CST 302	COMPILER DESIGN	Category	L	T	P	Credit	Year of Introduction
		PCC	3	1	0	4	2019

Preamble:

The purpose of this course is to create awareness among students about the phases of a compiler and the techniques for designing a compiler. This course covers the fundamental concepts of different phases of compilation such as lexical analysis, syntax analysis, semantic analysis, intermediate code generation, code optimization and code generation. Students can apply this knowledge in design and development of compilers.

Prerequisite: Sound knowledge in Data Structures, Formal Languages & Automata Theory.

Course Outcomes: After the completion of the course the student will be able to

CO1	Explain the phases in compilation process(lexical analysis, syntax analysis, semantic analysis, intermediate code generation, code optimization and code generation) and model a lexical analyzer **(Cognitive Knowledge Level: Apply)**
CO2	Model language syntax using Context Free Grammar and develop parse tree representation using leftmost and rightmost derivations (**Cognitive Knowledge Level: Apply)**
CO3	Compare different types of parsers(Bottom-up and Top-down) and construct parser for a given grammar **(Cognitive Knowledge Level: Apply)**
CO4	Build Syntax Directed Translation for a context free grammar, compare various storage allocation strategies and classify intermediate representations **(Cognitive Knowledge Level: Apply)**
CO5	Illustrate code optimization and code generation techniques in compilation **(Cognitive Knowledge Level: Apply)**

Mapping of course outcomes with program outcomes

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	✓	✓	✓	✓	✓							✓
CO2	✓	✓	✓	✓	✓							✓
CO3	✓	✓	✓	✓	✓							✓
CO4	✓	✓	✓	✓								✓
CO5	✓	✓	✓	✓								✓

Abstract POs defined by National Board of Accreditation			
PO#	**Broad PO**	**PO#**	**Broad PO**
PO1	Engineering Knowledge	**PO7**	Environment and Sustainability
PO2	Problem Analysis	**PO8**	Ethics
PO3	Design/Development of solutions	**PO9**	Individual and team work
PO4	Conduct investigations of complex problems	**PO10**	Communication
PO5	Modern tool usage	**PO11**	Project Management and Finance
PO6	The Engineer and Society	**PO12**	Life long learning

Assessment Pattern

Bloom's Category	Continuous Assessment Tests		End Semester Examination Marks
	Test 1 (Marks)	Test 2 (Marks)	
Remember	**20**	**20**	**20**
Understand	**40**	**40**	**40**
Apply	**40**	**40**	**40**
Analyze			

Evaluate			
Create			

Mark Distribution

Total Marks	CIE Marks	ESE Marks	ESE Duration
150	**50**	**100**	**3 hours**

Continuous Internal Evaluation Pattern:

Attendance : **10 marks**

Continuous Assessment - Test : **25 marks**

Continuous Assessment - Assignment : **15 marks**

Internal Examination Pattern:

Each of the two internal examinations has to be conducted out of 50 marks. First series test shall be preferably conducted after completing the first half of the syllabus and the second series test shall be preferably conducted after completing the remaining part of the syllabus. There will be two parts: Part A and Part B. Part A contains 5 questions (preferably, 2 questions each from the completed modules and 1 question from the partly completed module), having 3 marks for each question adding up to 15 marks for part A. Students should answer all questions from Part A. Part B contains 7 questions (preferably, 3 questions each from the completed modules and 1 question from the partly completed module), each with 7 marks. Out of the 7 questions, a student should answer any 5.

End Semester Examination Pattern:

There will be two parts; Part A and Part B. Part A contains 10 questions with 2 questions from each module, having 3 marks for each question. Students should answer all questions. Part B contains 2 full questions from each module of which student should answer any one. Each question can have maximum 2 sub-divisions and carries 14 marks.

Syllabus

Module - 1 (Introduction to compilers and lexical analysis)

Analysis of the source program - Analysis and synthesis phases, Phases of a compiler. Compiler writing tools. Bootstrapping. Lexical Analysis - Role of Lexical Analyser, Input Buffering, Specification of Tokens, Recognition of Tokens.

Module - 2 (Introduction to Syntax Analysis)

Role of the Syntax Analyser – Syntax error handling. Review of Context Free Grammars - Derivation and Parse Trees, Eliminating Ambiguity. Basic parsing approaches - Eliminating left recursion, left factoring. Top-Down Parsing - Recursive Descent parsing, Predictive Parsing, LL(1) Grammars.

