

CS6660 - COMPILER DESIGN		L T P C 3 0 0 3
Unit-I Introduction To Compilers		
Translators-Compilation and Interpretation-Language processors -The Phases of Compiler-Errors Encountered in Different Phases-The Grouping of Phases-Compiler Construction Tools -Programming Language basics.		
PART-A CS6660.1		
1.	Define Compiler (or) What does translator mean? (May/June,2006) Compiler is a program that reads a program written in one language –the source language- and translates it into an equivalent program in another language- the target language. In this translation process, the complier reports to its user the presence of the errors in the source program.	
2.	What are the classifications of a compiler? The classifications of compiler are: Single-pass compiler, Multi-pass compiler, Load and go compiler, Debugging compiler, Optimizing compiler.	
3.	Define linker. Linker is a program that combines (multiple) objects files to make an executable. It converts names of variables and functions to numbers (machine addresses).	
4.	Define Loader. Loader is a program that performs the functions of loading and linkage editing. The process of loading consists of taking re-locatable machine code, altering the re-locatable address and placing the altered instruction and data in memory at the proper location.	
5.	Define interpreter. (Nov/Dec, 2017) Interpreter is a language processor program that translates and executes source code directly, without compiling it to machine code.	
6.	Define editor. Editors may operate on plain text, or they may be wired into the rest of the complier, highlighting syntax errors as you go, or allowing you to insert or delete entire syntax constructed at a time.	
7.	What is meant by semantic analysis? The semantic analysis phase checks the source program for semantic errors and gathers type information for the subsequent code generation phase. It uses the hierarchical structure determined by the syntax-analysis phase to identify the operators and operand of expressions and statements.	
8.	What are the phases of a compiler? or How will you group the phases of compiler?(Nov/Dec ,2013) <ul style="list-style-type: none"> • Lexical analysis • Syntax analysis • Semantic analysis • Intermediate code generation • Code optimization • Code generation 	
9.	Mention few cousins of the compiler?(May/June,2012) (April/May 2017) <div style="display: flex; justify-content: space-between;"> Pre-processors. Assemblers. </div> <div style="display: flex; justify-content: space-between;"> Two pass assembly. Loader and link editors. </div>	
10.	What is front-end and back-end of the compiler? Explain briefly(May/June,2016) Often the phases of a compiler are collected into a front-end and back-end. Front-end consists of those phases that depend primarily on the source language and largely independent of the target machine. Back-end consists of those phases that depend on the target machine language and generally those portions do not depend on the source language, just the intermediate language. In back end we use aspects of code optimization, code generation, along with error handling and symbol table operations.	
11.	List of some compiler construction tools. (Nov/Dec 2016) <ul style="list-style-type: none"> • Parser generators • Scanner generators • Syntax-directed translation engines • Automatic code generators • Data flow engines. 	

12.	<p>Define Pre-processor. What are its functions?</p> <p>Pre-processors are the programs that allow user to use macros in the sources program. Macro means some set of instructions which can be repeatedly used in the sources program . The output of pre-processor may be given as input to compiler.</p> <p>The functions of a pre-processor are:</p> <ul style="list-style-type: none"> • Macro processing. • File inclusion. • Rational pre-processors • Language extensions
13.	<p>What is linear analysis?</p> <p>The stream of characters making up the source program is read from left-to-right and grouped into tokens that are sequence of characters having a collective meaning.</p>
14.	<p>What is cross compiler? (May/June, 2014) (Nov/Dec, 2017)</p> <p>There may be a compiler which run on one machine and produces the target code for another machine. Such a compiler is called cross compiler. Thus by using cross compiler technique platform independency can be achieved.</p>
15.	<p>What is Symbol Table or Write the purpose of symbol table.(May/June, 2014 & Nov/Dec 2016))</p> <p>Symbol table is a data structures containing a record for each identifier, with fields for the attributes of the identifier. The data structure allows us to find the record for each identifier quickly and to store or retrieve data from that record quickly.</p>
16.	<p>Define Passes.</p> <p>In an implementation of a compiler, portion of one or more phases are combined into a module called pass. A pass reads the source program or the output of the previous pass, makes the transformation specified by its phases and writes output into an intermediate file, which is read by subsequent pass.</p>
17.	<p>Difference Between Assembler, Compiler and Interpreter.</p> <p>Assembler: It is the Computer Program which takes the Computer Instructions and converts them in to the bits that the Computer can understand and performs by certain Operations.</p> <p>Compiler:</p> <ul style="list-style-type: none"> • It Converts High Level Language to Low Level Language. • It considers the entire code and converts it in to the Executable Code and runs the code. • C, C++ are Compiler based Languages. <p>Interpreter:</p> <ul style="list-style-type: none"> • It Converts the higher Level language to low level language or Assembly level language. It converts to the language of 0's and 1's. • It considers single line of code and converts it to the binary language and runs the code on the machine. • If it finds the error, the programs need to run from the beginning. • BASIC is the Interpreter based Language.
18.	<p>Draw the diagram of the language processing system.(Nov/Dec,2006) (May/June,2016)</p> <pre> graph TD SC[Source Code] --> PP[Pre Processor] PP --> PPC[Pre-processed Code] PPC --> C[Compiler] C --> TAC[Target Assembly Code] TAC --> A[Assembler] A --> RMC[Relocatable Machine Code] RMC --> L[Linker] LF[Library files/ Relocatable modules] --> L L --> EMC[Executable Machine Code] EMC --> LO[Loader] LO --> M[Memory] </pre>

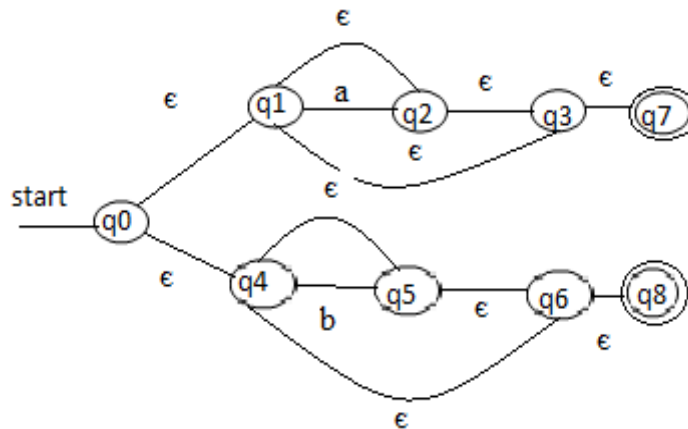
19.	State some software tools that manipulate source program?(May/June ,2006) i. Structure editors ii. Pretty printers iii. Static checkers iv. Interpreters.
20.	Define compiler-compiler. Systems to help with the compiler-writing process are often been referred to as compiler-compilers, compiler-generators or translator-writing systems. Largely they are oriented around a particular model of languages , and they are suitable for generating compilers of languages similar model.
21.	Difference between phase and pass. Phase: The compilation process into a series of sub processes are called as phase Pass: The collection of phase are called as pass.
22.	What are the two parts of a compilation? Explain briefly. (April/May 2016) (April/May 2017) There are two parts to compilation: 1. Analysis determines the operations implied by the source program which are recorded in a tree structure . 2. Synthesis takes the tree structure and translates the operations therein into the target program.
PART-B CS6660.1	
1	What is a compiler? State various phases of a compiler and explain them in detail.(16)(Nov/Dec 2016) (April/May,2017) (Nov/Dec, 2017)
2	Explain the various phases of a compiler in detail. Also write down the output for the following expression after each phase $a := b * c - d$.(16) (May/June,2016) (April/May,2017)
3	Write in detail about the analysis cousins of the compiler.(16) (May/June-2013) (April/May,2017)
4	Describe how various phases could be combined as a pass in a compiler? April/May 2008 Also briefly explain Compiler construction tools . (April/May 2015& Nov/Dec 2016) (April/May,2017) (Nov/Dec, 2017)
5	For the following expression (April/May,2017) $Position := initial + rate * 60$ Write down the output after each phase
6	Describe the following software tools i. Structure Editors ii. Pretty printers iii. Interpreters
7	Elaborate on grouping of phases in a compiler.(Nov/Dec 2016) (May/June,2016)
8	What is difference between a phase and pass of a compiler? Explain machine dependent and machine independent phase of compiler.
9	Explain the various errors encountered in different phases of compiler(Nov/Dec 2016) (May/June,2016)
10	Explain language processing system with neat diagram. (May/June,2016)
11	Give the transition diagram to represent relational operators. (April/May,2017) Give the REGULAR EXP FOR Unsigned integer is N^+ (April/May,2017)
Unit-II lexical analysis	
Need and Role of Lexical Analyzer-Lexical Errors-Expressing Tokens by Regular Expressions-Converting Regular Expression to DFA- Minimization of DFA-Language for Specifying Lexical Analyzers-LEX-Design of Lexical Analyzer for a sample Language.	
PART-A CS6660.2	
1.	What is Lexical Analysis? The first phase of compiler is Lexical Analysis. This is also known as linear analysis in which the stream of characters making up the source program is read from left-to-right and grouped into tokens that are sequences of characters having a collective meaning.
2.	What is a lexeme? Define a regular set. (Nov/Dec 2006) <ul style="list-style-type: none"> A Lexeme is a sequence of characters in the source program that is matched by the pattern for a token. A language denoted by a regular expression is said to be a regular set
3.	What is a sentinel?What is its usage? (April/May 2004)

