PES University, Bangalore

(Established under Karnataka Act No. 16 of 2013)

Department of Computer Science & Engineering
Session : Jan-May, 2023
Course Information (75 Hours)

UE20CS353 – COMPILER DESIGN (4-0-0-0-4)

<table>
<tr><th rowspan="2">Class #</th><th rowspan="2">Chapter Title / Reference Literature</th><th rowspan="2">Topics to be Covered</th><th colspan="2">% of Portion covered</th></tr>
<tr><th>Reference Chapter</th><th>Cumulative</th></tr>
<tr><td>1</td><td rowspan="3">Unit#1

T1: Chapters 1, 3, 4 Introduction, 1.1 - 1.2

Chapter 2: 2.6, 2.7

Lexical Analysis, 3.1–3.5, 3.8</td><td>Compilers: Introduction, Variants of Compiler</td><td rowspan="3">17.85</td><td rowspan="3">17.85</td></tr>
<tr><td>2</td><td>The Language Processing System:
- Discuss how GCC transforms source files to an executable file. (Hands-on)
Introduction to The Phases of a Compiler. The Grouping of Phases into passes.</td></tr>
<tr><td>3-6</td><td>The Phases of a Compiler (An Idea).
- Discuss the structure and importance of Symbol Table
- Discuss Challenges in scanning
- Write Grammar for entire C Language
- Parse the input (code in C language) according to the grammar. (Construct Parse Tree)
- Generate Abstract Syntax tree as output of Semantic Analyzer
- Generate three address code as output for Intermediate Code generation</td></tr>
</table>

		- Discuss simple IR optimization - CSE, constant folding, constant propagation - Generate Assembly code as the output of Code generator phase Students must be able to Parse an input in C Language containing Variable Declaration(+ initialization) Statements, Expressions (involving binary and unary operators like -, ++, -- and logical and relational operators), if, if-else, while, do-while, for-loop. Students must be able to draw a syntax tree for each of the above mentioned statements.		

7		**Lexical Analysis:**
		- The Role of the Lexical Analyzer,
		- Input Buffering :
		    - Discuss in detail the role of lexeme begin and forward pointers.
		    - Use of Sentinels
		- Lexical Errors		

8		Design of a Lexical Analyzer Generator. - Discuss examples that depict the behavior of lexer, how it resolves		

		ambiguities(for example, the maximal munch rule) - Introduce the lex and the yacc tool as part of the lab. - Write a lex program to identify keywords, identifiers. - Write a lex program to verify the output of each question discussed in class.		
9		Specification of Tokens. Recognition of Tokens.		
10		Discuss the implementation of a lexer: - Write pseudocode for a loop-switch implementation		
11-12		Lab 1 : Create a lexer for C language using the lex tool.		
13		Unit 1 Revision		
14		ISA-I		
15	**Unit#2** **T1: Chapter 2 2.4** **T1: Chapter 4**	**Syntax Analysis**: The role of the Parser, Syntax error handling, Error-Recovery Strategies. - Discuss different kinds of errors with examples. Introduction to different parsers: - Top Down Parsers:	**21.43**	**39.28**

	Syntax Analysis, **4.1.1, 4.1.3, 4.1.4, 4.3.3, 4.3.4, 4.4, 4.5 - 4.7.4, 4.9**	- RDP with backtracking - RDP without backtracking - Table driven Predictive Parsers(LL(1)) - Bottom-up Parsers: - Shift-reduce Parser - Table Driven Parser (LR(0), SLR(1), CLR(1), LR(1))		
16		Top–Down Parsing : - RDP with Backtracking Example 1 - RDP with Backtracking Example 2 - Drawbacks of RDP with backtracking.		
17		Top–Down Parsing : - Elimination of Left Recursion, - Left factoring. - RDP without backtracking (Handson) using simple expression grammar.		
18-19		Top–Down Parsing : - Model of Predictive Parser - Computation of First table - Computation of Follow table. - Construction of LL(1) Parser table		
20-21		Top–Down Parsing :		

		- LL(1) Parsing Algorithm - Discuss 2-3 examples on LL(1) parser. - When a grammar is LL(k) where k>1. - Students must be able to analyze the grammar for simple cases and identify whether or not the grammar belongs to LL(1) class of parsers.		
22		Top–Down Parsing : - Error recovery in LL(1) Parser - Discuss how RDP implementation without backtracking is more powerful than the Table driven Predictive Parser(i.e. LL(1) Parser).		
23		Bottom-Up Parsing : - Definition of Handle, - Handle pruning, - General Style of BUP : Shift-Reduce Parsing - Conflicts during Shift-Reduce Parsing		
24		Lab 2 : To validate the syntax of a C program which consists of simple type declaration, if, if-else, and while constructs, Arithmetic/relational expressions.		
25-26		Bottom-Up Parsing : - Model of LR Parser - Table Driven BUP : LR(0) Parser. The LR Parsing algorithm		