Module - 3 (Bottom-Up Parsing)

Handle Pruning. Shift Reduce parsing. Operator precedence parsing (Concept only). LR parsing - Constructing SLR, LALR and canonical LR parsing tables.

Module - 4 (Syntax directed translation and Intermediate code generation)

Syntax directed translation - Syntax directed definitions, S-attributed definitions, L-attributed definitions, Bottom-up evaluation of S-attributed definitions. Run-Time Environments - Source Language issues, Storage organization, Storage-allocation strategies. Intermediate Code Generation - Intermediate languages, Graphical representations, Three-Address code, Quadruples, Triples.

Module 5 – (Code Optimization and Generation)

Code Optimization - Principal sources of optimization, Machine dependent and machine independent optimizations, Local and global optimizations. Code generation - Issues in the design of a code generator, Target Language, A simple code generator.

Text Books

1. Aho A.V., Ravi Sethi and D. Ullman. Compilers – Principles Techniques and Tools, Addison Wesley, 2006.

Reference Books

1. D.M.Dhamdhere, System Programming and Operating Systems, Tata McGraw Hill & Company, 1996.
2. Kenneth C. Louden, Compiler Construction – Principles and Practice, Cengage Learning Indian Edition, 2006.

3. Tremblay and Sorenson, The Theory and Practice of Compiler Writing, Tata McGraw Hill & Company,1984.

Sample Course Level Assessment Questions

Course Outcome 1 (CO1):

1) Explain the phases of a compiler with a neat diagram.

2) Define a token. Identify the tokens in the expression a := b + 10.

Course Outcome 2 (CO2):

1) Illustrate the process of eliminating ambiguity, left recursion and left factoring the grammar.

2) Is the following grammar ambiguous? If so eliminate ambiguity.

E→E + E | E*E |(E) | id

Course Outcome 3 (CO3):

1. What are the different parsing conflicts in the SLR parsing table?
2. Design a recursive descent parser for the grammar

   E→E + T | T

   T→T*F | F

   F→(E) | id
3. Construct canonical LR(0) collection of items for the grammar below.

   S → L = R

   S → R

   L → * R

   L → id

   R → L

Also identify a shift reduce conflict in the LR(0) collection constructed above.

Course Outcome 4 (CO4):

1. Write the quadruple and triple representation of the following intermediate code

   R1 = C * D

   R2 = B + R1

   A = R2

   B[0] = A

2. Differentiate S-attributed Syntax Directed Translation(SDT) and L-attributed SDT. Write S - attributed SDT for a simple desktop calculator

Course Outcome 5 (CO5):

1. List out the examples of function preserving transformations.
2. What are the actions performed by a simple code generator for a typical three-address statement of the form x: = y op z.

Model Question Paper

QP CODE:

Reg No: _____

Name: _____ **PAGES : 4**

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

SIXTH SEMESTER B.TECH DEGREE EXAMINATION , MONTH & YEAR

Course Code: CST 302

Course Name: Compiler Design

Max.Marks:100 **Duration: 3 Hours**

PART A

Answer All Questions. Each Question Carries 3 Marks

1. Specify the analysis and synthesis parts of compilation.
2. Define the terms token, lexemes and patterns with examples.
3. Is the grammar S --> S | (S) S / Ɛ ambiguous? Justify your answer.
4. What is left recursive grammar? Give an example. What are the steps in removing left recursion?
5. Compare different bottom-up parsing techniques.
6. What are the possible actions of a shift reduce parser.

7. Differentiate synthesized and inherited attributes with examples.

8. Translate a[i] = b * c – b * d, to quadruple.

9. What is the role of peephole optimization in the compilation process

10. What are the issues in the design of a code generator

(10x3=30)

Part B

Answer any one question from each module. Each question carries 14 Marks)

11. (a) Explain the different phases of a compiler with a running example. **(9)**

(b) List and explain any three compiler construction tools. **(5)**

OR

12. (a) What is a regular definition? Give the regular definition of an unsigned integer **(7)**

(b) Express the role of transition diagrams in recognition of tokens. **(7)**

13. (a) What is Recursive Descent parsing? List the challenges in designing such a parser? **(4)**

(b) Consider the following grammar **(10)**

E-→E or T | T

T→ T and F |F

F→not F | (E) | true | false

(i) Remove left recursion from the grammar.

(ii) Construct a predictive parsing table.

(iii) Justify the statement "The grammar is LL (1)".