	A Sentinel is a special character that cannot be part of the source program. Normally we use ‘eof’ as the sentinel. This is used for speeding-up the lexical analyzer.																								
4.	Mention the issues in a lexical analyzer.(May/June-2013) There are several reason for separating the analysis phase of compiling into lexical analysis and parsing.1.Simpler design is perhaps the most important consideration. The separation of lexical analysis from syntax analysis often allows us to simplify one or the other of these phases. 2. Compiler efficiency is improved. 3. Compiler portability is enhanced.																								
5.	What is the role of lexical analysis phase? (Nov/Dec, 2017) The lexical analyzer breaks a sentence into a sequence of words or tokens and ignores white spaces and comments. It generates a stream of tokens from the input. This is modeled through regular expressions and the structure is recognized through finite state automata.																								
6.	What are the possible error recovery actions in a lexical analyzer?(May/june-2012),(April/May 2015) i. Panic mode recovery. Delete successive characters from the remaining input until the lexical analyzer can find a Well-formed token. This technique may confuse the parser. ii. Other possible error recovery actions: <ul style="list-style-type: none">• Deleting an extraneous characters• Inserting missing characters.• Replacing an incorrect character by a correct character.• Transposing two adjacent characters																								
7.	Define Tokens, Patterns and Lexemes (May/June2014 &Nov/Dec 2016) (Nov/Dec, 2017) Tokens: A token is a syntactic category. Sentences consist of a string of tokens. For example number, identifier, keyword, string etc are tokens Lexemes: Lexeme: Sequence of characters in a token is a lexeme. For example 100.01, counter, const, "How are you?" etc are lexemes. Patterns: Rule of description is a pattern. For example letter (letter digit)* is a pattern to symbolize a set of strings which consist of a letter followed by a letter or digit.																								
8.	Write short note on input buffering. (or).Why is buffering used in lexical analysis ? What are the commonly used buffering methods?(May/June 2014) Lexical analyzer scan the sources program line by line. For storing the input string, which is to be read, the lexical analyzer makes use of input buffer. The lexical analyzer maintains two pointer forward and backward pointer for scanning the input string. There are two types of input buffering schemes- one buffer scheme and two buffer scheme.																								
9.	What are the drawbacks of using buffer pairs? <ul style="list-style-type: none">i. This buffering scheme works quite well most of the time but with it amount of look ahead is limited.ii. Limited look ahead makes it impossible to recognize tokens in situations Where the distance, forward pointer must travel is more than the length of buffer.																								
10.	Define regular expressions.(or) Write a regular expression for an identifier. Regular expression is used to define precisely the statements and expressions in the source language. For e.g. in Pascal the identifiers is denotes in the form of regular expression as letter letter(letter digit)* .																								
11.	What are algebraic properties of regular expressions? The algebraic law obeyed by regular expressions are called algebraic properties of regular expression. The algebraic properties are used to check equivalence of two regular expressions. <table><tr><th>S.No</th><th>Properties</th><th>Meaning</th></tr><tr><td>1</td><td>$r_1 r_2=r_2 r_1$</td><td> is commutative</td></tr><tr><td>2</td><td>$r_1 (r_1 r_3)=(r_1 r_2) r_3$</td><td> is associative</td></tr><tr><td>3</td><td>$(r_1 r_2)r_3=r_1(r_2 r_3)$</td><td>Concatenation is associative</td></tr><tr><td>4</td><td>$r_1(r_2 r_3)=r_1 r_2 r_1 r_3$ $(r_2 r_3) r_1=r_2 r_1 r_3 r_1$</td><td>Concatenation is distributive over </td></tr><tr><td>5</td><td>$\epsilon r = r \epsilon = r$</td><td>$\epsilon$ is identity</td></tr><tr><td>6</td><td>$r^*=(r \epsilon)^*$</td><td>Relation between ϵ and *</td></tr><tr><td>7</td><td>$r^{**}=r^*$</td><td>*is idempotent</td></tr></table>	S.No	Properties	Meaning	1	$r_1 r_2=r_2 r_1$	is commutative	2	$r_1 (r_1 r_3)=(r_1 r_2) r_3$	is associative	3	$(r_1 r_2)r_3=r_1(r_2 r_3)$	Concatenation is associative	4	$r_1(r_2 r_3)=r_1 r_2 r_1 r_3$ $(r_2 r_3) r_1=r_2 r_1 r_3 r_1$	Concatenation is distributive over	5	$\epsilon r = r \epsilon = r$	ϵ is identity	6	$r^*=(r \epsilon)^*$	Relation between ϵ and *	7	$r^{**}=r^*$	*is idempotent
S.No	Properties	Meaning																							
1	$r_1 r_2=r_2 r_1$	is commutative																							
2	$r_1 (r_1 r_3)=(r_1 r_2) r_3$	is associative																							
3	$(r_1 r_2)r_3=r_1(r_2 r_3)$	Concatenation is associative																							
4	$r_1(r_2 r_3)=r_1 r_2 r_1 r_3$ $(r_2 r_3) r_1=r_2 r_1 r_3 r_1$	Concatenation is distributive over																							
5	$\epsilon r = r \epsilon = r$	ϵ is identity																							
6	$r^*=(r \epsilon)^*$	Relation between ϵ and *																							
7	$r^{**}=r^*$	*is idempotent																							
12.	What is meant by recognizer? It is a part of LEX analyzer that identifies the presence of a token on the input is a recognizer for the language defining that token. It is the program, which automatically recognizes																								

	the tokens. A recognizer for a language L is a program that takes an input string “x” and response “yes” if “x” is sentence of L and “no” otherwise.
13.	List the operations on languages. (May/June,2016) <ul style="list-style-type: none"> • Union - $L \cup M = \{s \mid s \text{ is in } L \text{ or } s \text{ is in } M\}$ • Concatenation - $LM = \{st \mid s \text{ is in } L \text{ and } t \text{ is in } M\}$ • Kleene Closure - L^* (zero or more concatenations of L) • Positive Closure - L^+ (one or more concatenations of L)
14.	Mention the various notational shorthands for representing regular expressions. <ul style="list-style-type: none"> • One or more instances (+) • Zero or one instance (?) • Character classes ([abc] where a,b,c are alphabet symbols denotes the regular expressions $a \mid b \mid c$.) • Non regular sets
15.	List the rules for constructing regular expressions? The rules are divided into two major classifications (i) Basic rules (ii) Induction rules Basic rules: <ol style="list-style-type: none"> ϵ is a regular expression that denotes $\{\epsilon\}$ (i.e.) the language contains only an empty string. For each a in Σ, then a is a regular expression denoting $\{a\}$, the language with only one string, which consists of a single symbol a. Induction rules: <ol style="list-style-type: none"> If R and S are regular expressions denoting languages LR and LS respectively then, $(R) \cup (S)$ is a regular expression denoting $LR \cup LS$ $(R).(S)$ is a regular expression denoting $LR . LS$ $(R)^*$ is a regular expression denoting LR^*.
16.	Define DFA. A DFA is an acceptor for which any state and input character has utmost one transition state that the acceptor changes to. If no transition state is specified the input string is rejected. <ol style="list-style-type: none"> It has no ϵ-transitions. For each state “S” and input symbol ‘a’, there is at most one edge labeled ‘a’ leaving from “S” DFA is easy to simulate.
17.	What are the conditions to satisfy for NFA? <ol style="list-style-type: none"> NFA should have one start state. NFA may have one or more accepting states. ϵ-Transitions may present on NFA. There can be more than transitions, on same input symbol from any one state. In NFA, from any state “S” <ol style="list-style-type: none"> There can be at most 2 outgoing ϵ-transitions. There can be only one transition on real input symbol. On real input transition it should reach the new state only.
18.	Write a short note on LEX. A LEX source program is a specification of lexical analyzer consisting of set of regular expressions together with an action for each regular expression. The action is a piece of code, which is to be executed whenever a token specified by the corresponding regular expression is recognized. The output of a LEX is a lexical analyzer program constructed from the LEX source specification.
19.	How to recognize tokens Consider the following grammar and try to construct an analyzer that will return <token, attribute> pairs. $\text{relop} \rightarrow < \mid = \mid > \mid < > \mid = >$ $\text{id} \rightarrow \text{letter} (\text{letter} \mid \text{digit})^*$ $\text{num} \rightarrow \text{digit}^+ ('.' \text{digit}^+)? (E ('+' \mid '-')? \text{digit}^+)?$ $\text{delim} \rightarrow \text{blank} \mid \text{tab} \mid \text{newline}$ $\text{ws} \rightarrow \text{delim}^+$
20.	What is finite automate ? A finite automate is a mathematical model that consists of- <ol style="list-style-type: none"> Input set Σ A finite set of states S A transition function to move from one state to other.

