JEPPIAAR ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CS6660

COMPILER DESIGN

Question Bank

III YEAR A & B / BATCH : 2016 -20

Vision of Institution

To build Jeppiaar Engineering College as an Institution of Academic Excellence in Technical education and Management education and to become a World Class University.

Mission of Institution

M1	To excel in teaching and **learning, research and innovation** by promoting the principles of scientific analysis and creative thinking
M2	To participate in the production, **development and dissemination of knowledge** and interact with **national and international communities**
M3	To equip students with **values, ethics and life skills** needed to enrich their lives and enable them to meaningfully contribute to the **progress of society**
M4	To prepare students **for higher studies and lifelong learning**, enrich them with the **practical and entrepreneurial skills** necessary to excel as future professionals and contribute to **Nation's economy**

Program Outcomes (POs)

PO1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations
PO4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Vision of Department

To emerge as a globally prominent department, developing ethical computer professionals, innovators and entrepreneurs with academic excellence through quality education and research**.**

Mission of Department

M1	To create **computer professionals** with an ability to identify and **formulate the engineering problems** and also to provide **innovative solutions** through **effective teaching learning process.**
M2	To **strengthen the core-competence** in computer science and engineering and to create an ability to **interact** effectively with industries.
M3	To produce engineers with good professional skills, **ethical values** and life skills for the **betterment of the society.**
M4	To encourage students towards **continuous and higher level learning** on technological advancements and provide a platform for **employment and self-employment.**

Program Educational Objectives (PEOs)

PEO1	**To address the real time complex engineering problems using innovative approach with strong core computing skills.**
PEO2	**To apply core-analytical knowledge and appropriate techniques and provide solutions to real time challenges of national and global society**
PEO3	**Apply ethical knowledge for professional excellence and leadership for the betterment of the society.**
PEO4	**Develop life-long learning skills needed for better employment and entrepreneurship**

Program Specific Outcomes (PSOs)

Students will be able to

PSO1	An ability to understand the core concepts of computer science and engineering and to enrich problem solving skills to analyze, design and implement software and hardware based systems of varying complexity.
PSO2	To interpret real-time problems with analytical skills and to arrive at cost effective and optimal solution using advanced tools and techniques.
PSO3	An understanding of social awareness and professional ethics with practical proficiency in the broad area of programming concepts by lifelong learning to inculcate employment and entrepreneurship skills.

SYLLABUS

UNIT I INTRODUCTION TO COMPILERS 5

Translators-Compilation and Interpretation-Language processors -The Phases of Compiler-ErrorsEncountered in Different Phases-The Grouping of Phases-Compiler Construction Tools - Programming Language basics.

UNIT II LEXICAL ANALYSIS 9

Need and Role of Lexical Analyzer-Lexical Errors-Expressing Tokens by Regular Expressions-Converting Regular Expression to DFA- Minimization of DFA-Language for Specifying LexicalAnalyzers-LEX-Design of Lexical Analyzer for a sample Language.

UNIT III SYNTAX ANALYSIS 10

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 .

UNIT IV SYNTAX DIRECTED TRANSLATION & RUN TIME ENVIRONMENT 12

Syntax directed Definitions-Construction of Syntax Tree-Bottom-up Evaluation of S-AttributeDefinitions- 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 FORTAN.

UNIT V CODE OPTIMIZATION AND CODE GENERATION 9

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.

TEXTBOOK:

1. Alfred V Aho, Monica S. Lam, Ravi Sethi and Jeffrey D Ullman, “Compilers – Principles,Techniques and Tools”, 2nd Edition, Pearson Education, 2007.

REFERENCES:

1. Randy Allen, Ken Kennedy, “Optimizing Compilers for Modern Architectures: A Dependence-based Approach”, Morgan Kaufmann Publishers, 2002.

2. Steven S. Muchnick, “Advanced Compiler Design and Implementation, “Morgan Kaufmann Publishers - Elsevier Science, India, Indian Reprint 2003.

3. Keith D Cooper and Linda Torczon, “Engineering a Compiler”, Morgan Kaufmann Publishers Elsevier Science, 2004.

4. Charles N. Fischer, Richard. J. LeBlanc, “Crafting a Compiler with C”, Pearson Education, 2008.

Course Outcomes (COs)

C311.1	Examining the functioning of compilation process
C311.2	Evaluating the role of tokens in analysis phase of compiler.
C311.3	Evaluating the role of Parser in a compiler.
C311.4	Summarize the semantic action taken by the compiler during semantic phase of the compiler.
C311.5	Analyse the method of finding code generation and code optimization techniques in compilation.

INDEX

UNIT I INTRODUCTION

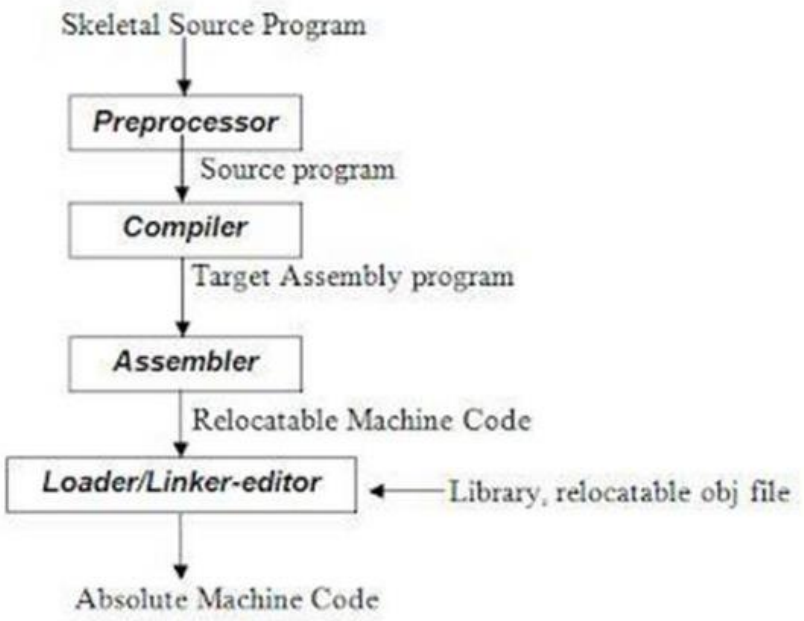
Translators-Compilation and Interpretation-Language processors -The Phases of Compiler-ErrorsEncountered in Different Phases-The Grouping of Phases-Compiler Construction Tools - Programming Language basics.

S. No.	Question	Course Outcome	Blooms Taxanomy Level
1	**Define Token.APRIL/MAY2011,MAY/JUNE 2013** The token can be defined as a meaningful group of characters over the character set of the programming language like identifiers, keywords, constants and others.	**C311.1**	BTL1
2	**Define Symbol Table.NOV/DEC 2016, MAY/JUNE 2014** A Symbol table is a data structure 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.	**C311.1**	BTL1
3	**What is a Complier? MAY/JUNE 2007** A Complier is a program that reads a program written in one language-the source language-and translates it in to an equivalent program in another language-the target language. As an important part of this translation process, the compiler reports to its user the presence of errors in the source program.	**C311.1**	BTL1
4	**What is an interpreter? NOV/DEC 2017** Interpreter is a program which converts source language to machine language line by line. No intermediate object code is generated, hence are memory efficient. Ex: Python, COBOL.	**C311.1**	BTL1
5	**What do you mean by Cross-Compiler? NOV/DEC 2017** **A cross compiler** is a compiler capable of creating executable code for a platform other than the one on which the compiler is run. (ie). A compiler may run on one machine and produce target code for another machine.	**C311.1**	BTL1
6	**What are the cousins of compiler? APRIL/MAY2004,APRIL/MAY2005,APRIL/MAY 2012,MAYY/JUNE 2013, MAY/JUNE 2012, APRIL/MAY**		

	2017 The following are the cousins of compilers i. Preprocessors ii. Assemblers iii. Loaders **iv.** Link editors.	**C311.1**	BTL1
7	**What are the four obsoletes of quality What are the main two parts of compilation? What are they performing? MAY/JUNE 2016 , APRIL/MAY 2010, APRIL/MAY 2017, APRIL/MAY 2018** The two main parts are -**Analysis** part breaks up the source program into constituent pieces and creates . An intermediate representation of the source program. -**Synthesis** part constructs the desired target program from the intermediate representation.	**C311.1**	BTL1
8	**What are an assembler and interpreter?** **APRIL/MAY 2011** Assembler is a program, which converts the assembly language in to machine language. Interpreter is a program which converts source language into machine language line by line.	**C311.1**	BTL1
9	**State any two reasons as to why phases of compiler should be grouped. MAY/JUNE 2014** The reasons for grouping, 1. Implementation purpose 2. Compiler work is based on two things; one is based on language other one is based on machine.	**C311.1**	BTL1
10	**State some software tools that manipulate source program?** i. Structure editors ii. Pretty printers iii. Static checkers iv. Interpreters.	**C311.1**	BTL1

11	**What is a Structure editor?** A structure editor takes as input a sequence of commands to build a source program .The structure editor not only performs the text creation and modification functions of an ordinary text editor but it also analyzes the program text putting an appropriate hierarchical structure on the source program.	**C311.1**	BTL1
12	**What are a Pretty Printer and Static Checker?** A Pretty printer analyses a program and prints it in such a way that the structure of the program becomes clearly visible. • A static checker reads a program, analyses it and attempts to discover potential bugs with out running the program.	**C311.1**	BTL1
13	**How many phases does analysis consists?** Analysis consists of three phases i .Linear analysis ii .Hierarchical analysis iii. Semantic analysis	**C311.1**	BTL1
14	**What happens in Hierarchical analysis?** This is the phase in which characters or tokens are grouped hierarchically in to nested collections with collective meaning.	**C311.1**	BTL1
15	**What happens in Semantic analysis?** This is the phase in which certain checks are performed to ensure that the components of a program fit together meaningfully	**C311.1**	BTL1
16	**State some compiler construction tools?** **NOV/DEC 2016, APRIL /MAY 2008** v. Parse generator vi. Scanner generators vii. Syntax-directed viii. translation engines ix. Automatic code generator x. . Data flow engines.	**C311.1**	BTL1

17	**What is a Loader? What does the loading process do?** A Loader is a program that performs the two functions i. Loading ii .Link editing The process of loading consists of taking relocatable machine code, altering the relocatable address and placing the altered instructions and data in memory at the proper locations.	**C311.1**	BTL1
18	**What does the Link Editing does?** Link editing: This allows us to make a single program from several files of relocatable machine code. These files may have been the result of several compilations, and one or more may be library files of routines provided by the system and available to any program that needs them	**C311.1**	BTL1
19	**What is a preprocessor? Nov/Dev 2004** A preprocessor is one, which produces input to compilers. A source program may be divided into modules stored in separate files. The task of collecting the source program is sometimes entrusted to a distinct program called a preprocessor. The preprocessor may also expand macros into source language statements.	**C311.1**	BTL1
20	**State some functions of Preprocessors** i) Macro processing ii) File inclusion iii) Relational Preprocessors iv) Language extensions	**C311.1**	BTL1
21	**State the general phases of a compiler** i) Lexical analysis ii) Syntax analysis iii) Semantic analysis iv) Intermediate code generation v) Code optimization vi) Code generation	**C311.1**	BTL1
22	**What is an assembler?** Assembler is a program, which converts the source language in to assembly language.	**C311.1**	BTL1
23	**Depict diagrammatically how a language is processed.** **MAY/JUNE 2016**		

