed automatically from R.E than from arbitrary grammars.
d) Separating the syntactic structure of a language into lexical and nonlexical parts provides a convenient way of modularizing the front end of a compiler into two manageable-sized components.

11. Define left factoring?

The process of factoring out of common prefixes of alternates is called as left factoring.

12. What are the difficulties with top down parsing?

a) Left recursion
b) Backtracking
c) The order in which alternates are tried can affect the language accepted
d) When failure is reported. We have very little idea where the error actually occurred.

13. What is meant by recursive-descent parser?

A parser that uses a set of recursive procedures to recognize its input with no backtracking is called a recursive-descent parser. To avoid the necessity of a recursive language, we shall also consider a tabular implementation of recursive descent called predictive parsing.

14. Define top down parsing?

It can be viewed as an attempt to find the left most derivation for an input string. It can be viewed as attempting to construct a parse tree for the input starting from the root and creating the nodes of the parse tree in preorder.

15. Define LL (1) grammar?

A grammar G is LL (1) if and only if, whenever A→ α / β are two distinct productions of G of the following conditions

a) For no terminal 'a' do both α and β derive strings beginning with α.
b) At most one of α and β can derive the empty string.
c) If β*→ e then α does not derive any string beginning with a terminal in FOLLOW (A).

16. What are the possibilities of non-recursive predictive parsing?

a) If X = a = $ the parser halts and announces successful completion of parsing
b) If X = a = $ the parser pops X off the stack and advances the input pointer to the next symbol
c) If X is a nonterminal, the program consults entry M[X,a] of the parsing table M. this entry will be either an X-production of the grammar or an error entry.

17. What are the actions available in shift reduce parser?

a) Shift
b) Reduce
c) Accept
d) Error

18. Define handle?

A handle of a string is a substring that matches the right side of a production, and whose reduction to the non-terminal on the left side of the production represents one step along the reverse of a right most derivation.

19. Define viable prefixes?

The set of prefixes of right sentential forms that can appear on the stack of a shift reduce parser are called viable prefixes.

20. What are the two common ways of determining precedence relations should hold between a pair of terminals?

a) Based on associative and precedence of operators.
b) Construct an unambiguous grammar for the language, a grammar that reflects the correct associativity and precedence in its parse tree.

21. Define operator grammar?

A grammar with the property that no production right side is £ or has two adjacent nonterminals are called an operator grammar.

22. Define LR parser?

LR parsers can be used to parse a large class of context free grammars. The technique is called LR (K) parsing.
"L" denotes that input sequence is processed from left to right
"R" denotes that the right most derivation is performed
"K" denotes that at most K symbols of the sequence are used to make a decision.

23. What are the drawbacks of LR parser?

a) Parsing tables are too complicated to be generated by hand, need an automated parser generator.
b) Cannot handle ambiguous grammar without special tricks.

24. Give the reasons for LR parser?

a) LR parsers can handle a large class of CF languages
b) An LR parser can detect syntax errors as soon as they occur
c) The LR parsing method is the most general non-back tracking shift reduce parsing method
d) LR parsers can handle all language recognizable by LL(1).

25. What are the techniques for producing LR parsing Table?

1) Shift s, where s is a state
2) Reduce by a grammar production $A \rightarrow \beta$
3) Accept and
4) Error

26. Define augmented grammar?

If G is a grammar with start symbol S, then G' the augmented grammar for G, is G with a new start symbol S' and production S'→S. the purpose of this new starting production is to indicate to the parser when it should stop parsing and announce acceptance of the input.

27. Define LR (0) items?

LR (0) item for a grammar G is a production of G with a dot at some position of the right side. Thus production A→XYZ yields the four items.

A→.XYZ, A→X.YZ, A→XY.Z, A→XYZ.

28. What are the two functions of LR parsing algorithm?

a) Action function
b) GOTO function

29. What are the three types of YACC?

a) Declarations
b) Translation rules
c) supporting c-routines

30. Give note about semantic actions?

It is a sequence of c statements. In a semantic action, the symbol $$ refers to the attribute value associated with the NT on the left, and $i refers to the value associated with the i^{th} grammar symbol on the right. It is called semantic action.

31. Define SLR parser?

The parsing table consisting of the parsing action and goto function determined by constructing an SLR parsing table algorithm is called SLR(1) table. An LR parser using the SLR (1) table is called SLR (1) parser. A grammar having an SLR (1) parsing table is called SLR (1) grammar.