	<p>iv)A special state called start state. v)A special state called accept state. The finite automate can be diagrammatically represented by a directed graph called transition graph.</p>
21.	<p>What is the need for separating the analysis phase into lexical analysis and parsing? i)Separation of lexical analysis from syntax allows the simplified design. ii)Efficiency of compiler gets increased. Reading of input source file is a time consuming process and if it is been done once in lexical analyzer then efficiency get increased . Lexical analyzer uses buffering techniques to improve the performances. iii)Input alphabet peculiarities and other device specific anomalies can be restricted to the lexical analyzer.</p>
22.	<p>Draw a transition diagram to represent relational operators. The relational operators are:<,<=,>,>=,!=</p>
23.	<p>State any reasons as to why phases of compiler should be grouped.(May/June 2014) 1.By keeping the same front end and attaching different back ends one can produce a compiler for same source language on different machine. 2.By keeping different fronts ends and same back end one can compile several different language on the same machine.</p>
24.	<p>Define a context free grammar. A context free grammar G is a collection of the following</p> <ul style="list-style-type: none"> • V is a set of non terminals • T is a set of terminals • S is a start symbol • P is a set of production rules • G can be represented as $G = (V, T, S, P)$ • Production rules are given in the following form • Non terminal $\rightarrow (V \cup T)^*$
25.	<p>Write a regular expression for identifier and number. (April/May/,2017) Regular expression for identifier - letter(letter digit)*. Regular expression for integer constant - digit⁺ Regular expression for real constant - digit⁺.digit⁺</p>

26. Draw a NFA for $a^*|b^*$. (April/May-2004, Nov/Dec-2005)



27. What is the time and space complexity of NFA and DFA.

Automaton	Space	Time
NFA	$O(r)$	$O(r ^* r)$
DFA	$O(2^{ r })$	$O(x)$

28. Write a regular definition to represent date and time in the following format: MONTH DAY YEAR. (April/May-2015)

MONTH--> [Jan-Dec]

DAY--> [1-31]

YEAR--> [1900-2025]

29. Write a grammar for branching statements. (May/June, 2016)

St -> id := expr

| if expr then St

| if expr then St else St

| while expr do st

| begin Sts end

30. What are the various parts of LEX program. (April/May, 2017)

{definitions}

%%

{transition rules}

%%

{user subroutines}

Example:

digit [0-9]

letter [a-zA-Z]

%%

{letter}({letter}|{digit}) printf("id: %s\n", yytext);*

\n

printf("new line\n");

%%

main() {

yylex();

}

PART-B CS6660.2

1. Explain about input buffering. (8)

2. Explain in detail about the role of Lexical analyzer with the possible error recovery actions. (May/June-2013 & Nov/Dec 2016) (April/May 2015)

3. Describe the specification of tokens and how to recognize the tokens (16) (May/June-2013)

4. Describe the language for specifying lexical Analyzer. (16)

5. prove that the following two regular expressions are equivalent by showing that minimum state DFA's are same.
(i) $(a|b)^*$

	(ii)(a* b*)*#
6.	Draw the transition diagram to represent relational operators. (April/May,2017) Give the REGULAR EXP FOR Unsigned integer is N^+ (April/May,2017) Draw the transition diagram for unsigned numbers.(6) Nov/Dec,2006,2007
7.	Construct the NFA from the i.(a/b)*a(a/b) & ii.(ab*/ab) using Thompson's construction algorithm.(10) May/June 2007 (Nov/Dec, 2017)
8.	write notes on regular exp to NFA. Construct RE to NFA for the sentence (a/b)*a. (May/June,2016) Write a algorithm to construct an NFA into a regular expression. Construct DFA to recognize (a/b)* ab.
9.	write an algorithm for minimizing the number of states of a DFA. Give the minimized DFA for the following expression. Nov/Dec,2006,2007 (Nov/Dec 2016) (a/b)*abb
10.	Construct a Minimal DFA for abb(a/b)* using direct method (April/May,2017) (Nov/Dec, 2017)
11.	Write LEX specifications and necessary C code that reads English words from a text file and response every occurrence of the sub string 'abc' with 'ABC'. The program should also compute number of characters, words and lines read. It should not consider and count any lines(s) that begin with a symbol '#'
12.	a) Prove that the following two regular expressions are equivalent by showing that the minimum state DFA's are same. (i) (a/b)* (ii)(a*/b*) (April/May 2015)
13.	Discuss how finite automata is used to represent tokens and perform lexical analysis with examples (Nov/Dec 2016)
14.	Differentiate between Tokens, Patterns and Lexemes (May/June,2016) (April/May,2017) What are the issues in lexical analysis. (Nov/Dec, 2017) Write notes on regular expressions

Unit-III Syntax Analysis

Need and Role of the Parser-Context Free Grammars -Top Down Parsing -General Strategies-Recursive Descent Parser Predictive Parser-LL(1) Parser-Shift Reduce Parser-LR Parser-LR (0)Item-Construction of SLR Parsing Table -Introduction to LALR Parser - Error Handling and Recovery in Syntax Analyzer-YACC-Design of a syntax Analyzer for a Sample Language .

PART-A CS6660.3

1.	Define parser.(or) What is the role of parser ?(April/May 2015) A parsing or syntax analysis is a process which takes the input string w and produces either a parse tree(syntactic structure) or generates the syntactic errors.
2.	Mention the basic issues in parsing. There are two important issues in parsing: 1) Specification of syntax. 2) Representation of input after parsing.
3.	Why lexical and syntax analyzers are separated out? The reason for separating the lexical and syntax analyzer. <ul style="list-style-type: none"> • Simpler design. • Compiler efficiency is improved. • Compiler portability is enhanced.
4.	Briefly explain the concept of derivation. Derivation from S means generation of string w from S. For constructing derivation two things are important. i. Choice of non-terminal from several others. ii. Choice of rule from production rules for corresponding non terminal Instead of choosing the arbitrary rules for corresponding non terminal i. Either leftmost derivation-leftmost non terminal in a sentinel form ii. Or rightmost derivation -rightmost non terminal in a sentinel form
5.	Define ambiguous grammar. (May/June,2016) A grammar G is said to be ambiguous if it generates more than one parse tree for some sentence of language L (G). i.e. both leftmost and rightmost derivations are same for the given sentence.
6.	What is operator precedence parser? A grammar is said to be operator precedence if it possesses the following properties: i. No production on the right side is ϵ

	ii. There should not be any production rule possessing two adjacent terminals at the right hand side.
7.	What is meant by left recursion? A grammar is left recursive if it has a non terminal A such that there is derivation $A \Rightarrow^+ A\alpha$ for some string α . Top down parsing methods cannot handle left-recursion grammars, so a transformation that eliminates left recursion is needed. Ex:- $E \rightarrow E + T \mid T$ $T \rightarrow T * F \mid F$ $F \rightarrow (E) \mid id$
8.	Write the algorithm to eliminate left recursion from a grammar? 1. Arrange the non terminals in some order A_1, A_2, \dots, A_n 2 for $i := 1$ to n do begin for $j := 1$ to $i-1$ do begin replace each production of the form $A_i \rightarrow A_j Y$ by the productions $A_i \rightarrow \delta_1 \gamma \mid \delta_2 \gamma \mid \dots \mid \delta_k \gamma$, where $A_j \rightarrow \delta_1 \mid \delta_2 \mid \dots \mid \delta_k$ are all the current A_j -productions; end eliminate the immediate left recursion among the A_i - productions end.
9.	What is meant by left factoring? Left factoring is a grammar transformation that is useful for producing a grammar suitable for predictive parsing. The basic idea is that when it is not clear which of two alternative productions to use to expand a non terminal A, we may be able to rewrite the A production to defer the decision until we have seen enough of the input to make the right choice.
10.	What is LR Parsers? LR(k) parsers scan the input from (L) left to right and construct a (R) rightmost derivation in reverse. LR parsers consist of a driver routine and a parsing table. The k is for the number of input symbols of look ahead that are used in making parsing decisions. When k is omitted, k is assumed to be 1.
11.	Mention the properties of parse tree? <ul style="list-style-type: none"> • The root is labeled by the start symbol. • Each leaf is labeled by a token. • Each interior node is labeled by a non-terminal. • If A is the Non terminal, labeling some interior node and $x_1, x_2, x_3 \dots x_n$ are the labels of the children
12.	Define Handles. A handle of a right-sentential form Y is a production $A \rightarrow \beta$ and a position of Y where the string β may be found and replaced by A to produce the previous right-sentential form in a rightmost derivation of Y
13.	What are the problems in top down parsing? a) Left recursion. b) Backtracking. c) The order in which alternates are tried can affect the language accepted.
14.	Define 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. The recursive procedures can be quite easy to write.
15.	Define predictive parsers. A predictive parser is an efficient way of implementing recursive-descent parsing by handling the stack of activation records explicitly. The predictive parser has an input, a stack, a parsing table and an output.
16.	What is non recursive predictive parsing? Non recursive predictive parser can be maintained by a stack explicitly, rather than implicitly via recursive calls. The key problem during predictive parsing is that determining the production to be applied for a non-terminal. The non recursive parser looks up the production to be applied in a