		**C311.1**	BTL1
24	**What is linear analysis?** Linear analysis is one 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. Also called lexical analysis or scanning.	**C311.1**	BTL1
25	**What are the classifications of a compiler?** Compilers are classified as: · Single- pass Multi-pass · Load-and-go · Debugging or optimizing	**C311.1**	BTL1
26	**List the phases that constitute the front end of a compiler.** The front end consists of those phases or parts of phases that depend primarily on the source language and are largely independent of the target machine. These include • · Lexical and Syntactic analysis • · The creation of symbol table • · Semantic analysis • · Generation of intermediate code A certain amount of code optimization can be done by the front end as well. Also includes error handling that goes along with each of these phases.	**C311.1**	BTL1

27	**Mention the back-end phases of a compiler.** The back end of compiler includes those portions that depend on the target machine and generally those portions do not depend on the source language, just the intermediate language. These include • Code optimization • Code generation, along with error handling and symbol- table operations.	**C311.1**	BTL1
28	**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.	**C311.1**	BTL1
29	**What are the advantages of a interpreter ?** Modification of user program can be easily made and implemented as execution proceeds. Type of object that denotes a various may change dynamically. Debugging a program and finding errors is simplified task for a program used for interpretation. The interpreter for the language makes it machine independent.	**C311.1**	BTL1
30	**What are the disadvantages of a interpreter** The execution of the program is *slower*. *Memory* consumption is more	**C311.1**	BTL1
31	**What are the components of a Language processing system?** Preprocessor Compiler Assembler Loader-Linker editor	**C311.1**	BTL1
32	**Mention the list of compilers.** BASIC compilers C# compilers	**C311.1**	BTL1

	C compilers C++ compilers COBOL compilers		
33	**What is the main difference between phase and pass of a compiler?** A phase is a sub process of the compilation process whereas combination of one or more phases into a module is called pass.	**C311.1**	BTL1
34	**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.	**C311.1**	BTL1
35	**How will you group the phases of compiler?** **Front and Back Ends:** The phases are collected into a front end and a back end. **Front End: C**onsists of those phases or parts of phases that depend primarily on the source language and are largely independent of target machine **Back End:** Includes those portions of the compiler that depend on the target machine and these portions do not depend on the source language. **Passes:** It is common for several phases to be grouped into one pass, and for the activity of these phases to be interleaved during the pass.	**C311.1**	BTL1
36	**Why lexical and syntax analyzers are separated out?** Reasons for separating the analysis phase into lexical and syntax analyzers: Simpler design. Compiler efficiency is improved. Compiler portability is enhanced.	**C311.1**	BTL1
37	**Mention the basic issues in parsing.** There are two important issues in parsing. Specification of syntax Representation of input after parsing.	**C311.1**	BTL1
38	**Define parser.** Hierarchical analysis is one in which the tokens are grouped hierarchically into nested collections with collective meaning. Also termed as Parsing.	**C311.1**	BTL1
39	**What happens in linear analysis?** This is the phase in which the stream of characters making up the source program is read from left to right and grouped in to tokens that are sequences of characters having collective meaning.	**C311.1**	BTL1
40	**Give the properties of intermediate representation?** a) It should be easy to produce. b) It should be easy to translate into the target program	**C311.1**	BTL1

41	**What are the tools available in analysis phase?** •Structure editors •Pretty printer •Static checkers •Interpreters.	**C311.1**	BTL1
42	**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	**C311.1**	BTL1
43	**What are the functions performed in synthesis phase?** •Intermediate code generation •Code generation •Code optimization	**C311.1**	BTL1
45	**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.	**C311.1**	BTL1
46	**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.	**C311.1**	BTL1
47	**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.	**C311.1**	BTL1
48	**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.	**C311.1**	BTL1
49	**What Are The Various Types Of Intermediate Code Representation?** There are mainly three types of intermediate code representations. 1. Syntax tree 2. Postfix	**C311.1**	BTL1

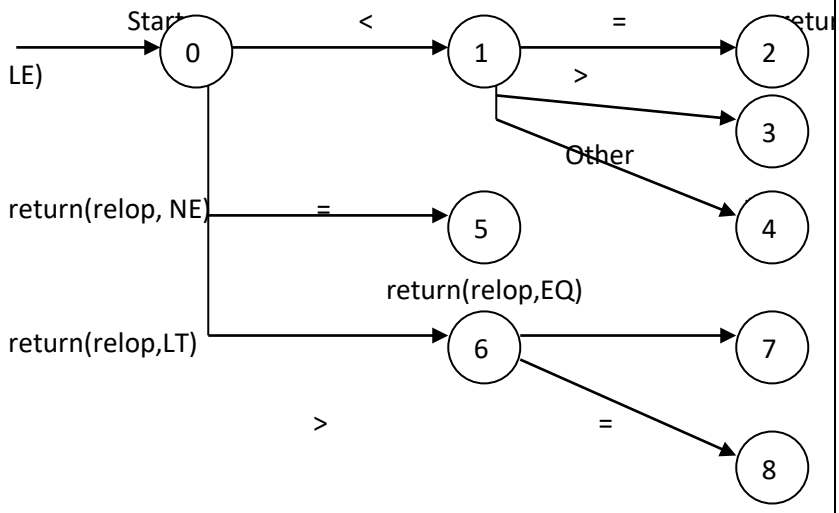
	3. Three address code		
50	**Define Token.** Sequence of characters that have a collective meaning.	**C311.1**	BTL1
	PART B		
1	What are the various phases of a compiler? Explain each phase in detail by using the input "a=(b+c)*(b+c)*2". (Page No.10) APRIL/MAY 2011, APRIL/MAY 2012, MAY/JUNE 2014, MAY/JUNE 2013, NOV/DEC 2016, NOV/DEC 2017	**C311.1**	BTL5
2	Explain the various Compiler Construction Tools. (Page No.22) APRIL/MAY 2011, APRIL/MAY 2012, NOV/DEC2014,MAY/JUNE 2015, NOV/DEC 2016, APRIL/MAY 2017, NOV/DEC 2017	**C311.1**	BTL5
3	What are the cousins of a Compiler? Explain them in detail. Explain the need for grouping of phases of compiler. . (Page No.16) NOV/DEC 2014, APRIL/MAY 2017	**C311.1**	BTL5
4	Write about the Error handling in different phases. (OR) Explain various Error encountered in different phases of compiler. (Page No.11) MAY/JUNE 2016, NOV/DEC 2016	**C311.1**	BTL5
5	Draw the transition diagram for relational operators and unsigned numbers.(Page No.131&133) APRIL/MAY 2017	**C311.1**	BTL2
6	For the following expression MAY/JUNE 2016, APRIL/MAY 2017 Position: =initial+ rate*60.Write down the output after each phase. (Page No.13)	**C311.1**	BTL2
7	i) Explain language processing system with neat diagram. (Page No.5) MAY/JUNE 2016 ii) Explain the need for grouping of phases (Page No. 20)	**C311.1**	BTL5

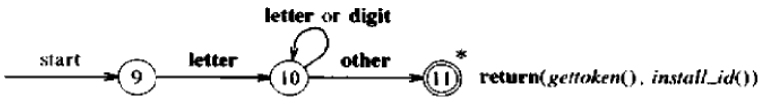
	MAY/JUNE 2016, NOV/DEC 2016		
8	i). Analyze the given expressions 4:*+=cba with different phases of the compiler (Page No.10) (ii). Classify the concepts of compiler and Interpreter. (Page No.2)	**C311.1**	BTL 4&3
9	Generalize the important terminologies used in programming language basics (Page No.25)	**C311.1**	BTL 6
10	(i).How to solve the source program to target machine code by using language processing system. (Page No.4) (ii).Write in detail about the cousins of the compiler. (Page No.1-5)	**C311.1**	BTL 3
11	(i).Describe the errors encountered in different phases of compiler.(Page No.194) (ii).Explain the functions of Preprocessor. (Page No.1-3)	**C311.1**	BTL 2
12	(i).Tell the various phases of the compiler and examine with programs segment (Page No.10) (ii).Discuss in detail about symbol table. (Page No.5)	**C311.1**	BTL 1
13	Describe the topic on (Page No.12) (i) Parser Generators (ii) Syntax directed translation engines (iii)Scanner Generators.	**C311.1**	BTL 1
14	What is meant by lexical analysis? Identify the lexemes that makeup the token in the following program segment.indicate the correspond token and pattern. Void swap(int i, int j) { int t; t = i ; i = j ; j = t ; } **REFER NOTES**	**C311.1**	BTL 6
15	(i).Give the Properties of intermediate representation. (Page No.91) (ii).Discuss the concepts of Parameter pass Mechanisms. (Page No.33 to 35)	**C311.1**	BTL 2

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.