32. Give the two optional sections in the declarations parts?

a) Ordinary 'c' declarations
   Delimited by % {and %}, a temporary variables used by the translation rules or procedures.
b) Declarations of grammar tokens.

33. What are two classes of items in LR parser?

a) Kernel items, which include the initial item, S'→.S, and all items whose dots are not at the left end.
b) Non-Kernel items, which have their dots at the left end.

34. What are the three techniques for constructing LR parsing table?

a) SLR (simple LR)
b) Canonical LR
c) LALR (Look ahead LR)

35. Define bottom up parsing?

It attempts to construct a parse tree for an input string is beginning at leaves and working up towards the root (i.e.) reducing a string 'w' to the start symbol of a grammar. At each reduction step, a particular substring matching the right side of a production is replaced by the symbol on the left of that production. It is a rightmost derivation and it's also known as shifts reduce parsing.

36. Define LALR grammar?

Last parser construction method, the LALR technique. This method is often used in practice because the tables obtained by it are considerably smaller than the canonical LR tables, yet most common syntactic constructs of programming language can be expressed conveniently by an LALR grammar. If there are no parsing action conflicts, then the given grammar is said to be an LALR (1) grammar. The collection of sets of items constructed is called LALR (1) collections.

37. Define operator precedence grammar?

It is an £-free operator grammar in which the precedence relations <., =., and .> constructed as above are disjoint. That is for any pair of terminals a and b, never more than one of the relations a<.b, a=.b, and a.>b is true.

38. Define handle pruning?

Reducing β to A in αβw is called handle pruning, i.e., removing the children of A from the parse tree.

39. Eliminate left recursion from the grammar.

S→ Aa / b, A→Ac / Sd/e?

Replaced as A→bdA' / A'
A'→cA'/ adA'/ £

Finally we obtain

S→Aa / b,
A→bdA' / A',
A'→cA' / adA' / £

40. Define left recursion. Give an example?

A grammar is left recursive if it has a nonterminal 'A' such that there is a derivation A→+ Aα for some strings α.

Example: Pair of productions A→Aα/β.

Replaced by nonterminal productions
A→βA' and A→αA'€E

UNIT IV

1. What are the advantages of generating an intermediate representation?

i) Ease of conversion from the source program to the intermediate code.
ii) Ease with which subsequent processing can be performed from the intermediate code.

2. Define a syntax-directed translation?

Syntax-directed translation specifies the translation of a construct in terms of Attributes associated with its syntactic components. Syntax-directed translation uses a context free grammar to specify the syntactic structure of the input. It is an input-output mapping.

3. Define an attribute. Give the types of an attribute?

An attribute may represent any quantity, with each grammar symbol, it associates a set of attributes and with each production, a set of semantic rules for computing values of the attributes associated with the symbols appearing in that production.

Example: a type, a value, a memory location etc.,

i) Synthesized attributes.
ii) Inherited attributes.

4. Define annotated parse tree?

A parse tree showing the values of attributes at each node is called an annotated parse tree. The process of computing an attribute values at the nodes is called annotating parse tree.

Example: an annotated parse tree for the input 3*5+4n.

5. Define dependency graph?

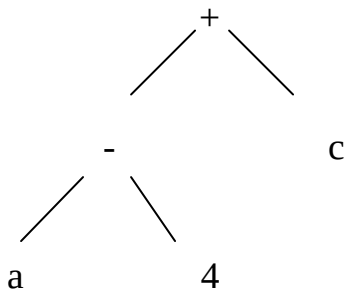
The interdependencies among the inherited and synthesized attributes at the nodes in a parse tree can be depicted by a directed graph is called a dependency graph.

Example: Production E→ E1 + E2
Semantic Rule E.val:= E1.va; + E2.val

6. Define syntax tree. Give an example?

An (abstract) syntax tree is a condensed form of parse tree useful for representing language constructs.

Example: syntax tree for a – 4 + c



7. What are the functions used to create the nodes of syntax trees?

i) Mknode (op, left, right)
ii) Mkleaf (id,entry)
iii) Mkleaf (num, val)

8. What are the functions for constructing syntax trees for expressions?

i) The construction of a syntax tree for an expression is similar to the translation of the expression into postfix form.
ii) Each node in a syntax tree can be implemented as a record with several fields.