	parsing table.
17.	<p>Write the algorithm for FIRST? (Or) Define FIRST in predictive parsing. (May/June,2016)</p> <p>1. If X is terminal, then FIRST(X) is {X}.</p> <p>2. If $X \rightarrow \epsilon$ is a production, then add ϵ to FIRST(X).</p> <p>3. If X is non terminal and $X \rightarrow Y_1 Y_2 \dots Y_k$ is a production, then place a in FIRST(X) if for some i, a is in FIRST(Y_i), and ϵ is in all of FIRST(Y_1),...FIRST(Y_{i-1});</p>
18.	<p>Write the algorithm for FOLLOW? (Or) Define FOLLOW in predictive parsing. (May/June,2016)</p> <p>1. Place \$ in FOLLOW(S), where S is the start symbol and \$ is the input right end marker.</p> <p>2. If there is a production $A \rightarrow aB\beta$, then everything in FIRST(β) except for ϵ is placed in FOLLOW(B).</p> <p>3. If there is a production $A \rightarrow aB$, or a production $A \rightarrow aB\beta$ where FIRST(β) contains ϵ, then everything in FOLLOW(A) is in FOLLOW(B).</p>
19.	<p>Write the algorithm for the construction of a predictive parsing table?</p> <p>Input : Grammar G</p> <p>Output : Parsing table M</p> <p>Method :</p> <ol style="list-style-type: none"> For each production $A \rightarrow \alpha$ of the grammar, do steps b and c. For each terminal a in FIRST(α), add $A \rightarrow \alpha$ to M[A, a] If ϵ is in FIRST(α), add $A \rightarrow \alpha$ to M[A, b] for each terminal b is FOLLOW(A). if ϵ is in FIRST(α) and \$ is in FOLLOW(A), and $A \rightarrow \alpha$ to M[A, \$] Make each undefined entry of M be error.
20.	<p>What is LL(1) grammar?</p> <p>A grammar whose parsing table has no multiply-defined entries is said to be LL(1).</p>
21.	<p>Define handle pruning. (Nov/Dec-2016)</p> <p>A technique to obtain the rightmost derivation in reverse (called canonical reduction sequence) is known as handle pruning (i.e.) starting with a string of terminals w to be parsed. If w is the sentence of the grammar then $w = Y_n$, where Y_n is the nth right sentential form of unknown right most derivation.</p>
22.	<p>What is shift reduce parsing?</p> <p>The bottom up style of parsing is called shift reduce parsing. This parsing method is bottom up because it attempts to construct a parse tree for an input string beginning at the leaves and working up towards the root.</p>
23.	<p>What are the four possible action of a shift reduce parser?</p> <ol style="list-style-type: none"> Shift action – the next input symbol is shifted to the top of the stack. Reduce action – replace handle. Accept action – successful completion of parsing. Error action- find syntax error.
24.	<p>Define viable prefixes.</p> <p>The set of prefixes of right sentential forms that can appear on the stack of a shift-reduce parser are called viable prefixes. An equivalent definition of a viable prefix is that it is a prefix of a right sentential form that does not continue past the right end of the rightmost handle of that sentential form.</p>
25.	<p>Define LR (0) item.</p> <p>An LR(0) item of a grammar G is a production of G with a dot at some position of the right side.</p> <p>Eg: $A \rightarrow .XYZ$ $A \rightarrow X.YZ$ $A \rightarrow XY.Z$ $A \rightarrow XYZ.$</p>
26.	<p>What is augmented grammar? (or) How will you change the given grammar to an augmented grammar?</p> <p>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 $S' \rightarrow S$. It is to indicate the parser when it should stop and announce acceptance of the input.</p>
27.	<p>Left factor the following grammar:</p> <p>$S \rightarrow iEtS \mid iEtSeS \mid a$</p> <p>$E \rightarrow b.$</p> <p>Ans: The left factored grammar is,</p> <p>$S \rightarrow iEtSS' \mid a$</p> <p>$S' \rightarrow eS \mid \epsilon$</p> <p>$E \rightarrow b$</p>

28.	Eliminate the left recursion for the grammar $A \rightarrow Ac \mid Aad \mid bd$ (April/May, 2017) $A \rightarrow bd A'$ $A' \rightarrow adA' \mid cA' \mid \epsilon$
29.	What are the various conflicts that occur in during shift-reduce parsing? (April/May, 2017) There are two conflicts that occur in shift-reduce parsing: 1. Shift-reduce conflict: The parser cannot decide whether to shift or to reduce. 2. Reduce-reduce conflict: The parser cannot decide which of several reductions to make.
30.	Explain the need of augmentation. To indicate to the parser that, when it should stop parsing and when to announce acceptance of the input. Acceptance occurs when the parser is about to reduce by $S' \rightarrow S$.
31.	How to make an ACTION and GOTO entry in SLR parsing table? i. If $[A \rightarrow \alpha.a\beta]$ is in I_i and $\text{goto}(I_i, a) = I_j$ then set $\text{ACTION}[I_i, a]$ to $\text{shift } j$. Here α must be a terminal. ii. If $[A \rightarrow \alpha.]$ is in I_i then set $\text{ACTION}[I_i, a]$ to "reduce $A \rightarrow \alpha$ " for all a in $\text{FOLLOW}(A)$, here A should not be S' . iii. If $S' \rightarrow S$ is in I_i then set $\text{ACTION}[I_i, \$]$ to "accept".
32.	What are the rules to apply Closure function in CLR parser? Let 'L' is a set of LR (1) items for grammar then 2 steps can compute Closure of L. i. Initially every item in I is added to Closure (I). ii. Consider, $A \rightarrow X.BY, a$ $B \rightarrow Z$ are 2 productions and X, Y, Z are grammar symbols. Then add $B \rightarrow .Z$, $\text{FIRST}(Ya)$ as the new LR(1) item, if it is not already there. This rule has to be applied till no new items can be added to Closure (I).
33.	Specify the advantages of LALR. Merging of states with common cores can never produce a shift/reduce conflict that was not present in any one of the original states. Because shift actions depends only one core, not the look ahead.
34.	Define terminal & Nonterminal Terminals are the basic symbols from which the strings are formed. Nonterminals are syntactic variables that denote set of strings.
35.	Mention the demerits of LALR parser. <ul style="list-style-type: none"> • _ Merger will produce reduce / reduce conflict. • _ On erroneous input, LALR parser may proceed to do some reductions after the LR parser has declared an error, but LALR parser never shift a symbol after the LR parser declares an error.
36.	What is the syntax for YACC source specification program? Declarations %% Translation rules %% Supporting C-routines
37.	Differentiate Kernel and non-Kernel items. Kernel items, which include the initial item, $S' \rightarrow .S$ and all items whose dots are not at the left end. Whereas the nonkernel items have their dots at the left end.
38.	What are the components of LR parser? <ul style="list-style-type: none"> ➤ An input. ➤ An output. ➤ A stack. ➤ A driver program. ➤ A parsing table.
39.	List the different techniques to construct an LR parsing table? <ul style="list-style-type: none"> • Simple LR(SLR). • Canonical LR. • Lookahead LR (LALR).
40.	Eliminate left recursion from the following grammar $A \rightarrow Ac/Aad/bd/c$. (May/June 2013) $A \rightarrow bdA'$ $A' \rightarrow aAcA' / adA' / \epsilon$

41.	<p>Write the regular expression for identifier and whitespace.(Nov/Dec 2013)</p> <p>An Identifier (or Name) $/[a-zA-Z_][0-9a-zA-Z_]* /$</p> <ol style="list-style-type: none"> 1. Begin with one letters or underscore, followed by zero or more digits, letters and underscore. 2. You can use meta character \w (word character) for $[a-zA-Z0-9_]$; \d (digit) for $[0-9]$. Hence, it can be written as $/[a-zA-Z_]\w* /$. 3. To include dash (-) in the identifier, use $/[a-zA-Z_][\w-]* /$. Nonetheless, dash conflicts with subtraction and is often excluded from identifier. <p>An Whitespace $\backslash s \backslash s /$ # Matches two whitespaces $\backslash S \backslash S \backslash s /$ # Two non-whitespaces followed by a whitespace $\backslash s + /$ # one or more whitespaces $\backslash S + \backslash s \backslash S + /$ # two words (non-whitespaces) separated by a whitespace</p>
42.	<p>Eliminate the left recursion for the grammar (Nov/Dec 2013)</p> <p>$S \rightarrow Aa b$ $A \rightarrow Ac Sd e$</p> <p>Sol: Let's use the ordering S, A ($S = A1$, $A = A2$).</p> <ul style="list-style-type: none"> • When $i = 1$, we skip the "for j" loop and remove immediate left recursion from the S productions (there is none). • When $i = 2$ and $j = 1$, we substitute the S-productions in $A \rightarrow Sd$ to obtain the A-productions $A \rightarrow Ac Aad bd \epsilon$ • Eliminating immediate left recursion from the A productions yields the grammar: $S \rightarrow Aa b$ $A \rightarrow bdA' A'$ $A' \rightarrow cA' adA' \epsilon$
43.	<p>Compare the feature of DFA and NFA.(May/June 2014) (Nov/Dec, 2017)</p> <ol style="list-style-type: none"> 1.Both are transition functions of automata. In DFA the next possible state is distinctly set while in NFA each pair of state and input symbol can have many possible next states. 2.NFA can use empty string transition while DFA cannot use empty string transition. 3.NFA is easier to construct while it is more difficult to construct DFA. 4.Backtracking is allowed in DFA while in NFA it may or may not be allowed. 5.DFA requires more space while NFA requires less space.
44.	<p>Eliminate left recursion for the grammar. $E \rightarrow E+T/T, T \rightarrow T * F/F, F \rightarrow (E)/id$ (April/May-08,Marks 2)</p> <p>$A \rightarrow A\alpha/\beta$ then convert it to $A \rightarrow \beta A'$ $A' \rightarrow \alpha A'$ $A' \rightarrow \epsilon$</p> <p>Eg : $E \rightarrow TE'$ $E' \rightarrow +TE'/\epsilon$ $T \rightarrow FT'$ $T' \rightarrow *FT'/\epsilon$ $F \rightarrow (E)/id$</p>
45.	<p>Construct a parse tree for $-(id+id)$ (Nov/Dec 2016) (Nov/Dec, 2017)</p> <p>$E \rightarrow E+E / E * E / -E / (E) / id$</p> <pre> graph TD E1[E] --- Minus[-] E1 --- E2[E] E2 --- LParen[(] E2 --- E3[E] E2 --- RParen[)] E3 --- E4[E] E3 --- Plus[+] E3 --- E5[E] E4 --- ID1[id] E5 --- ID2[id] </pre>