S. No.	Question	Course Outcome	Blooms Taxanomy Level
1	**Write a grammar for branching statements. MAY/JUNE 2016** **Stmt-> if expr then stmt** **| if expr then stmt else stmt** **| €** **expr-> term relop term** **| term** **term -> id**	**C311.2**	BTL1
2	**What is a lexeme? Define a regular set. APRIL/MAY2011,MAY/JUNE2013 MAY/JUNE2014, NOV/DEC 2017** 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.	**C311.2**	BTL1
3	**What is a regular expression? State the rules, which define regular expression? MAY/JUNE 2007,APRIL/MAY 2018** Regular expression is a method to describe regular Language Rules: 1) €-is a regular expression that denotes {€} that is the set containing the empty string 2) If a is a symbol in ∑,then a is a regular expression that	**C311.2**	BTL1

	denotes {a} 3) Suppose r and s are regular expressions denoting the languages L(r) and L(s) Then, a) (r)/(s) is a regular expression denoting L(r) U L(s). b) (r)(s) is a regular expression denoting L(r)L(s) c) (r)* is a regular expression denoting L(r)*. d) (r) is a regular expression denoting L(r).		
4	**What are the Error-recovery actions in a lexical analyzer? APRIL/MAY 2012, MAY/JUNE 2013,APRIL/MAY 2015,APRIL/MAY 2018** Deleting an extraneous character Inserting a missing character Replacing an incorrect character by a correct character Transposing two adjacent characters	**C311.2**	BTL1
5	**Draw a transition diagram to represent relational operators**. **NOV/DEC 2007** 	**C311.2**	BTL2
6	**What are the issues to be considered in the design of lexical analyzer? MAY/JUNE 2009** How to Precisely Match Strings to Tokens How to Implement a Lexical Analyzer	**C311.2**	BTL1

7	**Write short notes on buffer pair**. **APRIL/MAY 2008** Lexical analyzer will detect the tokens from the source language with the help of input buffering. The reason is, the lexical analyzer will scan the input character by character, it will increase the cost file read operation. So buffering is used. The buffer is a pair in which each half is equal to system read command.	**C311.2**	BTL1
8	**How the token structure is is specified? Or Define Patterns.** **APRIL/MAY 2010, MAY/JUNE 2013** Token structure is specified with the help of Pattern. The pattern can be described with the help of Regular Expression	**C311.2**	BTL1
9	**What is the role of lexical analyzer? NOV/DEC 2011, NOV/DEC 2014, NOV/DEC 2017** Its main task is to read input characters and produce as output a sequence of tokens that parser uses for syntax analysis. Additionally task is removing blank, new line and tab characters	**C311.2**	BTL1
10	**Give the transition diagram for an identifier.** **NOV/DEC 2011**  **Fig. 3.13. Transition diagram for identifiers and keywords.**	**C311.2**	BTL2
11	**Why is buffering used in lexical analysis? What are the commonly used buffering methods?** **MAY/JUNE 2014** Lexical analyzer needs to get the source program statement from character by character, without buffering it is difficult to synchronize the speed between the read write hardware and the	**C311.2**	BTL1

	lexical program. Methods are two way buffering and sentinels.		
12	**Write regular expression to describe a languages consist of strings made of even numbers a and b.** **NOV/DEC 2014** ((a+b)(a+b))*	**C311.2**	BTL1
13	**What are the various parts in LEX program? APRIL/MAY 2017** Lex specification has three parts declarations %% pattern specifications %% support routines	**C311.2**	BTL1
14	**Write regular expression for identifier and number. NOV/DEC 2012, APRIL/MAY 2017** For identifier (a-z)((a-z)|(0-9))*other symbols For numbers (0-9)(0-9)*	**C311.2**	BTL1
15	**What is the need for separating the analysis phase into lexical analysis and parsing? (Or) What are the issues of lexical analyzer?** • 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. • Compiler efficiency is improved • Compiler portability is enhanced	**C311.2**	BTL1
16	**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	**C311.2**	BTL1

	collective meaning.		
17	**What is a sentinel? What is its usage?** **<u>April/May 2004</u>** 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.	**C311.2**	BTL1
18	**What is a regular expression? State the rules, which define regular expression?** Regular expression is a method to describe regular language **Rules:** 1) ε-is a regular expression that denotes {ε} that is the set containing the empty string 2) If a is a symbol in ∑,then a is a regular expression that denotes {a} 3) Suppose r and s are regular expressions denoting the languages L(r) and L(s) Then, a) (r)/(s) is a regular expression denoting L(r) U L(s). b) (r)(s) is a regular expression denoting L(r)L(s) c) (r)* is a regular expression denoting L(r)*. d) (r) is a regular expression denoting L(r).	**C311.2**	BTL1
19	**Construct Regular expression for the language L= {w ε{a,b}/w ends in abb}** Ans: {a/b}*abb.	**C311.2**	BTL2
20	**What is recognizer?** Recognizers are machines. These are the machines which accept the strings belonging to certain language. If the valid strings of such language are accepted by the machine then it is said that the corresponding language is accepted by that machine, otherwise it is rejected.	**C311.2**	BTL1
21	**Differentiate tokens, patterns, lexeme.**		

	NOV/DEC 2016 · Tokens- Sequence of characters that have a collective meaning. · Patterns- There is a set of strings in the input for which the same token is produced as output. This set of strings is described by a rule called a pattern associated with the token · Lexeme- A sequence of characters in the source program that is matched by the pattern for a token.	**C311.2**	BTL2
22	**List the operations on languages.** **MAY/JUNE 2016** · **Union –** L U M ={s | s is in L or s is in M} · **Concatenation** – LM ={st | s is in L and t is in M} · **Kleene Closure –** L* (zero or more concatenations of L) · **Positive Closure –** L+ (one or more concatenations of L)	**C311.2**	BTL1
23	**Write a regular expression for an identifier.** An identifier is defined as a letter followed by zero or more letters or digits. The regular expression for an identifier is given as **letter (letter | digit)***	**C311.2**	BTL1
24	**Mention the various notational short hands for representing regular expressions.** · One or more instances (+) · Zero or one instance (?) · Character classes ([abc] where a,b,c are alphabet symbols denotes the regular expressions a | b | c.) · Non regular sets	**C311.2**	BTL1
25	**What is the function of a hierarchical analysis?** Hierarchical analysis is one in which the tokens are grouped hierarchically into nested collections with collective meaning. Also termed as Parsing.	**C311.2**	BTL1
26	**What does a semantic analysis do?** Semantic analysis is one in which certain checks are performed to ensure that components of a program fit together meaningfully. Mainly performs type checking.	**C311.2**	BTL1

27	**What is a lexical error ?** Lexical errors are the errors thrown by your lexer when unable to continue. Which means that there's no way to recognise a *lexeme* as a valid *token* for you lexer. Syntax errors, on the other side, will be thrown by your scanner when a given set of already recognised valid tokens don't match any of the right sides of your grammar rules.	**C311.2**	BTL1
28	**State the conventions of a transition diagram.** Certain states are said to be accepting or final .These states indicates that a lexeme has been found, although the actual lexeme may not consist of all positions b/w the lexeme Begin and forward pointers we always indicate an accepting state by a double circle. In addition, if it is necessary to return the forward pointer one position, then we shall additionally place a * near that accepting state. One state is designed the state ,or initial state ., it is indicated by an edge labeled "start" entering from nowhere .the transition diagram always begins in the state before any input symbols have been used.	**C311.2**	BTL1
29	**What is DFA?** • A Deterministic Finite Automaton (DFA) is a special form of a NFA. • No state has ε- transition • For each symbol a and state s, there is at most one labeled edge a leaving s. i.e. transition function is from pair of state-symbol to state (not set of states).	**C311.2**	BTL1
30	**Define NFA**. A NFA accepts a string x, if and only if there is a path from the starting state to one of accepting states such that edge labels along this path spell out x. ε- transitions are allowed in NFAs. In other words, we can move from one state to another one without consuming any symbol.	**C311.2**	BTL1
31	**What is a finite automata?**	**C311.2**	BTL1

	A *recognizer* for a language is a program that takes a string x, and answers "yes" if x is a sentence of that language, and "no" otherwise. • We call the recognizer of the tokens as a *finite automaton*. • A finite automaton can be: *deterministic (DFA)* or *non-deterministic (NFA).*		
32	**Differentiate NFA and DFA. NOV/DEC 2017** NFA: NFA or Non Deterministic Finite Automaton is the one in which there exists many paths for a specific input from current state to next state. Transition Function δ : Q X ∑ → 2^Q DFA: Deterministic Finite Automaton is a FA in which there is only one path for a specific input from current state to next state. There is a unique transition on each input symbol. Transition Function δ : Q X ∑ → Q	**C311.2**	BTL2
33	**What are the rules that define the regular expression over alphabet? (Or) List the rules that form the BASIS.** **NOV/DEC 2016** • € is a regular expression denoting { € }, that is, the language containing only the empty string. • For each 'a' in Σ, is a regular expression denoting { a }, the language with only one string consisting of the single symbol 'a' . • If R and S are regular expressions, then (R) | (S) means L(r) U L(s) R.S means L(r).L(s) R* denotes L(r*)	**C311.2**	BTL1
34	**Construct Regular expression for the language L= {w ε{0,1}/w**		

	consists of odd number of 0's} RE = 0(001)*11	**C311.2**	BTL1
35	**Give the parts of a string?** Prefix of s, suffix of s, substring of s, proper prefix, proper suffix, proper substring and subsequence of s.	**C311.2**	BTL1
36	**What are the implementations of lexical analyzer?** a) Use a lexical analyzer generator, such as Lex compiler, to produce the lexical analyzer from a regular expression based specification. b) Write the lexical analyzer in a conventional systems-programming language using the I/O facilities of that language to read the input. c) Write the lexical analyzer in assembly language and explicitly manage the reading of input.	**C311.2**	BTL1
37	**Define the length of a string?** It is the number of occurrences of symbols in string, "s" denoted by |s|. Example: s=abc, |s| =3.	**C311.2**	BTL1
38	**Define regular set?** A language denoted by a regular expression is said to be a regular set.	**C311.2**	BTL1
39	**Define character class with example.** The notation [abc] where a, b, c are alphabet symbols denotes the regular expression a/b/c. **Example:** [A-z] = a | b | c | -------| z Regular expression for identifiers using character classes [a – z A – Z] [A – Z a – z 0 – 9] *	**C311.2**	BTL1
40	**Write the R.E. for the set of statements over {a,b,c} that contain no two consecutive b's** **Answer: (B/c) (A/c/ab/cb) ***	**C311.2**	BTL2
41	**Describe the language denoted by the R.E. (0/1)*0(0/1)(0/1)** **Answer:** The set of all strings of 0's and 1's with the third symbol from the right end is 0.	**C311.2**	BTL1
42	**What are the tasks in lexical analyzer?** • One task is stripping out from the source program comments and white space in the form of blank, tab, new	**C311.2**	BTL1

	line characters. • Another task is correlating error messages from the compiler with the source program.		
43	**Define parser.** Hierarchical analysis is one in which the tokens are grouped hierarchically into nested collections with collective meaning. Also termed as Parsing.	**C311.2**	BTL1
44	**Write the R.E. for the set of statements over {a,b,c} that contain an even no of a's.** **Ans: ((b/c)* a (b/c) * a)* (b/c)***	**C311.2**	BTL1
45	**Describe the language denoted by the following R.E. 0(0/1)*0** **Answer:** The set of all strings of 0's and 1's starting and ending with 0.	**C311.2**	BTL1
46	**Describe the language denoted by the following R.E.** (00/11)*((01/10)(00/11)*(01/10)(00/11)*) **Answer:** The set of all strings of 0's and 1's with even number of 0's	**C311.2**	BTL1
47	**Draw the NFA for (0/1)***	**C311.2**	BTL2
48	**Draw the DFA for a(abb)***	**C311.2**	BTL2
49	**Draw the Deterministic Finite Automata for the language Even no.of 0's and 1's.**	**C311.2**	BTL2
50	**Draw the Non Deterministic Finite Automata for the language Odd no.of 0's and 1's.**	**C311.2**	BTL2
	PART B		
1	Explain Input Buffering with example. (Page No.88) NOV/DEC 2011	**C311.2**	BTL5
2	Explain the role of Lexical Analyzer in detail with necessary examples. (Page No.84) MAY/JUNE 2016, MAY/JUNE 2013, APRIL/MAY 2011, NOV/DEC 2016 Discuss how finite automata is used to represent tokens and perform lexical analysis with examples. NOV/DEC 2016	**C311.2**	BTL5