9. Define DAG. Give an example?

DAG is a directed acyclic graph for an expression identifies the common sub expression in the expression.

Example: DAG for the expression a- 4 *c

P1 = mkleaf(id,a) P2 = mknum(num,4)
P3 = mkleaf(id,c) P4 = mknode('*',p2,p3)
P5 = mknode('-',p1,p4)

10.What are the three kinds of intermediate representations?

i) Syntax trees.
ii) Postfix notation.
iii) Three address code.

11.Define postfix notation?

Postfix notation is a linearized representation of a syntax tree. It is a list of the nodes of the tree in which a node appears immediately after its children.
The syntax tree is, a: = b*-c
The postfix notation for the syntax tree is, abc-*c

12.Define three-address code?

Three address code is a sequence of statements of the form x: = y op z. where x, y, z are names, constants, or compiler generated temporaries, op stand for any type of operator. Since a statement involves not more than three references it is called three-address statement, and hence a sequence of such statement is called three address codes.

13.What are the types of three address statements?

Assignment statements, assignment instruction, copy statements, conditional jump, address-address statements, indexed assignment statements, address and pointer statements.

14.What are the three types of implementations of three-address statements?

i) Quadruples
ii) Triples
iii) Indirect Triples.

15.What are the methods of translating Boolean expressions?

There are two principal methods of representing the value of a Boolean expression.

a) Encode true and false numerically and to evaluate a Boolean expression analogous to an arithmetic expression.
b) Flow-of –control. Represent the value of a Boolean expression by a position reached in a program.

16. What are the two purposes of Boolean expressions?

a) They are used to compute logical expressions.
b) Often they are used as condition expression in statements that alter the flow of control, such as if-then, if-then-else, or while-do statements.

17. Define quadruple. Give an example?

A quadruple is a record structure with four fields: op, arg1, arg2 and result. The op field contains an internal code for the operator.

Example: x: =y op z

18. Give the advantages of quadruples?

i) Can perform peephole optimization.
ii) The contents of field's arg1, arg2 and result are normally pointers
iii) The symbol-table entries for the names represented by these fields.
iv) If So, temporary names must be entered into the symbol table as they Are created.

19.Define triple. Give an example?

Triple is a record structure with three fields: op, arg1 and arg2. The fields arg1 and arg2 are either pointes to the symbol-table or pointers into the triple structure. This method is used to avoid temporary names into the symbol table.

20.Define indirect triples. Give the advantage?

Listing pointers to triples rather than listing the triples themselves are called indirect triples.

Advantages: it can save some space compared with quadruples, if the same temporary value is used more than once.

21. Define translation scheme?

A translation scheme is a CFG in which program fragments called semantic action are embedded within the right sides of productions. A translation scheme is like a syntax-directed definition, except that the order of evaluation of the semantic rules is explicitly shown.

22.What are the three address code for a or b and not c?

The three address sequence is

T1:= not c
T2:= b and T1
T3:= a or T2.

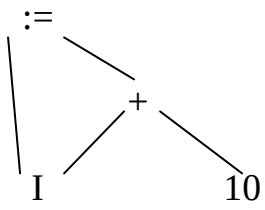
23.Write a three address code for the expression a < b or c < d?

100: if a<b goto 103
101: t1:=0
102: goto 104
103: t1:=1
104: if c<d goto 107
105: t2:=0
106: goto 108
107: t2:=1
108: t3:=t1 or t2

24.What are the parameter transmission mechanisms?

1. Call by value
2. Call by value-result
3. Call by reference
4. Call by name

25.Construct a DAG for the expression I: = I + 10?



26.What are the various data structure used for implementing the symbol table?

1. Linear list
2. Binary tree
3. Hash table

27. What is the purpose of DAG?

i) A label for each node. For leaves the label is an identifier and for interior nodes, an operator symbol
ii) For each node a list of attached identifiers.

28.Define backpatching?

It constructs the syntax tree for the input, and then walks the tree in depth-first order. Backpatching can be used to generate code for Boolean expressions and flow-of-control statements in a single pass is that during one single pass we may not know the labels that control must go to at the time the jump statements are generated.

29.What are the three functions of backpatching?

i) Makelist(i) – create a new list.
ii) Merge(p1,p2) – concatenates the lists pointed to by p1 and p2.
iii) Backpatch(p,i) – insert i as the target label for the statements pointed to by p.