PART-B CS6660.3	
1.	<p>i. Find the language from (4) $S \rightarrow 0S1 \mid 0A1 \mid A \rightarrow 1A0 \mid 10$</p> <p>ii. Define Parse tree , Regular Expression , Left most derivation , Right most (4) derivation , and write example for each.</p> <p>iii. Write algorithm to convert NFA from Regular expression. (4)</p> <p>iv. Find the language from (4) $S \rightarrow 0S1 \mid 0A \mid 0 \mid 1B \mid 1 \quad A \rightarrow 0A \mid 0 \quad B \rightarrow 1B \mid 1$</p>
2.	Explain context free grammar with examples. (May/June,2016)
3.	<p>i. Prove the grammar is ambiguous. (4) (Nov/Dec 2016) $E \rightarrow E+E \mid E * E \mid (E) \mid id$</p> <p>ii. Specify the demerits of ambiguous grammar. (2)</p> <p>iii. What are the rules to convert an unambiguous grammar from ambiguous grammar .Write necessary steps for the above ambiguous grammar.</p> <p>iv. Using unambiguous grammar, write Leftmost derivation ,draw parse tree for the string $id * id * id + id * id$</p>
4.	<p>Construct Stack implementation of shift reduce parsing for the grammar given below:</p> $E \rightarrow E+E$ $E \rightarrow E * E$ $E \rightarrow (E)$ <p>$E \rightarrow id$ and the input string is $id1 + id2 * id3$</p> <p>Describe the conflicts that may occur during shift reduce parsing. (May/June,2016)</p>
5.	<p>Construct a recursive decent parser for the following grammar and write the algorithm for elimination of left recursion and give the stack implementation for the sentence “ $id + id * id \\$”</p> $E \rightarrow E+T \mid T$ $T \rightarrow T * F \mid F$ $F \rightarrow (E) \mid id$
6.	Write in detail about Recursive Predictive parser and Non-Recursive Predictive parser (Nov/Dec, 2017)
7.	Explain LR parsing algorithm. (Nov/Dec, 2017)
8.	<p>Construct Predictive Parser (Non-Recursive Predictive parser) for the following grammar and write the algorithm for FIRST AND FOLLOW and find moves made by predictive parser on input $id + id * id$</p> $E \rightarrow E+T \mid T$ $T \rightarrow T * F \mid F$ $F \rightarrow (E) \mid id$ <p>(May/June-2012&13) (Nov/Dec 2016) (Nov/Dec, 2017)</p>
9.	<p>Check whether the following grammar is a LL(1) grammar</p> $S \rightarrow iEtS \mid iEtSeS \mid a$ $E \rightarrow b$ <p>Also define the FIRST and FOLLOW procedures (May/June,2016)</p>
10.	<p>Construct non recursion predictive parsing table for the following grammar.And write the algorithm,</p> $E \rightarrow E \text{ or } E / E \text{ and } E / \text{ not } E / (E) / 0 / 1.$ <p>(Dec-12,Marks 16). (May/June,2016)</p> <p>(or)</p> $E \rightarrow E \text{ or } T / T$ $T \rightarrow T \text{ and } F / F$ $F \rightarrow \text{not } G$ $G \rightarrow (E) / 0 / 1$ <p>(or)</p> $bexpr \rightarrow bexpr \text{ OR } bterm \mid bterm$ $bterm \rightarrow bterm \text{ AND } bfactor \mid bfactor$ $bfactor \rightarrow \text{NOT } bfactor \mid (bexpr) \mid \text{true} \mid \text{false}$
11.	<p>Consider the following grammar (Nov/Dec-2012) (Nov/Dec 2016) (April/May,2017)</p> $E \rightarrow E+T \mid T$ $T \rightarrow TF \mid F$ $F \rightarrow F * \mid a \mid b$ <p>construct the SLR parsing table for this grammar. Also parse the input $a * b + a$.</p>
12.	<p>(i)Construct SLR parsing table for the following grammar (10). (Nov/Dec-2012, April/May-04)</p> $S \rightarrow L = R \mid R \quad L \rightarrow * R \mid id \quad R \rightarrow L$

13.	Parse the string (a,a) using SLR parsing table. $S \rightarrow (L) \mid a$ $L \rightarrow L, S \mid S$
14.	Construct a predictive Parsing table for the grammar. (April/May,2017) $S \rightarrow (L) \mid a$ $L \rightarrow L, S \mid S$ And show whether (a,(a,(a,a))) is accepted or not.
15.	Generate SLR Parsing table for the following grammar. $S \rightarrow Aa \mid bAc \mid Bc \mid bBa$ $A \rightarrow d$ $B \rightarrow d$ And parse the sentences “bdc” and “dd”. (April/May 2015)
16.	Construct CLR parsing table to parse the sentence id=id*id for the following grammar. $S \rightarrow L=R \mid R$ $L \rightarrow *R \mid id$ $R \rightarrow L$
17.	Construct LALR parsing table for the grammar. $E \rightarrow E+T \mid T$ $T \rightarrow T * F \mid F$ $F \rightarrow (E) \mid id$
18.	Write in detail about i. Recursive descent parsing with algorithm. ii. Top down parsing with algorithm.
19.	(i) Write the algorithm to eliminate left-recursion and left-factoring and apply both to the following grammar. (8) $E \rightarrow E+T \mid E-T \mid T$ $T \rightarrow a \mid b(E)$ (April/May 2015)
20.	Construct a parse tree for the input string w = cad using top down parser (Nov/Dec 2016) $S \rightarrow cAd$ $A \rightarrow ab \mid a$

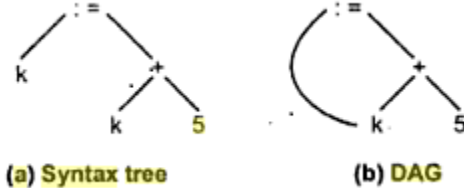
Unit IV Syntax Directed Translation & Run Time Environment

Syntax directed Definitions-Construction of Syntax Tree-Bottom-up Evaluation of S-Attribute Definitions-Design of predictive translator - Type Systems-Specification of a simple type checker-Equivalence of Type Expressions-Type Conversions.

RUN-TIME ENVIRONMENT: Source Language Issues-Storage Organization-Storage Allocation-Parameter Passing-Symbol Tables-Dynamic Storage Allocation-Storage Allocation in FORTRAN.

PART-A CS6660.4

1.	Define syntax directed definition. Syntax directed definition is a generalization of context free grammar in which each grammar production $X \rightarrow \alpha$ is associated with it a set of semantic rules of the form $a := f(b_1, b_2, \dots, b_k)$, where a is an attribute obtained from the function f. It is a generalization of a CFG in which each grammar symbol has an associated set of attributes like, synthesized attribute and inherited attribute
2.	How the value of synthesized attribute and inherited attribute are computed? synthesized attribute : It was computed from the values of attributes at the children of that node in the parse tree. inherited attribute : It was computed from the value of attributes at the siblings and parent of that node.
3.	What are the functions used for construction of syntax tree for expression? Explain. Construction syntax tree for an expression means translation of expression into postfix form. The nodes for each operator and operand is created. Each node can implemented as a record with multiple fields. Following are the function used in syntax tree for expression. <ol style="list-style-type: none"> Mknode(op,left,right) This function creates a node with field operator having operator as label, and the two pointer to left and right. Mknode(id,entry) This function creates identifier node with label id and a pointer to symbol table is given by 'entry'. Mkleaf(num,val)