3	Explain the specification of tokens. (Page No.92) MAY/JUNE 2016,MAY/JUNE 2013, APRIL/MAY 2008, NOV/DEC 2014	**C311.2**	BTL5
4	Elaborate in detail the recognition of tokens. (Page No.98) APRIL/MAY 2012, NOV/DEC 2014	**C311.2**	BTL6
5	Write an algorithm to convert NFA to DFA and minimize DFA. Give an example.NOV/DEC 2017 (Page No.140)	**C311.2**	BTL5
6	What are the issues in Lexical analysis? (Page No.84) MAY/JUNE 2016,APRIL/MAY 2012, MAY/JUNE 2013, MAY/JUNE 2014, APRIL/MAY 2017, NOV/DEC 2017	**C311.2**	BTL5
7	(i) Minimize the regular expression (a+b)*abb. (or) Conversion of regular expression (a/b)*abb to NFA. (Page No.121) MAY/JUNE 2016, NOV/DEC 2016 (ii)Write an algorithm for minimizing the number of states of a DFA. (Page No.141) NOV/DEC 2016	**C311.2**	BTL2
8	(i) Design a lexical analyzer for recognizing the tokens such as identifiers and keywords. (Page No.98) (ii) Describe the error recovery schemes in the lexical phase of a compiler. (Page No.85) MAY/JUNE 2015	**C311.2**	BTL2
9	(i) Differentiate tokens, patterns, lexeme. (Page No.85) MAY/JUNE 2016, APRIL/MAY 2017 (ii) Write notes on regular expressions. (Page No.94)	**C311.2**	BTL2
10	(i) Write notes on regular expression to NFA. Construct Regular expression to NFA for the sentence (a/b)*a and ab*/ab (Page No.121) NOV/DEC 2017 (ii) Construct DFA to recognize the language (a/b)*ab. (Page No.135) MAY/JUNE 2016	**C311.2**	BTL5
11	Convert the Regular Expression abb(a/b)* to DFA using Direct method and minimize it. APRIL/MAY 2017 (Page No.135)	**C311.2**	BTL2

12	Write an algorithm for constructing a DFA from a regular expression. Discuss with an example. (Page No.179)	**C311.2**	BTL 2
13	Solve the given regular expression (a/b)* abb (a/b)* into NFA using Thompson construction and then to minimized DFA. (Page No.152,180)	**C311.2**	BTL 3
14	(i).Solve the following regular expression into minimized DFA. (a/b)*baa (Page No.180) (ii).Comparison between NFA and DFA. (Page No.152)	**C311.2**	BTL 3 & 4
15	i).Describe the Input buffering techniques in detail. (Page No.115) (ii).Elaborate in detail the recognition of tokens. (Page No.128)	**C311.2**	BTL 1

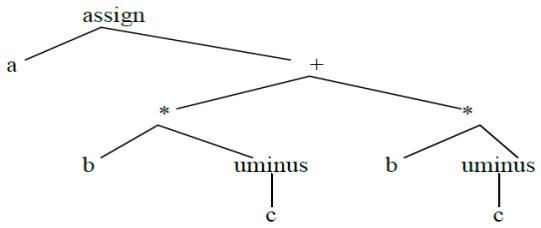
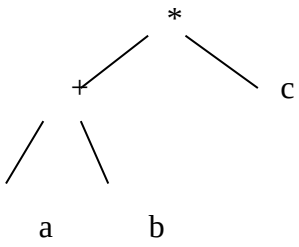
UNIT III SYNTAX ANALYSIS

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

S. No.	Question	Course Outcome	Blooms Taxanomy Level
1	**Differentiate Top Down Parser And Bottom Up Parser? Give Example for each. APRIL/MAY 2010** **Top down Parser are** the parsers which constructs the parse tree from the root to the leaves in pre- order for the given input string. Predictive Parser, Recursive Descendent Parser. **Bottom Up Parser** are the parsers which constructs the parse tree from the leaves to the root for the given input string. LR Parser, SLR Parser.	**C311.3**	BTL2
2	**Compare syntax tree and parse tree. NOV/DEC 2017** • Syntax tree is a variant of a parse tree in which each leaf represents an operand and each interior node represents an operator. • A parse tree may be viewed as a graphical representation for a derivation that filters out the choice regarding replacement order. Each interior node of a parse tree is labeled by some nonterminal A and that the children of the node are labeled from left to right by symbols in the right side of the production by which this A was replaced in the derivation. The leaves of the parse tree are terminal symbols.	**C311.3**	BTL2
3	**Define Handles. MAY/JUNE 2007** A handle of a string is a substring that matches the right side of a production. This reduction helps in constructing the parse tree or right most derivation.	**C311.3**	BTL1
4	**Define ambiguous grammar with an example, and specify it demerits. MAY/JUNE 2016 MAY/JUNE 2012, APRIL/MAY 2012**	**C311.3**	BTL1

	If a grammar produces more than one parse tree for the given input string then it is called ambiguous grammar. **Its demerit is** It is difficult to select or determine which parse tree is suitable for an input string. • **Ex:** E E+E / E*E / **id**		
5	**Mention the properties of parse tree.** **NOV/DEC 2012** • The root is labeled by the start symbol. • Each leaf is labeled by a token or by ε • Each interior node is labeled by a non terminal • If A is the Non terminal, labeling some interior node and x1, x2, x3 … .xn are the labels of the children.	**C311.3**	BTL1
6	**What do you mean by a syntax tree?** **NOV/DEC 2012** Syntax tree is a variant of a parse tree in which each leaf represents an operand and each interior node represents an operator.	**C311.3**	BTL1
7	**Define Handle pruning.** **NOV/DEC2011, APRIL/MAY 2011, NOV/DEC 2016 ,APRIL/MAY 2018** 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 $\alpha=\alpha n$ where αn is the nth right sentential form of unknown right most derivation.	**C311.3**	BTL1
8	**How will you define a context free grammar?** A context free grammar consists of terminals, non-terminals, a start symbol, and productions. i. Terminals are the basic symbols from which strings are formed. "Token" is a synonym for terminal. Ex: **if, then, else.** ii. Nonterminals are syntactic variables that denote sets of strings, which help define the language generated by the grammar. Ex: stmt, expr. iii. Start symbol is one of the nonterminals in a grammar and the set of strings it denotes is the language defined by the grammar. Ex: S. iv. The productions of a grammar specify the manner in	**C311.3**	BTL1

<table>
<tr><td></td><td>which the terminals and non-terminals can be combined to form strings Ex: expr-> id</td><td></td><td></td></tr>
<tr><td>9</td><td>Differentiate sentence and sentential form.
<table><tr><th>Sentence</th><th>Sentential form</th></tr><tr><td>If S=>w then the string w is called Sentence of G.</td><td>If S=>a then a is a sentential form of G.</td></tr><tr><td>Sentence is a string of terminals. Sentence is a sentential form with no nonterminals.</td><td>Sentential form may contain non terminals</td></tr></table></td><td>C311.3</td><td>BTL2</td></tr>
<tr><td>10</td><td>What is left factoring? Give an example. <u>NOV/DEC 2007</u>
Left factoring is a grammar transformation that is useful for producing a grammar suitable for predictive parsing.</td><td>C311.3</td><td>BTL1</td></tr>
<tr><td>11</td><td>Derive the string and construct a syntax tree for the input string ceaedae using the grammar S->SaA|A,A->AbB|B,B->cSd|e <u>MAY/JUNE 2009</u>
S->SaA
S->AaA
S->cSdaA
S->cSaAdaA
S->cAaAdaA
S->cBaAdaA
S->ceaBdaA
S->ceaedaB
C->ceaedae</td><td>C311.3</td><td>BTL2</td></tr>
<tr><td>12</td><td>List the factors to be considered for top-down parsing. <u>MAY/JUNE 2009</u>
We begin with the start symbol and at each step, expand one of the remaining non-terminals by replacing it with the right side of one of its productions. We repeat until only terminals remain. The top-down parse produces a leftmost derivation of the</td><td>C311.3</td><td>BTL1</td></tr>
</table>

	sentence		
13	**Draw syntax tree for the expression a=b*– c+b*– c. NOV/DEC 2017** 	**C311.3**	BTL2
14	**Construct a parse tree of (a+b)*c for the grammer E->E+E/E*E/(E)/id. (or) grammar –(id+id APRIL/MAY 2008, NOV/DEC 2016** 	**C311.3**	BTL2
15	**Eliminate Left Recursion for the grammar A→Ac|Aad|bd APRIL/MAY 2017** **A→bd** A' A'→c A'|ad A'| ϵ	**C311.3**	BTL2
16	**What are the various conflicts that occur during shift reduce parsing? APRIL/MAY 2017** Reduce/Reduce conflict Shift/ Reduce conflict	**C311.3**	BTL1
17	**Eliminate Left Recursion for the given grammar.** **MAY/JUNE 2007** ***E → E + T | T T →T * F | F F →(E)| id*** E →□TE'	**C311.3**	BTL2

	E' → +TE' | ϵ T → □FT' T' →□*FT' | ϵ F →□(E) | id		
18	**Write the algorithm for FIRST and FOLLOW in parser. <u>MAY/JUNE 2016</u>** FIRST(α) is the set of terminals that begin strings derived from α. Rules To compute FIRST(X), where X is a grammar symbol • If X is a terinal, then FIRST(X)={X} • If X-> ϵ is a production, then add ϵ to FIRST(X) • If X is a non terminal and X->Y_1 Y_2..Y_k is a production. Then add FIRST(Y_1) to FIRST (X). If Y_1 derives ϵ. Then add FIRST(Y_2) to FIRST(X) FOLLOW (A) is the set of terminals α that appear immediately to the right of A. For rightmost sentential form of A, $ will be in FOLLOW (A). **Rules** • **If $ is the input end-marker, and S is the start symbol, $ ∈ FOLLOW(S).** • **If there is a production, A → αBβ, then (FIRST (β) – ε) ⊆ FOLLOW (B).** • **If there is a production, A → αB, or a production A → αBβ, where ε ∈ FIRST (β), then FOLLOW (A) ⊆ FOLLOW (B).**	**C311.3**	BTL1
19	**What is dangling reference?**		