30.Give short note about call-by-name?

Call by name, at every reference to a formal parameter in a procedure body the name of the corresponding actual parameter is evaluated. Access is then made to the effective parameter.

31.How parameters are passed to procedures in call-by-value method?

This mechanism transmits values of the parameters of call to the called program. The transfer is one way only and therefore the only way to returned can be the value of a function.

```
Main ( )
{ print (5);    }
  Int
        Void print (int n)
        { printf (“%d”, n);  }
```

32. Define symbol table?

A compiler uses a symbol-table to keep track of scope and binding information about names. It is searched every time a name is encountered in the source text changes to the table occur, if a new name or new information about an existing name is discovered.

33.What are the semantic rules are defined in the declarations operations?

1) Mktable(previous)
2) Enter(table, name,type,offset)
3) Addwidth(table,width)
4) Enterproc(table,namenewtable)

34.Define short circuit code?

Translate the Boolean expression into three-address code without generating code for any of the Boolean operators and without having the code necessarily evaluate the entire expression. This style of evaluation is sometimes is called short-circuit or jumping code.

35.Give the syntax of case statements?

```
Switch expression
      Begin
```

```
        Case value:   statement
        Case value:   statement
          -------
        Case value:   statement
        Default   :   statement
End
```

36. Give the 2 attributes of syntax directed translation into 3-addr code?

i) E.place, the name that will hold the value of E and
ii) E.code , the sequence of 3-addr statements evaluating E.

37. Write a short note on declarations?

Declarations in a procedure, for each local name, we create a symbol table entry with information like the type and the relative address of the storage for the name. The relative address consists of an offset from the base of the static data area or the field for local data in an F record. The procedure enter (name, type, offset) create a symbol table entry.

38. Give the two parts of basic hashing scheme?

1) A hash table consisting of a fixed array of m pointers to table entries.
2) Table entries organized into m separate linked lists, called buckets. Each record in the symbol table appears on exactly one of these lists.

39. Write the grammar for flow-of-control statements?

The following grammar generates the flow-of-control statements, if-then, if-then-else, and while-do statements.

S → if E then S1
| If E then S1 else S2
| While E do S1.

40. Write the 3-addr code for the statements a =b*-c + b*-c?

Three address codes are: a=b*-c + b*-c

T1 = -c
T2 = b*T1
T3 = -c
T4 = b*T3
T5 = T2+T4
a:= T5.

UNIT V

1. Define code generations with ex?

It is the final phase in compiler model and it takes as an input an intermediate representation of the source program and output produces as equivalent target programs. Then intermediate instructions are each translated into a sequence of machine instructions that perform the same task.

2. What are the issues in the design of code generator?

- Input to the generator
- Target programs
- Memory management
- Instruction selection
- Register allocation
- Choice of evaluation order
- Approaches to code generation.

3. **Give the variety of forms in target program**.

- Absolute machine language.
- Relocatable machine language.
- Assembly language.

Give the factors of instruction selections.

- Uniformity and completeness of the instruction sets
- Instruction speed and machine idioms
- Size of the instruction sets.

5. **What are the sub problems in register allocation strategies?**

- During register allocation, we select the set of variables that will reside in register at a point in the program.
- During a subsequent register assignment phase, we pick the specific register that a variable reside in.

6. Give the standard storage allocation strategies.

- Static allocation
- Stack allocation.

7. Define static allocations and stack allocations

Static allocation is defined as lays out for all data objects at compile time.

Names are bound to storage as a program is compiled, so there is no need for a Run time support package.

Stack allocation is defined as process in which manages the run time as a Stack. It is based on the idea of a control stack; storage is organized as a stack, And activation records are pushed and popped as activations begin and end.

8. **Write the addressing mode and associated costs in the target machine.**

MODE	FORM	ADDRESS	ADDED COST
Absolute	M	M	1
Register	R	R	0
Indexed	c(R)	c+contents(R)	1
Indirect register	*R	contents(R)	0
Indirect indexed	*c(R)	contents(c+contents(R))	1

9. **Define basic block and flow graph.**

A basic block is a sequence of consecutive statements in which flow of Control enters at the beginning and leaves at the end without halt or possibility Of branching except at the end.

A flow graph is defined as the adding of flow of control information to the Set of basic blocks making up a program by constructing a directed graph.