	This function creates node for number with label num and val is for value of that number.
4.	What do you mean by DAG? Compare DAG with syntax tree. (Nov/Dec, 2017) It is Directed Acyclic Graph. In this common sub expressions are eliminated. So it is a compact way of representation. Like syntax tree DAG has nodes representing the subexpressions in the expression. These nodes have operator, operand1 and operand2 where operands are the children of that node. The Difference between DAG and syntax tree is that common subexpressions has more than one parent and in syntax tree the common subexpression would be represented as duplicated subtree.
5.	Construct a syntax tree and DAG for $k:=k+5$. (Nov/Dec, 2017)  <p>(a) Syntax tree (b) DAG</p>
6.	What is S-attributed definition? S-Attributed Grammars are a class of attribute grammars characterized by having no inherited attributes, but only synthesized attributes. Inherited attributes, which must be passed down from parent nodes to children nodes of the abstract syntax tree during the semantic analysis of the parsing process, are a problem for bottom-up parsing because in bottom-up parsing, the parent nodes of the abstract syntax tree are created after creation of all of their children. Attribute evaluation in S-attributed grammars can be incorporated conveniently in both top-down parsing and bottom-up parsing.
7.	What is L-attributed definition? L-attributed grammars are a special type of attribute grammars. They allow the attributes to be evaluated in one left-to-right traversal of the abstract syntax tree. As a result, attribute evaluation in L-attributed grammars can be incorporated conveniently in top-down parsing. Many programming languages are L-attributed. Special types of compilers, the narrow compilers, are based on some form of L-attributed grammar. These are comparable with S-attributed grammars. Used for code synthesis. The Class of L-attributes can be evaluated in depth first order.
8.	Define a translation scheme. A translation scheme is a context-free grammar in which semantic rules are embedded within the right sides of the productions. So a translation scheme is like a syntax-directed definition, except that the order of evaluation of the semantic rules is explicitly shown. The position at which an action is to be executed.
9.	What are the advantages of SDT? Syntax directed translation is a scheme that indicate the order in which semantic rules are to be evaluated. The main advantage of SDT is that it helps in deciding evaluation order. The evaluation of semantic actions associated with SDT may generate code, save information in symbol table, or may issue error messages.
10.	What is type checking? Type checker verifies that the type of a construct (constant,variable,array,list,object) matches what is expected in its usage context.
11.	What are static and dynamic errors? Static error: It can be detected at compile time. Eg: Undeclared identifiers. Dynamic errors: It can be detected at run time. Eg: Type checking
12.	What are the advantages of compile time checking? (i) It can catch many common errors. (ii) Static checking is desired when speed is important, since it can result faster code that does not perform any type checking during execution.
13.	What are the advantages of the dynamic checking? <ul style="list-style-type: none"> It usually permits the programmer to be less concerned with types. Thus, it frees the programmer. It may be required in some cases like array bounds check, which can be performed only during execution. It can give in clearer code.

	<ul style="list-style-type: none"> It may rise to in more robust code by ensuring thorough checking of values for the program identifiers during execution.
14.	Define type systems. Type system of a language is a collection of rules depicting the type expression assignments to program objects. An implementation of a type systems is called a type checker.
15.	Write Static vs. Dynamic Type Checking Static: Done at compile time (e.g., Java) Dynamic: Done at run time (e.g., Scheme) Sound type system is one where any program that passes the static type checker cannot contain run-time type errors. Such languages are said to be strongly typed.
16.	Define procedure definition. A procedure definition is a declaration that, in its simplest form, associates an identifier with a statement. The identifier is the procedure name, and the statement body. Some of the identifiers appearing in a 'procedure definition' are special and are called 'formal parameters' of the procedure. Arguments, known as 'actual parameters' may be passed to a called 'procedure'; they are substituted for the formal in the body.
17.	Define activation trees. A recursive procedure p need not call itself directly; p may call another procedure q, which may then call p through some sequence of procedure calls. We can use a tree called an activation tree, to depict the way control enters and leaves activation. In an activation tree a) Each node represents an activation of a procedure, b) The root represents the activation of the main program c) The node for a is the parent of the node for b if and only if control flows from activation a to b, and d) The node for a is to the left of the node for b if and only if the lifetime of a occurs before the lifetime of b.
18.	Write notes on control stack? A control stack is to keep track of live procedure activations. The idea is to push the node for activation onto the control stack as the activation begins and to pop the node when the activation ends.
19.	Write the scope of a declaration? A portion of the program to which a declaration applies is called the scope of that declaration. An occurrence of a name in a procedure is said to be local to the procedure if it is in the scope of a declaration within the procedure; otherwise, the occurrence is said to be nonlocal.
20.	Define binding of names. When an environment associates storage location s with a name x, we say that x is bound to s; the association itself is referred to as a binding of x. A binding is the dynamic counterpart of a declaring.
21.	What is the use of run time storage? The run time storage might be subdivided to hold: a) The generated target code b) Data objects, and c) A counterpart of the control stack to keep track of procedure activation.
22.	What is an activation record? (or) What is frame? Information needed by a single execution of a procedure is managed using a contiguous block of storage called an activation record or frame, consisting of the collection of fields such as a) Return value b) Actual parameters c) Optional control link d) Optional access link e) Saved machine status f) Local data g) Temporaries.
23.	What does the runtime storage hold? Runtime storage holds 1) The generated target code 2) Data objects 3) A counterpart of the control stack to keep track of procedure activations.
24.	What are the various ways to pass a parameter in a function? <ul style="list-style-type: none"> call-by-value call-by-reference call-by-value-result(copy-restore) :this method is a hybrid between call by value and call by references. call-by-name

25.	What are the limitations of static allocation? (Nov/dec 2012) a) The size of a data object and constraints on its position in memory must be known at compile time. b) Recursive procedure is restricted. c) Data structures cannot be created dynamically.																																																												
26.	What is stack allocation? Stack allocation 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 respectively.																																																												
27.	List the fields in activation record .(Nov/Dec 2014) <ul style="list-style-type: none">• Actual parameters• Returned Values• Control link• Access link• Saved machine status• Local data• Temporaries																																																												
28.	What is dangling references? (May/June,2016) Whenever storage can be de-allocated, the problem of dangling references arises. A dangling reference occurs when there is a reference to storage that has been de allocated.																																																												
29.	Constructed a decorated parse tree according to the syntax directed definition, for the following input statement (4+7.5*3)/2. (April/May 2015) <div><pre>graph TD E1["E val=13.25"] --- E2["E val=26.5"] E1 --- T1["T val=2"] E2 --- E3["E val=22.5"] E2 --- P1["+"] E3 --- E4["E val=2.5"] E3 --- M1["*"] E4 --- V1["2.5"] M1 --- T2["T val=3"] T2 --- V2["3"] P1 --- E5["E val=4"] E5 --- V3["4"]</pre></div>																																																												
30.	Write a 3-address code for; x=*y ; a=&x. (April/May 2015) t1:=*y x:=t1 t1:=&x a:=t1																																																												
31.	Place the above code in Triplets and indirect Triplets.(April/May 2015) Triple: <table><tr><td></td><td>Op</td><td>Arg1</td><td>Arg2</td></tr><tr><td>(0)</td><td>*</td><td>Y</td><td></td></tr><tr><td>(1)</td><td>=</td><td>X</td><td>(0)</td></tr></table> Indirect: <table><tr><td></td><td>Op</td><td>Arg1</td><td>Arg2</td></tr><tr><td>11</td><td>*</td><td>Y</td><td></td></tr><tr><td>12</td><td>=</td><td></td><td>(11)</td></tr></table> <table><tr><td></td><td>statement</td></tr><tr><td>(0)</td><td>11</td></tr><tr><td>(1)</td><td>12</td></tr></table> Triple: <table><tr><td></td><td>Op</td><td>Arg1</td><td>Arg2</td></tr><tr><td>(0)</td><td>&</td><td>x</td><td></td></tr><tr><td>(1)</td><td>=</td><td>a</td><td>(0)</td></tr></table> Indirect: <table><tr><td></td><td>Op</td><td>Arg1</td><td>Arg2</td></tr><tr><td>11</td><td>&</td><td>x</td><td></td></tr><tr><td>12</td><td>=</td><td>a</td><td>(11)</td></tr></table> <table><tr><td></td><td>statement</td></tr><tr><td>(0)</td><td>11</td></tr><tr><td>(1)</td><td>12</td></tr></table>		Op	Arg1	Arg2	(0)	*	Y		(1)	=	X	(0)		Op	Arg1	Arg2	11	*	Y		12	=		(11)		statement	(0)	11	(1)	12		Op	Arg1	Arg2	(0)	&	x		(1)	=	a	(0)		Op	Arg1	Arg2	11	&	x		12	=	a	(11)		statement	(0)	11	(1)	12
	Op	Arg1	Arg2																																																										
(0)	*	Y																																																											
(1)	=	X	(0)																																																										
	Op	Arg1	Arg2																																																										
11	*	Y																																																											
12	=		(11)																																																										
	statement																																																												
(0)	11																																																												
(1)	12																																																												
	Op	Arg1	Arg2																																																										
(0)	&	x																																																											
(1)	=	a	(0)																																																										
	Op	Arg1	Arg2																																																										
11	&	x																																																											
12	=	a	(11)																																																										
	statement																																																												
(0)	11																																																												
(1)	12																																																												