	MAY/JUNE 2012,APRIL/MAY 2012 A dangling reference occurs when there is a reference to storage that has been deallocated. It is a logical error to use dangling references, since the value of deallocated storage is undefined according to the semantics of most languages.	**C311.3**	BTL1
20	**Write the rule to eliminate left recursion in a grammar.** **NOV/DEC 2012** A - > Aα|β : A -> βA' ; A' -> αA'|£	**C311.3**	BTL1
21	**Mention the role of semantic analysis.** **NOV/DEC 2012** It is used to check the type information of the syntactically verified statements.	**C311.3**	BTL1
22	**What is the output of syntax analysis phase? What are the three general types of parsers for grammars?** Parser (or) parse tree is the output of syntax analysis phase General types of parsers: 1) Universal parsing 2) Top-down 3) Bottom-up	**C311.3**	BTL1
23	**What are the different strategies that a parser can employ to recover from a syntactic error?** • Panic mode • Phrase level • Error productions • Global correction	**C311.3**	BTL1
24	**What are the goals of error handler in a parser?** The error handler in a parser has simple-to-state goals: • It should report the presence of errors clearly and accurately	**C311.3**	BTL1

	.• It should recover from each error quickly enough to be able to detect subsequent errors. • It should not significantly slow down the processing of correct programs.		
25	**What is phrase level error recovery?** On discovering an error, a parser may perform local correction on the remaining input; that is, it may replace a prefix of the remaining input by some string that allows the parser to continue. This is known as phrase level error recovery.	**C311.3**	BTL1
26	**Define context free language. When will you say that two CFGs are equal?** • A language that can be generated by a grammar is said to be a context free language. • If two grammars generate the same language, the grammars are said to be equivalent.	**C311.3**	BTL1
27	**Give the definition for leftmost and canonical derivations.** • Derivations in which only the leftmost nonterminal in any sentential form is replaced at each step are termed leftmost derivations • Derivations in which the rightmost nonterminal is replaced at each step are termed canonical derivations.	**C311.3**	BTL1
28	**What is a parse tree?** A parse tree may be viewed as a graphical representation for a derivation that filters out the choice regarding replacement order. Each interior node of a parse tree is labeled by some nonterminal A and that the children of the node are labeled from left to right by symbols in the right side of the production by which this A was replaced in the derivation. The leaves of the parse tree are terminal symbols.	**C311.3**	BTL1
29	**Why do we use regular expressions to define the lexical syntax of a language?** i. The lexical rules of a language are frequently quite simple,	**C311.3**	BTL1

	and to describe them we do not need a notation as powerful as grammars. ii. Regular expressions generally provide a more concise and easier to understand notation for tokens than grammars. iii. More efficient lexical analyzers can be constructed automatically from regular expressions than from arbitrary grammars iv. Separating the syntactic structure of a language into lexical and non lexical parts provides a convenient way of modularizing the front end of a compiler into two manageable-sized components.		
30	**When will you call a grammar as the left recursive one?** A grammar is a left recursive if it has a nonterminal A such that there is a derivation A⇒Aα for some stringα.	**C311.3**	BTL1
31	**Define 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 nonterminal "A", we may be able to rewrite the "A" productions to refer the decision until we have seen enough of the input to make the right choice.	**C311.3**	BTL1
32	**Left factor the following grammar:** **S → iEtS | iEtSeS |a E → b.** Ans: The left factored grammar is, S → iEtSS′ | a S′ → eS | ε E → b	**C311.3**	BTL2
33	**Why SLR and LALR are more economical to construct than canonical LR?** For a comparison of parser size, the SLR and LALR tables for a grammar always have the same number of states, and this number is typically several hundred states for a language like Pascal. The canonical LR table would typically	**C311.3**	BTL1

	have several thousand states for the same size language. Thus, it is much easier and more economical to construct SLR and LALR tables than the canonical LR tables.		
34	**Write the configuration of an LR parser?** A configuration of an LR parser is a pair whose first component is the stack contents and whose second component is the unexpended input: (s0 X1 s1 X2 s2 …Xm sm , ai ai+1 … an $)	**C311.3**	BTL1
35	**What is meant by goto function in LR parser? Give an example** • The function goto takes a state and grammar symbol as arguments and produces a state . • The goto function of a parsing table constructed from a grammar G is the transition function of a DFA that recognizes the viable prefixes of G. Ex: goto(I,X) Where I is a set of items and X is a grammar symbol to be the closure of the set of all items [A→αX.β] such that [A→α.Xβ] is in I	**C311.3**	BTL1
36	**LR (k) parsing stands for what?** The "L" is for left-to-right scanning of the input, the "R" for constructing a rightmost derivation in reverse, and the k for the number of input symbols of lookahead that are used in making parsing decisions.	**C311.3**	BTL1
37	**What do you mean by viable prefixes?** • The set of prefixes of right sentential forms that can appear on the stack of a shiftreduce parser are called viable prefixes. • A viable prefix is that it is a prefix of a right sentential form that does not continue the past the right end of the rightmost handle of that sentential form.	**C311.3**	BTL1
38	**What is meant by Predictive parsing? Nov/Dec 2007** A special form of Recursive Descent parsing, in which the look-ahead symbol unambiguously determines the procedure selected for each nonterminal, where no backtracking is required.	**C311.3**	BTL1
39	**Write the rule to eliminate left recursion in a grammar.**	**C311.3**	BTL 6

	Prepare and Eliminate the left recursion for the grammar **S →Aa | b** **A →Ac | Sd |ε** **Ans:** Rules➔A - > Aα|β : A -> βA' ; A' -> αA'|£ ILR➔ S→Aa|b A→SdA'|A' A'→cA'| ε		
40	**Define a context free grammar.** A context free grammar G is a collection of the following 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 → (V U T)*	**C311.3**	BTL1
41	**Define ambiguous grammar.** A grammar G is said to be ambiguous if it generates more than one parse tree for some sentence of language L(G).	**C311.3**	BTL1
42	**List the properties of LR parser.** 1. LR parsers can be constructed to recognize most of the programming languages for which the context free grammar can be written. 2. The class of grammar that can be parsed by LR parser is a superset of class of grammars that can be parsed using predictive parsers. 3. LR parsers work using non backtracking shift reduce technique yet it is efficient one.	**C311.3**	BTL1
43	**Mention the types of LR parser.** SLR parser- simple LR parser LALR parser- lookahead LR parser Canonical LR parser	**C311.3**	BTL1
44	**What are the problems with top down parsing?** The following are the problems associated with top down parsing: Backtracking Left recursion Left factoring Ambiguity	**C311.3**	BTL1
45	**Write short notes on YACC.** YACC is an automatic tool for generating the parser program. YACC stands for Yet Another Compiler Compiler which is basically the utility available from UNIX. Basically YACC is LALR parser generator. It can report conflict or ambiguities in the form of error messages.	**C311.3**	BTL1
46	**Define LR(0) items.**	**C311.3**	BTL1

	An LR(0) item of 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**.**		
47	**What are kernel & non-kernel items?** **Kernel items,** whish include the initial item, S'→ **.**S, and all items whose dots are not at the left end. **Non-kernel items,** which have their dots at the left end.	**C311.3**	BTL1
48	**Solve the following grammar is ambiguous: S→aSbS / bSaS / ε** **LMD 1:** **S=>aSbS** **=>abSaSbS** **=>abaSbS** **=>ababS** **=>abab** **LMD 2:** **S=>aSbS** **=>abS** **=>abaSbS** **=>ababS** **=>abab**	**C311.3**	BTL1
49	**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.	**C311.3**	BTL1
50	**Define yield of the string?** A string that is derived from the root variable is called the yield of the tree.	**C311.3**	BTL1
51	**Summarize the merits and demerits of LALR parser.APRIL/MAY 2018** • This is the extension of LR(O) items, by introducing the one symbol of lookahead on the input. • It supports large class of grammars. • The number of states is LALR parser is lesser than that of LR(1) parser. Hence, LALR is preferable as it can be used with reduced memory. • Most syntactic constructs of programming language can be stated conveniently.	**C311.3**	BTL1
52	Draw the activation tree for the following code. **APRIL/MAY 2018**	**C311.3**	BTL1

	int main()
	{
	printf('Enter Your Name");
	scanf("%s",username);
	int show_data(username);
	printf("Press Any Key to Continue…");
	….
	int show_data(char *user)
	{
	printf("Your Name is %s", username);
	return 0;
	}
	}
	REFER NOTES		

	PART B		
1	(i) Explain Top- Down parsing and Bottom up Parsing. (Page No. 181&195) MAY/JUNE 2007 (ii)Explain Error Recovery in Predictive Parsing. (Page No.192) MAY/JUNE 2007, NOV/DEC 2007, APRIL/MAY 2005	**C311.3**	BTL5
2	Construct an SLR parsing table for the above grammar. (Page No.218) E -> E + T E -> T T -> T * F T -> F F -> (E) F-> id MAY/JUNE 2009, APR/MAY 2011, APRIL/MAY 2008 , MAY/JUNE 2014 NOV/DEC 2012, MAY/JUNE 2015, NOV/DEC 2016 (OR) Construct an SLR parsing table for the given grammar.	**C311.3**	BTL2

	APRIL/MAY 2017 (Refer Notes) G: E -> E+T | T T -> TF | F F -> F* | a | b		
3	Explain LR parsing algorithm with an example.(Page No. 218) NOV/DEC 2017	**C311.3**	BTL5
4	Construct the predictive parser or non recursive predictive parsing table for the following grammar: S -> (L) | a L -> L, S | S Construct the behavior of the parser on the sentence (a, a) and (a,(a,(a,a))) using the grammar specified above. APRIL/MAY 2012 , MAY/JUNE 2007, APRIL/MAY 2005,NOV/DEC 2012, MAY/JUNE 2012 MAY/JUNE 2013 , APRIL/MAY 2017 (Refer Notes)	**C311.3**	BTL2
5	Construct Parsing table for the grammar and find moves made by predictive parser on input id + id * id and find FIRST and FOLLOW. (Page No.186) NOV/DEC 2016, NOV/DEC 2017 E -> E + T E -> T T -> T * F T -> F F -> (E) F-> id	**C311.3**	BTL2
6	Give an algorithm for finding the FIRST and FOLLOW positions for a given non-terminal. (Page No.188) MAY/JUNE 2009 APRIL/MAY 2008	**C311.3**	BTL5
7	Explain Context free grammars with examples (Page No. 165) MAY/JUNE 2016	**C311.3**	BTL5
8	Consider the grammar, E -> E + T E -> T T -> T * F T -> F F -> (E) F-> id Construct a LALR parsing table for the grammar given above. Verify whether the input string id + id * id is accepted by the	**C311.3**	BTL2