10. Write the step to partition a sequence of 3 address statements into basic blocks.

1. First determine the set of leaders, the first statement of basic blocks.
   - The rules we can use are the following.
   - The first statement is a leader.
   - Any statement that is the target of a conditional or unconditional goto is a leader.
   - Any statement that immediately follows a goto or conditional goto statement is a leader.
2. For each leader, its basic blocks consists of the leader and all statements Up to but not including the next leader or the end of the program.

11. Give the important classes of local transformations on basic blocks

- Structure preservation transformations
- Algebraic transformations.

12. Describe algebraic transformations.

It can be used to change the set of expressions computed by a basic blocks into A algebraically equivalent sets. The useful ones are those that simplify the Expressions place expensive operations by cheaper ones.

X = X+ 0
X = X * 1

13. What is meant by register descriptors and address descriptors?

A register descriptor keeps track of what is currently in each register. It is Consulted whenever a new register is needed.

An address descriptor keeps track of the location where ever the current Value of the name can be found at run time. The location might be a register, a Stack location, a memory address,

14. What are the actions to perform the code generation algorithms?

- Invoke a function get reg to determine the location L.
- Consult the address descriptor for y to determine y', the current location of y.
- If the current values of y and/or z have no next uses, are not live on exit from the block, and are in register, alter the register descriptor.

15. Write the code sequence for the d:=(a-b)+(a-c)+(a-c).

Statement	Code generation	Register descriptor	Address descriptor
t:=a-b	MOV a,R0 SUB b,R0	R0 contains t	t in R0
u:=a-c	MOV a,R1 SUB c,R1	R0 contains t R1 contains u	t in R0 u in R1
v:=t+u	ADD R1,R0	R0 contains v R1 contains u	u in R1 v in R0
d:=v+u	ADD R1,R0 MOV R0,d	R0 contains d	d in R0 d in R0 and memory

16. Write the labels on nodes in DAG.

A DAG for a basic block is a directed acyclic graph with the following Labels on nodes:

- Leaves are labeled by unique identifiers, either variable names or constants.
- Interior nodes are labeled by an operator symbol.

- Nodes are also optionally given a sequence of identifiers for labels.

17. Give the applications of DAG.

- Automatically detect the common sub expressions
- Determine which identifiers have their values used in the block.
- Determine which statements compute values that could be used outside the blocks.

18. Define Peephole optimization.

A Statement by statement code generation strategy often produces target code that contains redundant instructions and suboptimal constructs. "Optimizing" is misleading because there is no guarantee that the resulting code is optimal. It is a method for trying to improve the performance of the target program by examining the short sequence of target instructions and replacing this instructions by shorter or faster sequence.

19. Write the characteristics of peephole optimization?

- Redundant-instruction elimination
- Flow-of-control optimizations.
- Algebraic simplifications
- Use of machine idioms

20. What are the structure preserving transformations on basic blocks?

- Common sub-expression elimination
- Dead-code elimination
- Renaming of temporary variables
- Interchange of two independent adjacent statement

21. Define Common sub-expression elimination with ex.

It is defined as the process in which eliminate the statements which has the Same expressions. Hence this basic block may be transformed into the equivalent Block.

Ex:

a : =b + c
b :=a - d
c :=b + c

After elimination:

a : =b + c
b :=a - d

c :=a

22. Define Dead-code elimination with ex.

It is defined as the process in which the statement x=y+z appear in a basic block, where x is a dead that is never subsequently used. Then this statement may be safely removed without changing the value of basic blocks.

23. Define Renaming of temporary variables with ex.

We have the statement u:=b + c ,where u is a new temporary variable, and change all uses of this instance of t to u, then the value of the basic block is not changed.

24. Define reduction in strength with ex.

Reduction in strength replaces expensive operations by equivalent cheaper ones on the target machines. Certain machine instructions are cheaper than others and can often be used as special cases of more expensive operators.

Ex:

X^2 is invariably cheaper to implement as x*x than as a call to an exponentiation routine.

25. Define use of machine idioms.

The target machine may have harder instructions to implement certain specific operations efficiently. Detecting situations that permit the use of these instructions can reduce execution time significantly.

26. Define code optimization and optimizing compiler

The **term code-optimization** refers to techniques a compiler can employ in an attempt to produce a better object language program than the most obvious for a given source program.

Compilers that apply code-improving transformations are called Optimizing-compilers.