OR

14. (a) What is Recursive Descent parsing? List the problems in designing such a parser **(4)**

(b) Design a recursive descent parser for the grammar S→cAd, A→ab/ b **(5)**

Find the FIRST and FOLLOW of the non-terminals S, A and B in the grammar **(5)**

S→aABe
A→Abc | b
B→d

15. (a) Construct the LR(0) set of items and their GOTO function for the grammar **(10)**
S→S S + | S S * | a

(b) Is the grammar SLR? Justify your answer **(4)**

OR

16. (a) Identify LR(1) items for the grammar
S→ CC **(7)**
C→ cC | d

(b) Construct LALR table for the above grammar **(7)**

17. (a) Design a Syntax Directed Translator(SDT) for the arithmetic expression (4 * 7 + 19) * 2 and draw an annotated parse tree for the same. **(8)**

(b) Consider the grammar with following translation rules and E as the start symbol **(6)**

E → E1 # T {E.value=E1.value x T.value ;}
| T{E.value=T.value ;}
T → T1 & F{ T.value=T1.value + F.value ;}
| F{T.value= F.value ; }
F → num { F.value=num. lvalue ;}

Compute E.value for the root of the parse tree for the expression
2#3 & 5# 6 &7

OR

18. (a) Write Syntax Directed Translator (SDT) and parse tree for infix to postfix translation of an expression. **(8)**

    (b) Explain the storage allocation strategies. **(6)**

19. (a) Describe the principal sources of optimization **(7)**

    (b) Illustrate the optimization of basic blocks with examples. **(7)**

OR

20. (a) Write the Code Generation Algorithm and explain the *getreg* function **(6)**

    (b) Generate target code sequence for the following statement **(8)**

d := (a-b)+(a-c)+(a-c).

Teaching Plan

No	Contents	No. of Lecture Hours
Module - 1(Introduction to Compilers and lexical analyzer) (8 hours)		
1.1	Introduction to compilers, Analysis of the source program	1 hour
1.2	Phases of the compiler – Analysis Phases	1 hour
1.3	Phases of the Compiler - Synthesis Phases	1 hour
1.4	Symbol Table Manager and Error Handler	1 hour
1.5	Compiler writing tools, bootstrapping	1 hour
1.6	The role of Lexical Analyzer , Input Buffering	1 hour
1.7	Specification of Tokens	1 hour
1.8	Recognition of Tokens	1 hour

Module – 2 (Introduction to Syntax Analysis) (10 hours)		
2.1	Role of the Syntax Analyser, Syntax error handling	1 hour
2.2	Review of Context Free Grammars	1 hour
2.3	Parse Trees and Derivations	1 hour
2.4	Grammar transformations, Eliminating ambiguity	1 hour
2.5	Eliminating left recursion	1 hour
2.6	Left factoring the grammar	1 hour
2.7	Recursive Descent parsing	1 hour
2.8	First and Follow	1 hour
2.9	Predictive Parsing table constructor	1 hour
2.10	LL(1) Grammars	1 hour
Module - 3 (Bottom up parsing) (9 hours)		
3.1	Bottom-up parsing - Handle Pruning	1 hour
3.2	Shift Reduce parsing	1 hour
3.3	Operator precedence parsing (Concept only)	1 hour
3.4	LR parsing , SLR Grammar, items	1 hour
3.5	Augmented Grammar, Canonical collection of LR(0) items	1 hour
3.6	SLR Parser Table Construction	1 hour
3.7	Constructing Canonical LR Parsing Tables	1 hour
3.8	Constructing LALR Parsing Tables	1 hour
3.9	LALR parser	1 hour
Module - 4 (Syntax Directed Translation and Intermediate code Generation) (9 hours)		
4.1	Syntax directed definitions	1 hour
4.2	S- attributed definitions, L- attributed definitions	1 hour
4.3	Bottom- up evaluation of S- attributed definitions.	1 hour
4.4	Source Language issues	1 hour
4.5	Storage organization	1 hour

4.6	Storage- allocation strategies	1 hour
4.7	Intermediate languages , Graphical representations	1 hour
4.8	Three-Address code	1 hour
4.9	Quadruples, Triples	1 hour
Module - 5 (Code Optimization and Generation) (9 hours)		
5.1	Principal sources of optimization	1 hour
5.2	Machine dependent optimizations	1 hour
5.3	Machine independent optimizations	1 hour
5.4	Local optimizations	1 hour
5.5	Global optimizations	1 hour
5.6	Issues in the design of a code generator – Lecture 1	1 hour
5.7	Issues in the design of a code generator – Lecture 2	1 hour
5.8	Target Language	1 hour
5.9	Design of a simple code generator.	1 hour