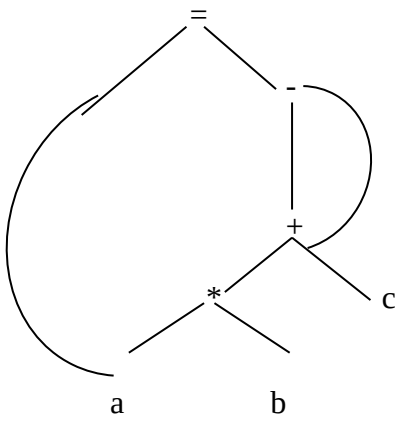
	grammar or not. (Page No. 240) MAY/JUNE 2009 APRIL/MAY 2008		
9	Check whether the following grammar is a LL(1) grammar.MAY/JUNE 2016 APRIL/MAY2005 S-> iEtS | iEtSeS | a E-> b Also define the FIRST and FOLLOW procedures. (Page No. 191)	**C311.3**	BTL2
10	Consider the grammar E -> E + E | E * E | (E) | id . Show the sequence of moves made by the shift-reduce parser on the input id1 + id2 * id3 and determine whether the given string is accepted by the parser or not. (Page No. 198) MAY/JUNE2016	**C311.3**	BTL2
11	What is a shift-reduce parser? Explain in detail the conflicts that may occur during shift-reduce parsing. (Page No.201) MAY/JUNE 2012, APRIL/MAY 2012	**C311.3**	BTL5
12	Consider the grammar given below. E -> E + T E -> T T -> T * F T -> F F -> (E) F-> id Construct an LR parsing table for the above grammar. Give the moves of LR parser on id*id+id (Page No.218 & 220) MAY/JUNE 2007	**C311.3**	BTL2
13	(i)Explain the non-recursive predictive parsing with its algorithm. (Page No.190) MAY/JUNE 2016,APRIL/MAY 2005, NOV/DEC 2007 (ii)Explain the LR parsing algorithm in detail. (Page. No. 218) NOV/DEC 2007, APRIL/MAY 2005	**C311.3**	BTL5
14	(i)What is an ambiguous grammar? Is the following grammar ambiguous? Prove E -> E + E | E * E | (E) | id. MAY/JUNE 2014 (OR) G: E -> E + E | E * E | (E) | -E |id. for the sentence id+id*id. (Refer Notes) NOV/DEC 2016 (ii)List all LR(0) items for the following grammar	**C311.3**	BTL2

	(Refer Notes) MAY/JUNE 2013 S->AS|b A->SA|a		
15	Design a syntax rule (YACC) for arithmetic expression. (Page No.257)	**C311.3**	BTL5
16	. Consider the grammar given below. S -> CC C -> aC C -> d Construct a CLR parsing table for the above grammar. (Page No.230)	**C311.3**	BTL2
17	. Construct parse tree for the input string w = cad using top-down rser. (Page No.181) NOV/DEC 2016 S - > cAd A - > ab | a .	**C311.3**	BTL2

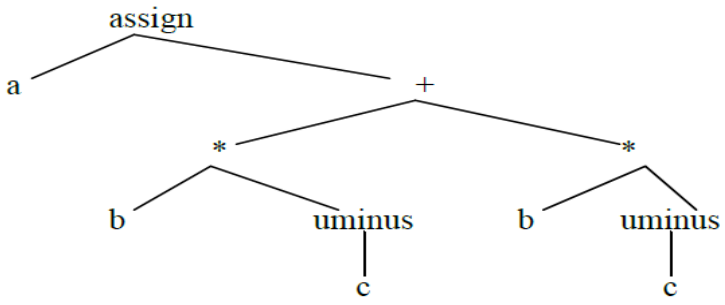
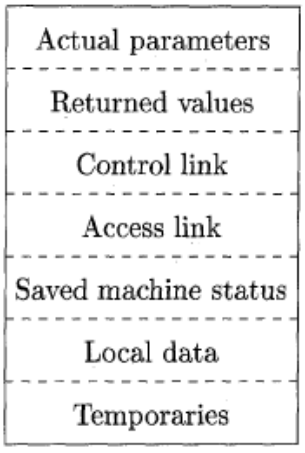
UNIT IV SYNTAX DIRECTED TRANSLATION & RUN TIME ENVIRONMENT

Syntax directed Definitions-Construction of Syntax Tree-Bottom-up Evaluation of S-AttributeDefinitions- 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 FORTAN.

S. No.	Question	Course Outcome	Blooms Taxanomy Level
1	**What are the limitations of static allocation?** **APRIL/MAY 2011** The size of the data object and constraints on its position in memory must be known at compile time. Recursive procedures are restricted, because all activations of a procedure use the same bindings for local names. Data structures cannot be created dynamically, since there is no mechanism for storage allocation at run time.	**C311.4**	BTL1
2	**Draw the DAG for the statement a = (a*b+c)–(a*b+c).** **NOV/DEC 2017**  	**C311.4**	BTL1
3	**Define DAG.** **MAY/JUNE 2016 , NOV/DEC 2007, MAY/JUNE 2007** A DAG for a basic block is a directed acyclic graph with	**C311.4**	BTL1

	the following labels on nodes: i) Leaves are labeled by unique identifiers, either variable names or constants. ii) Interior nodes are labeled by an operator symbol. iii)Nodes are also optionally given a sequence of identifiers for labels.		
4	**When does dangling references occur** **MAY/JUNE 2016** When there is a reference to storage that has been de-allocated, logical error occurs as it uses dangling reference where the value of de-allocated storage is undefined according to the semantics of most languages.	**C311.4**	BTL1
5	**Mention the two rules for type checking.** **NOV/DEC 2011, APRIL/MAY 2017** Type checker for a language is based on information about the syntactic constructs in the language, the notion of types, and the rules for assigning types to language constructs.	C311.4	BTL1
6	**What is syntax directed translation? (or) Write down syntax directed definition of a simple desk calculator.** **NOV/DEC 2016** A syntax directed definition specifies the values of attributes by associating semantic rules with the grammar productions Production E->E1+T Semantic Rule E.code=E1.code||T.code||'+'	**C311.4**	BTL1
7	**What do you mean by binding of names?** **APRIL/MAY 2017** A binding is an association between two entities: Name and memory location (for variables)	**C311.4**	BTL1

	Name and function Typically a binding is between a name and the object it refers to.		
8	**What is synthesized attributes?** A synthesized attribute at node N is defined only in terms of attribute values of children of N and at N	**C311.4**	BTL1
9	**What is inherited attributes ?** An inherited attribute at node N is defined only in terms of attribute values at N's parent, N itself and N's siblings	**C311.4**	BTL1
10	**What is a syntax tree? Draw the syntax tree for the assignment statement** **a := b * -c + b * -c.** **APRIL/MAY 2011, NOV/DEC 2011 NOV/DEC 2012** A syntax tree depicts the natural hierarchical structure of a source program. Syntax tree:  	**C311.4**	BTL1
11	**What are the fields of activation record?**  	**C311.4**	BTL1

12	**What is the order of calling sequence ?** The caller evaluates the actual parameters The caller stores a return address and the old value of *top-sp* into the callee's activation record. The callee saves the register values and other status information. The callee initializes its local data and begins execution.	**C311.4**	BTL1
13	**What are the functions and properties of Memory Manager?** Two basic functions: Allocation Deallocation Properties of memory managers: Space efficiency Program efficiency Low overhead	**C311.4**	BTL1
14	**What is static checking?** A compiler must check that the source program follows both syntactic and semantic conversions of the source language. This checking called static checking detects and reports programming errors.	**C311.4**	BTL1
15	**Give some examples of static checking?** Type checks: A compiler should report an error if an operator is applied to an incompatible operand. Flow of control checks: Statements that cause flow of control to leave a construct must have some place to which to transfer the flow of control.	**C311.4**	BTL1

16	**What is a Procedure?** A procedure definition is a declaration that associates an identifier with a statement. The identifier is the procedure name , and the statement is the procedure body.	**C311.4**	BTL1
17	**What is an Activation tree?** An activation tree is used to depict the way control enters and leaves activations, i)Each node represents an activation of a procedure. ii)The root represents the activation of the main program. iii)The node for a is the parent of the node for b if and only if control flows from activation a to b. iv)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.	**C311.4**	BTL1
18	**What is the use of a control stack?** A control stack is used to keep track of live procedure activations.The idea is to push the node for an activation onto the control stack as the activation begins and to pop the node when the activation ends.	**C311.4**	BTL1
19	**What are the types of storage allocation strategies? (OR) List Dynamic Storage allocation techniques.** **NOV/DEC 2016, NOV/DEC 2017** Static allocation : Lays out storage for all objects at compile time. 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	**C311.4**	BTL1
20	**Define dependency graph**. If an attribute b at a node in a parse tree depends on an attribute c, then the semantic rule for b at the node must be evaluated after the semantic rule that defines	**C311.4**	BTL1

	c. The interdependencies among the inherited and synthesized attributes at the nodes in a parse tree can be depicted by a directed graph called dependency graph.		
21	**What methods have been proposed for evaluating semantic rules?** Parse – tree methods Rule – based methods Oblivious methods	**C311.4**	BTL1
22	**What are the functions used to create the nodes of syntax tree?** mknode(op,left,right) mkleaf(id, entry) mkleaf(num,val)	**C311.4**	BTL1
23	**What is a topological sort?** A topological sort of a directed acyclic graph is any ordering m1,m2,….,mk of the nodes of the graph such that edges go from nodes earlier in the ordering to later nodes.	**C311.4**	BTL1
24	**What is a type expression?** The type of a language construct will be denoted by a "type expression" . Informally a type expression is either a basic type or is formed by applying an operator called a type constructor to other type expressions.	**C311.4**	BTL1
25	**What is a type system?** A type system is a collection of rules for assigning type expressions to the various parts of a program. A type checker implements a type system.	**C311.4**	BTL1
26	**What are coercions?** Conversion from one type to another is said to be implicit if it is to be done automatically by the compiler. Implicit	**C311.4**	BTL1

	type conversions are also called coercions.		
27	**What is an intermediate code?** Intermediate codes are machine independent codes, but they are close to machine instructions.The given program in a source languageis converted to an equivalent program in an intermediate languaue by the intermediate code generator.	**C311.4**	BTL1
28	**What are quadruples?** Quadruples are close to machine instructions, but they are nor actual machine instructions.	**C311.4**	BTL1
29	**What is three address code?** We use the term "three address code" because each statement usually contains three addresses (two for operands, one for the result). General form : X:=Y op Z	**C311.4**	BTL1
30	**What are the representations of three address code?** Quadruples Triples Indirect triples.	**C311.4**	BTL1
31	**What do you mean by strongly typed language?** A language is strongly typed if its compiler can guarantee that the programs that it accepts will execute without type errors.	**C311.4**	BTL1
32	**What is sound type system?** A sound type system eliminates the need for dynamic checking for type errors because it allows us to determine statically that these errors cannot occur when target program runs.	**C311.4**	BTL1
33	**Define environment and state.**		