32.	<p>Write down syntax directed definition of a simple desk calculator .(Nov/Dec 2016)</p> <p>Production</p> <p>$L \rightarrow En$ $E \rightarrow E1 + T$ $E \rightarrow T$ $T \rightarrow T1 * F$ $T \rightarrow F$ $F \in (E)$ $F \rightarrow \text{digit}$</p> <p>Semantic Rules</p> <p>$\text{print}(E.\text{val})$ $E.\text{val} := E1.\text{val} + T.\text{val}$ $E.\text{val} := T.\text{val}$ $T.\text{val} := T1.\text{val} * F.\text{val}$ $T.\text{val} := F.\text{val}$ $F.\text{val} := E.\text{val}$ $F.\text{val} := \text{digit.lexval}$</p>
33.	<p>List Dynamic Storage allocation techniques. (Nov/Dec 2016)</p> <ul style="list-style-type: none"> Stack allocation – manages the run-time storage as a stack. Heap allocation – allocates and deallocates storage as needed at run time from a data area known as heap
34.	<p>Write the 3 address code for the sentence : $D=(A-B) + (A-C) + (A-C)$</p> <p>3 address code :</p> <p>$T1 = A - B$ $T2 = A - C$ $T3 = T1 + T2$ $D = T3 + T2$</p>
35.	<p>Define binding of names. (April/May,2017)</p> <p>When an environment associates storage location s with a name x, we say that x is bound to s; the association itself is referred to as a binding of x. A binding is the dynamic counterpart of a declaring.</p>
36.	<p>Compare Syntax tree and parse tree</p> <p>A parse tree is a record of the rules (and tokens) used to match some input text whereas a syntax tree records the structure of the input and is insensitive to the grammar that produced it.</p>
37.	<p>Mention the rules for type checking. (April/May,2017)</p> <p>Translation scheme for checking the type of statements:</p> <p>1. Assignment statement:</p> <p style="text-align: center;">$S \rightarrow \text{id} := E \{ S.type := \text{if } \text{id}.type = E.type \text{ then } \text{void} \text{ else } type_error \}$</p> <p>2. Conditional statement:</p> <p style="text-align: center;">$S \rightarrow \text{if } E \text{ then } S1 \{ S.type := \text{if } E.type = \text{boolean} \text{ then } S1.type \text{ else } type_error \}$</p> <p>3. While statement:</p> <p style="text-align: center;">$S \rightarrow \text{while } E \text{ do } S1 \{ S.type := \text{if } E.type = \text{boolean} \text{ then } S1.type \text{ else } type_error \}$</p> <p>4. Sequence of statements:</p> <p style="text-align: center;">$S \rightarrow S1 ; S2 \{ S.type := \text{if } S1.type = \text{void} \text{ and } S1.type = \text{void} \text{ then } \text{void} \text{ else } type_error \}$</p> <p><u>Type checking of functions</u></p> <p>The rule for checking the type of a function application is :</p> <p style="text-align: center;">$E \rightarrow E1 (E2) \{ E.type := \text{if } E2.type = s \text{ and } E1.type = s \rightarrow t \text{ then } t \text{ else } type_error \}$</p>
PART –B CS6660.4	
1.	Explain the concept of syntax directed definition.
2.	Construct parse tree, syntax tree and annotated parse tree for the input string is $5*6+7$;
3.	Explain 1)Synthesized attribute 2)inherited attribute with suitable examples.
4.	Write a syntax directed definition and evaluate $9*3+2$ with parser stack using LR parsing

	method.
5.	Consider the following CFG, $E \rightarrow TR$ $R \rightarrow +TR$ $R \rightarrow -TR$ $R \rightarrow \epsilon$ $T \rightarrow \text{num}$ With translation scheme to generate to generate postfix expression equivalent to the given infix expression which is recognized by above grammar. All actions in the translation should be at the end of each production
6.	Explain the Implementing L-Attributed SDD's
7.	(i) Given the Syntax-Directed Definition below construct the annotated parse tree for the input expression: "int a, b, c". $D \rightarrow T L L.inh = T.type$ $T \rightarrow \text{int } T.type = \text{integer}$ $T \rightarrow \text{float } T.type = \text{float}$ $L \rightarrow L1, id \ L1.inh = L.inh \text{ addType}(id.entry, L.inh)$ $L \rightarrow id \text{ addType}(id.entry, L.inh)$ (ii) Given the Syntax-Directed Definition below with the synthesized attribute val, draw the annotated parse tree for the expression $(3+4) * (5+6)$. $L \rightarrow E \ L.val = E.val$ $E \rightarrow T \ E.val = T.val$ $E \rightarrow E1 + T \ E.val = E1.val + T.val$ $T \rightarrow F \ T.val = F.val$ $T \rightarrow T1 * F \ T.val = T1.val * F.val$ $F \rightarrow (E) \ F.val = E.val$ $F \rightarrow \text{digit} \ F.val = \text{digit.lexval}$
8.	Explain the various structures that are used for the symbol table constructions. (April/may 2012, 2014)
9.	Specify a type checker which can handle expressions, statements and functions. (Nov/Dec, 2017)
10.	Explain the organization of runtime storage in detail. (Nov/Dec, 2017)
11.	What are different storage allocation strategies? Explain. (May/June, 2016) (April/May, 2017)
12.	Explain any 4 issues in storage allocation (4). (April/May 2015)
13.	Give a Syntax directed Definitions to differentiate expressions formed by applying the arithmetic operators + and * to the variable X and constants ; expression : $X*(3*X+X*X)$. (April/May 2015) (8)
14.	For the given program fragment $A[i,j]=B[i,k]$ do the following: (i) Draw the annotated parse tree with the translation scheme to convert to three address code (6) (ii) Write the 3-address code (6) (iii) Determine the address of $A[3,5]$ where , all are integer arrays with size of A as $10*10$ and B as $10*10$ with $k=2$ and the start index position of all arrays is at 1. (assume the base addresses) (4) (April/May 2015)
15.	(i). Apply Back-patching to generate intermediate code for the following input. $x:=2+y;$ If $x < y$ then $x:=x+y;$ repeat $y:=y*2;$ while $x > 10$ do $x:=x/2;$ Write the semantic rule and derive the Parse tree for the given code (12) (ii) What is an Activation Record? Explain how its relevant to the intermediate code generation phase with respect to procedure declarations. (4) (April/May 2015)
16.	Illustrate type checking with necessary diagram (Nov/Dec 2016)
17.	A syntax directed translation scheme that takes string a's b's and c's as input and produces an output the number of substrings in the input string that correspond to the pattern $a(a b)^*c+(a b)^*b$. For example the translation of the input string "abbcabababc" is "3". (Nov/Dec 2016) (1) Write a context-free grammar that generates all strings of a's, b's and c's (2) Semantic attributes for the grammar symbols (3) For each production of the grammar a set of rules for evaluation of the semantic attributes
18.	Discuss specification of a simple type checker. (May/June, 2016) (April/May, 2017)

19.	Construct a syntax directed definition for constructing a syntax tree for assignment statements. $S \rightarrow id := E$ $E \rightarrow E + E$ $E \rightarrow E * E$ $E \rightarrow (E)$ $E \rightarrow id$ (May/June,2016)
Unit- V Code Optimization And Code Generation	
Principal Sources of Optimization-DAG- Optimization of Basic Blocks-Global Data Flow Analysis-Efficient Data Flow Algorithms-Issues in Design of a Code Generator - A Simple Code Generator Algorithm.	
Part – A CS6660.5	
1.	What is meant by optimization? It is a program transformation that made the code produced by compiling algorithms run faster or takes less space.
2.	What are the principle sources of optimization? The principle sources of optimization are, Optimization consists of detecting patterns in the program and replacing these patterns by equivalent but more efficient constructs. The richest source of optimization is the efficient utilization of the registers and instruction set of a machine.
3.	Mention some of the major optimization techniques. <ul style="list-style-type: none"> • Local optimization • Loop optimization • Data flow analysis • Function preserving transformations • Algorithm optimization.
4.	What are the methods available in loop optimization? Code movement - Strength reduction- Loop test replacement- Induction variable elimination
5.	What is the step takes place in peephole optimization? It improves the performance of the target program by examining a short sequence of target instructions. It is called peephole. Replace this instructions by a shorter or faster sequence whenever possible. It is very useful for intermediate representation.
6.	What are the characteristics of peephole optimization? (Nov/Dec 2016) a. Redundant instruction elimination. b. Flow of control optimization c. Algebraic simplifications d. Use of machine idioms
7.	What are the various types of optimization? The various type of optimization is, 1)Local optimization,2)Loop optimization,3)Data flow analysis,4)Global optimization.
8.	List the criteria for selecting a code optimization technique. The criteria for selecting a good code optimization technique are, It should capture most of the potential improvement without an unreasonable amount of effort. It should preserve the meaning of the program. It should reduce the time or space taken by the object program.
9.	What is meant by U-D chaining? It is Use-Definition chaining. It is the process of gathering information about how global data flow analysis can be used id called as use-definition (UD) chaining.
10.	What do you mean by induction variable elimination? It is the process of eliminating all the induction variables , except one when there are two or more induction variables available in a loop is called induction variable elimination.
11.	List any two structure preserving transformations adopted by the optimizer? The structure preserving transformations adopted by the optimizer are, Basic blocks.-Flow graphs.
12.	What are dominators? A node of flow graph is said to be a dominator, i.e one node dominates the other node if every path from the initial node of the flow graph to that node goes through the first node.(d Dom n).when d-node dominates n-node.
13.	What is meant by constant folding? Constant folding is the process of replacing expressions by their value if the value can be computed at complex time.