	The term environment refers to a function that maps a name to a storage location. The term state refers to a function that maps a storage location to the value held there.	**C311.4**	BTL1
34	**Define symbol table.** Symbol table is a data structure used by the compiler to keep track of semantics of the variables. It stores information about scope and binding information about names.	**C311.4**	BTL1
35	**What are the various ways to pass a parameter in a function?** Call by value Call by reference Copy-restore Call by name	**C311.4**	BTL2
36	**What are the functions used to create the nodes of syntax trees?** Mknode (op, left, right) Mkleaf (id,entry) Mkleaf (num, val)	**C311.4**	BTL2
37	**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	**C311.4**	BTL2
38	**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	**C311.4**	BTL2

	effective parameter.		
39	**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.	**C311.4**	BTL1
40	**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.	**C311.4**	BTL2
41	**What are the advantages of generating an intermediate representation?** Ease of conversion from the source program to the intermediate code. Ease with which subsequent processing can be performed from the intermediate code.	**C311.4**	BTL2
42	**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.	**C311.4**	BTL1
43	**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.	**C311.4**	BTL1
44	**What are the various data structure used for implementing the symbol table?** Linear list Binary tree	**C311.4**	BTL2

	Hash table		
45	**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 activation record. The procedure enter (name, type, offset) create a symbol table entry.	**C311.4**	BTL2
46	**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.	**C311.4**	BTL3
47	**List out the two rules for type checking** Type Synthesis Type inference	**C311.4**	BTL2
48	**What is S-Attributed Syntax Directed Translation(SDT)?** If an SDT uses only synthesized attributes, it is called as S-attributed SDT. S-attributed SDTs are evaluated in bottom-up parsing, as the values of the parent nodes depend upon the values of the child nodes.	**C311.4**	BTL2
49	**What is L-Attributed Syntax Directed Translation(SDT)?** If an SDT uses either synthesized attributes or inherited attributes with a restriction that it can inherit values from left siblings only, it is called as L-attributed SDT. Attributes in L-attributed SDTs are evaluated by depth-first and left-to-right parsing manner.	**C311.4**	BTL2

50	**When stack allocation is not possible ?** The values of local names must be retained when an activation ends. A called activation outlives the caller.	**C311.4**	BTL1
	PART B		
1	Discuss the various storage allocation strategies in detail. MAY/JUNE 2016,APRIL/MAY 2011, NOV/DEC 2011, NOV/DEC 2007, NOV/DEC 2014,MAY/JUNE 2013, APRIL/MAY 2017 (Page No.401)	**C311.4**	BTL6
2	Explain in detail about the specification of a simple type checker. MAY/JUNE 2016, MAY/JUNE 2012,APRIL/MAY 2012, NOV/DEC 2014 MAY/JUNE 2013, NOV/DEC 2016, APRIL/MAY 2017 (Page No.348)	**C311.4**	BTL5
3	Explain in detail about the translation of source language details into run time environment. (Page No.473) MAY/JUNE2009	**C311.4**	BTL5
4	Explain about runtime storage management. (Page No.470) NOV/DEC 2017	**C311.4**	BTL5
5	Explain about the parameter passing. (Page No.424) APRIL/MAY 2017	**C311.4**	BTL5
6	Construct a syntax directed definition for constructing a syntax tree for assignment statements MAY/JUNE 2016 S -> id: = E E -> E1 + E2 E -> E1 * E2 E -> - E1 E -> (E1) E -> id (Page No.288)	**C311.4**	BTL2
7	Write about Bottom-Up evaluation S-Attributed definitions. (Page No.293)	**C311.4**	BTL5
8	What is L-attributed definition? Give some example. (Page No.296)		

		C311.4	BTL1
9	Explain synthesized attribute and inherited attribute with suitable examples. (Page No.281)	**C311.4**	BTL5
10	Explain the specification of simple type checker for statements, expressions and functions. (Page No.348) NOV/DEC 2017	**C311.4**	BTL5
11	A syntax Directed Translation scheme that takes strings of a's , b's and c's as input and produces as output the number of substrings in the input string that corresponds to the pattern a(a|b)*c+(a|b)*b. For example the translation of the input string "abbcabcababc" is "3". i) Write a context free grammar that generate all strings of a's, b's and c's. ii) Give the semantic attributes for the grammar symbols. iii) For each production of the grammar present a set of rules for evaluation of the semantic attributes. (Page No.280) NOV/DEC 2016	**C311.4**	BTL2
12	(i).Discuss in detail about the Syntax Directed Definitions. (Page No.304) (ii).Discuss in detail about the specification of simple type checker. (Page No.348)	**C311.4**	BTL 2

13	Generate an intermediate code for the following code segment with the required syntax-directed translation scheme.
	if (a > b)
	x = a + b
	else
	x = a – b (Page No.303)
		C311.4	BTL 6

14	Compare and contrast of static, stack and Heap allocation. (Page No.401)	**C311.4**	BTL 6
15	Analyze the grammar and syntax-directed translation for	**C311.4**	BTL 4

	desk calculator and show the annotated parse tree for exprerssion (3 + 4) * (5 + 6). (Page No.303)		

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.

<table>
<tr><th>S. No.</th><th>Question</th><th>Course Outcome</th><th>Blooms Taxanomy Level</th></tr>
<tr><td>1</td><td>What are basic blocks? NOV/DEC 2011. APRIL/MAY 2005, APRIL/MAY 2010,APRIL/MAY 2008, NOV/DEC 2014 MAY/JUNE 2013, APRIL/MAY 2017

A sequence of consecutive statements which may be entered only at the beginning and when entered are executed in sequence without halt or possibility of branch , are called basic blocks.</td><td>C311.5</td><td>BTL1</td></tr>
<tr><td>2</td><td>What do you mean by copy propagation? APRIL/MAY 2017

After the assignment of one variable to another, a reference to one variable may be replaced with the value of the other variable.

If w := x appears in a block, all subsequent uses of w can be replaced with uses of x.

<table><tr><td>Before

b := z + y

a := b

x := 2 * a</td><td>After

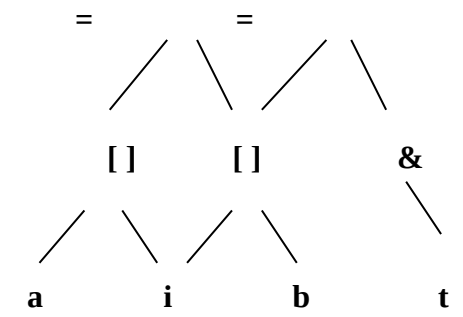
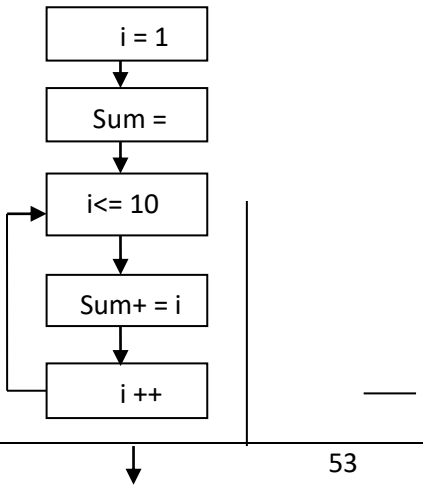
b := z + y

a := b

x := 2 * b</td></tr></table></td><td>C311.5</td><td>BTL1</td></tr>
<tr><td>3</td><td>What is a flow graph? NOV/DEC 2011. MAY/JUNE 2012,NOV/DEC 2014 APRIL/MAY 2008, MAY/JUNE 2013

The basic block and their successor relationships shown</td><td>C311.5</td><td>BTL1</td></tr>
</table>

	by a directed graph is called a flow graph. The nodes of a flow graph are the basic blocks.		
4	**Mention the applications of DAGs. (Or) List the advantages of DAG.** **NOV/DEC 2012** **MAY/JUNE 2013** We can automatically detect common sub expressions. We can determine the statements that compute the values, which could be used outside the block. We can determine which identifiers have their values used in the block.	**C311.5**	BTL1
5	**Write the three address code sequence for the assignment statement.** **MAY/JUNE 2016** d:=(a-b)+(a-c)+(a-c) t1=a-b t2=a-c t3=t1+t2 t4=t3+t2 d=t4	**C311.5**	BTL1
6	**What is meant by peephole optimization?** **MAY/JUNE 2007** Peephole optimization is a technique used in many compliers, in connection with the optimization of either intermediate or object code. It is really an attempt to overcome the difficulties encountered in syntax directed generation of code.	**C311.5**	BTL1
7	**What are the issues in the design of code generators?** **NOV/DEC 2007** Input to the code generator Target programs Memory management Instruction selection Register allocation Choice of evaluation order Approaches to code generation	**C311.5**	BTL1
8	**What is register descriptor and address descriptor?** **NOV/DEC 2012** A register descriptor keeps track of what is currently in each register. An address descriptor keeps track of the location where the current value of the name can be found at run time.	**C311.5**	BTL1
9	**Define DAG.**		

	NOV/DEC 2007, MAY/JUNE 2007 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.	**C311.5**	BTL1
10	**Name the techniques in Loop optimization.** **MAY/JUNE 2014** Code Motion, Induction variable elimination, Reduction in strength	**C311.5**	BTL1
11	**Draw DAG to represent a[i]=b[i]; a[i]=&t;** **NOV/DEC 2014** 	**C311.5**	BTL2
12	**Represent the following in flow graph** **NOV/DEC 2014** **i=1; sum=0;while (i<=10){sum+=i;i++;}** 	**C311.5**	BTL2

	_____		
13	**How to perform register assignment for outer loops? MAY/JUNE 2012** Outer loop L_1 contains an inner loop L_2 names allocated registers in L2 need not be allocated registers in L_1- L_2	**C311.5**	BTL1
14	**Define local optimization.** **APRIL/MAY 2011** The optimization performed within a block of code is called a local optimization.	**C311.5**	BTL1
15	**Define constant folding.** **MAY/JUNE 2013** Deducing at compile time that the value of an expression is a constant and using the constant instead is known as constant folding.	**C311.5**	BTL1
16	**What is code motion? APRIL/MAY 2004, MAY/JUNE 2007, APRIL/MAY-2008** Code motion is an important modification that decreases the amount of code in a loop.	**C311.5**	BTL1
17	**What are the properties of optimizing compilers? MAY/JUNE 2016, MAY/JUNE 2013, NOV/DEC 2007, NOV/DEC 2017** Transformation must preserve the meaning of programs. Transformation must, on the average, speed up the programs by a measurable amount A Transformation must be worth the effort. The source code should be such that it should produce minimum amount of target code. There should not be any unreachable code. Dead code should be completely removed from source language.	**C311.5**	BTL1
18	**Define Local transformation & Global Transformation.** A transformation of a program is called Local, if it can be	**C311.5**	BTL1

	performed by looking only at the statements in a basic block otherwise it is called global.		
19	**What is meant by Common Sub-expressions?** An occurrence of an expression E is called a common sub-expression, if E was previously computed, and the values of variables in E have not changed since the previous computation.	**C311.5**	BTL1
20	**What is meant by Dead Code? Or Define Live variable?APRIL/MAY 2011, NOV/DEC 2012** A variable is live at a point in a program if its value can be used subsequently otherwise, it is dead at that point. The statement that computes values that never get used is known Dead code or useless code. .	**C311.5**	BTL1
21	**What is meant by Reduction in strength?** Reduction in strength is the one which replaces an expensive operation by a cheaper one such as a multiplication by an addition	**C311.5**	BTL1
22	**What is meant by loop invariant computation?** An expression that yields the same result independent of the number of times the loop is executed is known as loop invariant computation.	**C311.5**	BTL1
23	**Define data flow equations.** A typical equation has the form Out[S] = gen[S] U (In[S] – kill[S]) and can be read as, "the information at the end of a statement is either generated within the statement, or enters at the beginning and is not killed as control flows through the statement". Such equations are called data flow equations.	**C311.5**	BTL1
24	**When is a flow graph reducible?** **APRIL/MAY 2012 MAY/JUNE 2012** A flow graph is reducible if and only if we can partition the edges into two disjoint groups often called the forward edges and back edges.	**C311.5**	BTL1
25	**What is induction variable?** A variable is called an induction variable of a loop if every time the variable changes values, it is incremented or decremented by some constant.	**C311.5**	BTL1
26	**What is a cross complier?** **NOV/DEC 2007, MAY/JUNE 2014** **A cross compiler** is a compiler capable of creating	**C311.5**	BTL1

	executable code for a platform other than the one on which the compiler is run. (ie). A compiler may run on one machine and produce target code for another machine.		
27	**What is global data flow analysis?** **NOV/DEC 2014** It is a process in which the values are computed using data flow properties. They are available expressions, reaching definition, live variable and busy variable.	**C311.5**	BTL1
28	**How would you represent the dummy blocks with no statements indicated in global data flow analysis?** **MAY/JUNE 2014** Refer notes	**C311.5**	BTL1
29	**What is the use of algebraic identities in optimization of basic blocks? MAY/JUNE 2012** The algebraic identities are used in Peephole optimization techniques. Simple transformations can be applied on the code in order to optimize it **for ex:** 2*a optimized to a + a.	**C311.5**	BTL1
30	**List the characteristics of peephole optimization.** **NOV/DEC 2016** · Redundant instruction elimination · Flow of control optimization · Algebraic simplification · Use of machine idioms	**C311.5**	BTL1
31	**Define code generations?** 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.	**C311.5**	BTL1

<table>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>32</td><td>Give the variety of forms in target program.
Absolute machine language.
Relocatable machine language.
Assembly language.</td><td>C311.5</td><td>BTL2</td></tr>
<tr><td>33</td><td>Give the factors of instruction selections.
Uniformity and completeness of the instruction sets
Instruction speed and machine idioms
Size of the instruction sets.</td><td>C311.5</td><td>BTL2</td></tr>
<tr><td>34</td><td>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.</td><td>C311.5</td><td>BTL2</td></tr>
<tr><td>35</td><td>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.</td><td>C311.5</td><td>BTL2</td></tr>
<tr><td>36</td><td>Write the code sequence for the d:=(a-b)+(a-c)+(a-c).
<table><tr><td>Statement</td><td>Code generation</td><td>Register descriptor</td><td>Address descriptor</td></tr><tr><td>t:=a-b</td><td>MOV a,R0
SUB b,R0</td><td>R0 contains t</td><td>t in R0</td></tr><tr><td>u:=a-c</td><td>MOV a,R1
SUB c,R1</td><td>R0 contains t
R1</td><td>t in R0
u in R1</td></tr></table></td><td>C311.5</td><td>BTL3</td></tr>
</table>

<table>
<tr><td></td><td></td><td>
<table>
<tr><td></td><td></td><td>contains u</td><td></td></tr>
<tr><td>v:=t+u</td><td>ADD R1,R0</td><td>R0 contains v
R1 contains u</td><td>u in R1
v in R0</td></tr>
<tr><td>d:=v+u</td><td>ADD R1,R0
MOV R0,d</td><td>R0 contains d</td><td>d in R0
d in R0 and memory</td></tr>
</table>
</td><td></td><td></td></tr>
<tr><td>37</td><td>Write the global data flow equation
Data-flow information can be collected by setting up and solving systems of equations of the form : out [S] = gen [S] U (in [S] – kill [S]) This equation can be read as “ the information at the end of a statement is either generated within the statement , or enters at the beginning and is not killed as control flows through the statement.”</td><td>C311.5</td><td>BTL1</td></tr>
<tr><td>38</td><td>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.</td><td>C311.5</td><td>BTL1</td></tr>
<tr><td>39</td><td>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</td><td>C311.5</td><td>BTL2</td></tr>
<tr><td>40</td><td>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</td><td>C311.5</td><td>BTL1</td></tr>
</table>

41	**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.	**C311.5**	BTL1
42	**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.	**C311.5**	BTL1
43	**Prepare the total cost of the following target code.** **MOV a, R0** **ADD b, R0** **MOV C, R0** **ADD R0,R1** **MOV R1,X** MOV a, R0 cost=2 ADD b, R0 cost=2 MOV C, R0 cost=2 ADD R0,R1 cost=1 MOV R1,X cost=2 Total cost=9	**C311.5**	BTL3
44	**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.**	**C311.5**	BTL1
45	**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.	**C311.5**	BTL2

46	**What are the different data flow properties?** □ Available expressions □ Reaching definitions □ Live variables □ Busy variables	**C311.5**	BTL2
47	**What do you mean by machine dependent and machine independent optimization?** The machine dependent optimization is based on the characteristics of the target machine for the instruction set used and addressing modes used for the instructions to produce the efficient target code. The machine independent optimization is based on the characteristics of the programming languages for appropriate programming structure and usage of efficient arithmetic properties in order to reduce the execution time.	**C311.5**	BTL2
48	**What are the basic goals of code movement?** • To reduce the size of the code i.e. to obtain the space complexity. • To reduce the frequency of execution of code i.e. to obtain the time complexity.	**C311.5**	BTL2
49	**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. MOV R0,R1 1 MOV R1,M 2 SUB 5(R0),*10(R1) 3	**C311.5**	BTL1

50	**Identify the constructs for optimization in basic blocks.** **NOV/DEC 2016** ➢ It is a linear piece of code. ➢ Analyzing and optimizing is easier. ➢ Has local scope - and hence effect is limited. ➢ Substantial enough, not to ignore it. ➢ Can be seen as part of a larger (global) optimization problem.	**C311.5**	BTL3
	PART B		
1	i)What are the issues in design of a code generator? Explain in detail. (PageNo:506) MAY/JUNE 2016, NOV/DEC 2007, Nov/Dec 2011, April/May 2012 , MAY/JUNE 2007 APRIL/MAY 2005 APRIL/MAY 2008,MAY/JUNE 2012, NOV/DEC 2016, APRIL/MAY 2017, NOV/DEC 2017 (ii)Define basic block. Write an algorithm to partition a sequence of three-address statements into basic blocks. (Page No:528) MAY/JUNE 2012, APRIL/MAY 2011, APRIL/MAY 2012	**C311.5**	BTL5
2	(i) Explain in the DAG representation of the basic block with example. (Page. No. 598) APRIL/MAY 2012 APRIL/MAY 2005, APRIL/MAY 2008, MAY/JUNE 2012, MAY/JUNE 2015, APRIL/MAY 2017 (ii) How to generate a code for a basic block from its dag representation? Explain. (Page No: 546) APRIL/MAY 2011, NOV/DEC 2011	**C311.5**	BTL5
3	(i) Explain the structure-preserving transformations for basic blocks. (Page No:530) NOV/DEC 2011 (ii) Explain the simple code generation algorithm in detail. (Page No.535) APRIL/MAY 2012, APRIL/MAY 2008 April/May 2011, NOV/DEC 2011, NOV/DEC 2012,MAY/JUNE 2012,	**C311.5**	BTL5

	MAY/JUNE 2013, MAY/JUNE 2016		
4	For the statement given, write three address statements and construct DAG. MAY/JUNE 2013 a+a*(b-c)+(b-c)*d (Refer Notes)	**C311.5**	BTL2
5	Explain the principle sources of code optimization in detail. (Page No:592) MAY/JUNE 2016 NOV/DEC 2011, MAY/JUNE 2012 ,MAY/JUNE 2007 ,MAY/JUNE 2009 ,APRIL/MAY 2008, APRIL/MAY 2005, NOV/DEC 2014 MAY/JUNE 2013 MAY/JUNE 2012, NOV/DEC 2017	**C311.5**	BTL5
6	(i)Write about Data Flow Analysis of structural programs. (Page No:611) NOV/DEC 2011, APRIL/MAY 2012, MAY/JUNE 2014 MAY/JUNE 2013 MAY/JUNE 2012 (ii)Explain in detail optimization of basic blocks with example. (Page No.598) NOV/DEC 2011, MAY/JUNE 2009, MAY/JUNE 2014, NOV/DEC 2014, APRIL/MAY 2017	**C311.5**	BTL5
7	(i)Write an algorithm to construct the natural loop of a back edge. (Page No:604) APRIL/MAY 2011 (ii) Explain in detail about code-improving transformations. (Page No:633) APRIL/MAY 2011	**C311.5**	BTL5
8	(i) Discuss in detail about global data flow analysis. (Page No:608) NOV/DEC 2016 (ii) Explain three techniques for loop optimization with examples. (Page No:633) NOV/DEC 2012, MAY/JUNE 2013, MAY/JUNE	**C311.5**	BTL5

	2015		
9	(i) Write an algorithm for constructing natural loop of a back edge. . (Page No:604) NOV/DEC 2016 (ii) Explain any four issues that crop up when designing a code generator (PageNo:506)	**C311.5**	BTL5
10	(i).Explain in detail about optimization of basic blocks. (Page No.598) (ii).Construct the DAG for the following Basic block & explain it. 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). (Page. No. 598)	**C311.5**	BTL 4
11	Explain loop optimization in detail and apply it to the code given below. i= 0 a:=n-3 if I < a then loop else end label loop b:= i -4 c:= p + b d := m[c] e := d-2 f:= i – 4 g:= p + f m[g]:= e i = i +1 a:= n- 3 if i < a then loop else end label end **RFER NOTES**	**C311.5**	BTL 3
12	Develop a DAG and optimal target code for the expression. x = ((a + b) / (b-c)) – (a + b) * (b-c) +f.	**C311.5**	BTL 6

	RFER NOTES		

13	Create DAG and three – address code for the following C program.
	i = 1;
	s = 0;
	while (i<= 10)
	{
	s = s+ a[i] [i];
	i = i + 1; }
	RFER NOTES
		C311.5	BTL 6

14	(i).Identify the optimization techniques applied on procesure calls? Explain with example. (Page No:633) (ii).Describe the concepts of Efficient Data flow algorithms. (Page No:597)	**C311.5**	BTL 1
15	(i).Describe the common examples of function preserving transformations and loop optimization process? (Page No:586) (ii).List the types of optimization. (Page No:583)	**C311.5**	BTL 1