14.	Define optimizing compilers. List its properties(Nov/Dec 2013) (Nov/Dec, 2017) An optimizing compiler is a compiler that tries to minimize or maximize some attributes of an executable computer program. The most common requirement is to minimize the time taken to execute a program; a less common one is to minimize the amount of memory occupied. Compilers that apply code-improving transformations are called optimizing compilers.
15.	When do you say a transformation of a program is local? A transformation of a program is called local, if it can be performed by looking only at the statement in a basic block.
16.	List the Properties of optimizing compiler. (May/June,2016) (Nov/Dec, 2017) Avoid redundancy Fewer jumps by using <i>straight line code</i> Parallelize Strength reduction Common subexpression elimination Constant folding Induction variable recognition and elimination strength reduction, dead code elimination Loop fission or loop distribution Loop fusion or loop combining Loop inversion Loop-invariant code motion
17.	Write a note on function preserving transformation. A compiler can improve a program by transformation without changing the function it compiles.
18.	List the function –preserving transformation. 1)Common subexpression elimination.2)Copy propagation.3)Dead code elimination.4)Constant folding.
19.	Define common subexpression. An occurrence of an expression E is called a common subexpression if E was previously computed and the values of variables in E have not changed since the previous computation.
20.	What is meant by loop optimization? The running time of a program may be improved if we decrease the number of instructions in a inner loop even if we increase the amount of code outside that loop.
21.	What is data flow analysis? (Nov/Dec 2012) The data flow analysis is the transmission of useful relationships from all parts of the program to the places where the information can be of use.
22.	Define code motion and loop-variant computation. Code motion: It is the process of taking a computation that yields the same result independent of the number of times through the loops and placing it before the loop. Loop –variant computation: It is eliminating all the induction variables, except one when there are two or more induction variables available in a loop
23.	Define loop unrolling with example.(Nov/Dec 2013) Loop overhead can be reduced by reducing the number of iterations and replicating the body of the loop. Example: In the code fragment below, the body of the loop can be replicated once and the number of iterations can be reduced from 100 to 50. for (i = 0; i < 100; i++) g (); Below is the code fragment after loop unrolling. for (i = 0; i < 100; i += 2) { g (); g (); }
24.	What is constant folding?(May/June 2013) Constant folding is the process of replacing expressions by their value if the value can be computed at compile time.

25.	How would you represent the dummy blocks with no statements indicated in global data flow analysis?(May/June 2013) Dummy blocks with no statements are used as technical convenience (indicated as open circles).
26.	What is meant by copy propagation or variable propagation? (April/May,2017) One concerns assignments of the form $f:=g$ called copy statements or copies for short Eg: it means the use of variable V1 in place of V2 1.V1:=V2 2.f:V1+f 3.g:=v2+f-6 In statement 2, V2 can be used in place of V1 by copy propagations. So, 1.V1=V2; 4.f:=V2+f 5.g:V2+f-6 This leads for common sub expression elimination further (V2+f is a common sub expression).
27.	Define busy expressions. An expression E is busy at a program point if and only if <ul style="list-style-type: none"> • An evaluation of E exists along some path P1,P2,...Pn starting at program point P1 and, • No definition of any operand of E exists before its evaluation along the path.
28.	Define code generation. or What role does the target machine play on the code generation phase of the compiler?(April/May 2015). The code generation is the final phase of the compiler. It takes an intermediate representation of the source program as the input and produces an equivalent target program as the output.
29.	Define Target machine. The target computer is byte-addressable machine with four bytes to a word and n-general purpose registers. R0, R1... ..Rn-1. It has two address instructions of the form Op, source, destination in which Op is an op-code, and source and destination are data fields.
30.	How do you calculate the cost of an instruction? The cost of an instruction can be computed as one plus cost associated with the source and destination addressing modes given by added cost. <div style="margin-left: 100px;"> MOV R0,R1 cost is 1 MOV R1,M cost is 2 SUB 5(R0),*10(R1) cost is 3 </div>
31.	Define basic block. (Nov/Dec 2013) A basic block contains sequence of consecutive statements, which may be entered only at the beginning and when it is entered it is executed in sequence without halt or possibility of branch.
32.	What are the rules to find “ leader” in basic block? <ul style="list-style-type: none"> • It is the first statement in a basic block is a leader. • Any statement which is the target of a conditional or unconditional goto is a leader. • Any statement which immediately follows a conditional goto is a leader.
33.	Define flow graph.(Nov/Dec 2013) Relationships between basic blocks are represented by a directed graph called flow graph.
34.	What do you mean by DAG? (May/June,2016) (Nov/Dec, 2017) It is Directed Acyclic Graph. In this common sub expressions are eliminated. So it is a compact way of representation.
35.	List the advantages of DAGs (Nov/Dec 2012) (Nov/Dec, 2017) It automatically detects common sub expression. We can determine which identifiers have their values used in the block. We can determine which statements compute values, and which could be used outside the block. It reconstruct a simplified list of quadruples taking advantage of common sub expressions and not performs assignments of the form $a=b$ unless necessary.
36.	What is meant by registers and address descriptor? Register descriptor: It contains information' s about,1.What registers are currently in use. 2.What registers are empty. Address descriptor: It keeps track of the location where the current value of the name can be found at run time.

37.	Define live variable.(Nov/Dec 2012) A variable is live at a point in a program if its value can be used subsequently.
38.	What are the different storage allocation strategies? (Nov/Dec, 2017) 1) Static allocation.-It lays out storage for all data objects at compile time. 2) Stack allocation.-It manages the runtime storage as a stack. 3) Heap allocation.-It allocates and de-allocates storage as needed at runtime from a data area.
39.	What is the use of Next-use information?(Nov/Dec 2013) <ul style="list-style-type: none"> • If a register contains a value for a name that is no longer needed, we should re-use that register for another name (rather than using a memory location) • So it is useful to determine whether/when a name is used again in a block • Definition: Given statements i, j, and variable x, <ul style="list-style-type: none"> – If i assigns a value to x, and – j has x as an operand, and – No intervening statement assigns a value to x, – Then j uses the value of x computed at i.
40.	How liveness variable calculated.(April/May 2015) A variable is live if it holds a value that will/might be used in the future. The representation of the program that we use for liveness analysis (determining which variables are live at each point in a program), is a control flow graph. The nodes in a control flow graph are basic statements (instructions). There is an edge from statement x to statement y if x can be immediately followed by y (control flows from x to y).
41.	Write the algorithm that orders the DAG nodes for generating optimal target code?.(April/May 2015) <pre> (1) while unlisted interior nodes remain do begin (2) select an unlisted node n, all of whose parents have been listed ; (3) list n; (4) while the leftmost child m of n has no unlisted parents and is not a leaf do /* since n was just listed , m is not yet listed*/ begin (5) list m; (6) n = m end end end </pre>
42.	Identify the constructs for optimization in basic block .(Nov/Dec 2016) There are two types of basic block optimizations. They are : Structure -Preserving Transformations Algebraic Transformations
43.	Define basic block. (April/May,2017) A basic block contains sequence of consecutive statements , which may be entered only at the beginning and when it is entered it is executed in sequence without halt or possibility of branch .
PART –B CS6660.5	
1.	Explain the principle sources of optimization in detail. (May/June,2016) (April/May,2017) (Nov/Dec, 2017)
2.	Explain optimization of basic blocks. (April/May,2017)
3.	Write about data flow analysis of structural programs.
4.	Optimize the following code using various optimization techniques: <pre> i=1,s=0; for(i=1;i<=3;i++) for(j=1;j<=3;j++) c[i][j]=c[i][j]+a[i][j]+b[i][j]; </pre>
5.	(i)Explain the issues in design of code generator. (May/June,2016) (April/May,2017) (Nov/Dec, 2017) (ii)Explain peephole optimization.
6.	Explain the simple code generator with a suitable example (May/June,2016)
7.	Write detailed notes on Basic blocks and flow graphs.
8.	Define a Directed Acyclic Graph. Construct a DAG and write the sequence of instructions for the expression $a+a*(b-c)+(b-c)*d$. (May/June 2014)

9.	(i) Write the code generation algorithm using dynamic programming and generate code for the statement $x=a/(b-c)-s*(e+f)$ [Assume all instructions to be unit cost] (12) (ii) What are the advantages of DAG representation? Give example.(4) (April/May 2015)
10.	(i)Write the procedure to perform Register Allocation and Assignment with Graph Coloring.(8) (ii)Construct DAG and optimal target code for the expression $X=((a+b)/(b-c))-(a+b)*(b-c)+f$. (April/May 2015).(8)
15.	(i)Perform analysis of available expressions on the following code by converting into basic blocks and compute global common sub expression elimination. (ii) Explain Loop optimization in details and apply it to the code (10) <pre> I:=0 A:=n-3 If i<a then loop else end Label loop B:=i_4 E:=p+b D:=m[c] E:=d-2 F:=I-4 G:=p+f M[g]:=e I:=i+1 A:=n-3 </pre> (iii)What is the optimization technique applied on procedure calls? Explain with example .(6) (April/May 2015)
16.	(i) Write an algorithm for constructing natural loop of back edge (ii) Explain any four issues that crop up when designing a code generator (Nov/Dec 2016)
17.	Explain global data flow analysis with necessary equations (Nov/Dec 2016)
18.	Explain about parameter passing mechanism (April/May,2017)
19.	Construct the DAG for the following Basic Block. (April/May,2017) <pre> 1. t1:=4*i 2. t2:=a[t1] 3. t3:=4*i 4. t4:=b[t3] 5. t5:=t2*t4 6. t6:=prod+t5 7. prod:=t6 8. t7:=i+1 9. i:=t7 10. if i<=20 goto (1) 11. </pre>