27		Table Driven BUP : - SLR(1) Parser - Conflicts in SLR and LR(0) parsers		
28		Table Driven BUP : - Solve 3-4 examples on SLR(1) Parser, - Discuss the meaning of Viable Prefix		
29		Table Driven BUP : - CLR Parser, - LALR Parser - Conflicts in SLR and LR(0) parsers		
30		Table Driven BUP : - More examples on CLR and LALR parsers. - Comparison of all table driven BUP.		
31-32		Assignment 1 : Extend the file from Lab 2 to handle: for loop(eg: for(i = 0, j=10; (i<n || j < i), i++,j--)), do-while loop, initialization within declaration(eg: int a=5, b, c=10;), array declarations(eg: int a[2]; int a[2][3]; int a[2][3][4][5]; int a[2][3], b[4];) logical expressions(||, &&, true, false, !),		

		unary operators : (-, ++, --)		
33		Unit 2 Revision		
34		ISA-II		
35-37		Lab 3 : To implement a Symbol Table. The symbol table must contain necessary information i.e. line number, token-name, type, value, storage required, scope information.		
38	**Unit #3** **T1: Chapter 5 Syntax Directed Translation** **5.1, 5.2.1–5.2.4, 5.3.1, 5.3.2, 5.4.1 – 5.4.3 , 5.5.4** **T1: Chapters 6 Intermediate –Code Generation**	**Syntax Directed Translation:** - Syntax–directed Definitions, - Inherited and Synthesised Attributes, - S–Attributed SDD and its evaluation. Discuss simple calculator examples.	**21.43**	**60.71**
39-40		S–Attributed SDD Examples : - Binary to Decimal conversion - Binary fraction to Decimal - Identifying the type of an expression - int or float - Identifying the sign of the evaluated expression - positive or negative Evaluating an SDD - Dependency Graphs		

41-42	**6.3.3, 6.4.1, 6.6.3, 6.6.4**	Lab 4 : Expression Evaluation : Given for example an input that contains statements such as, a=10, b=20, c= a+b, update the symbol table with the value of c=30. Check for Variable not declared error, mismatch types in expression and display relevant error messages.		
43-44		L–Attributed SDD Examples : - Update type of Variable and storage required in symbol table - Variable declaration verification		
45		More examples on L-attributed SDD: - Array type Variable Declaration		
46		S-attributed SDD to generate Syntax tree for - Expressions - Statements		
47		Lab 5 : Yacc to construct an Abstract Syntax tree for expressions. Define the structure of the node and print the output in postorder traversal.		
48		L-attributed SDD to generate intermediate code for: - Expressions		
49		L-attributed SDD to generate intermediate code for : - if statement		

		- if-else statement - while statement - For Statement - Boolean Expressions		
50		Syntax Directed Translation Schemes : - Postfix SDTs, - Problematic SDTs.		
51		Converting L-attributed SDD to SDT scheme.		
52		Implementing L-attributed SDT scheme during Bottom-Up Parsing.		
53		Unit 3 Revision		
54		ISA-III		
55	**Unit#4** **T1: Chapters 6 Intermediate –Code Generation, 6.1- 6.1.1, 6.2, 6.6: 6.6.1, 6.6.2, 6.6.5, 6.8**	**Intermediate–Code Generation:** - Advantages, - Syntax Tree and DAG construction : discuss examples using unambiguous expression grammar	**21.43**	**82.14**
56		Three-Address Code : Format, examples.		
57		Lab 6 : Yacc to construct an Intermediate Code Generation for expressions. Define a function to generate temporaries and print the output in the quadruple format..		
58		Data Structures for Three-address Code:		

<table>
<tr><td></td><td rowspan="3">T1: Chapter 8 Code Generation, 8.4, 8.5</td><td>- Quadruples
- Triples
- Indirect Triples</td><td rowspan="9"></td><td rowspan="9"></td></tr>
<tr><td rowspan="2">59</td><td>SSA Form</td></tr>
<tr><td>Control Flow Graph generation</td></tr>
<tr><td>60</td><td rowspan="6">T1: Chapter 9 Machine Independent Optimization : 9.1, 9.2.5</td><td>Converting Program to SSA Form.</td></tr>
<tr><td rowspan="2">61</td><td>Machine Independent Optimization: Different Optimizations, Optimization on CFG.</td></tr>
<tr><td>Next-use Algorithm.</td></tr>
<tr><td>62</td><td>Live-variable Analysis.</td></tr>
<tr><td>63</td><td>Unit 4 revision</td></tr>
<tr><td>64</td><td>ISA-IV</td></tr>
<tr><td>65-66</td><td></td><td>Assign 2 : To extend the Lab 5(AST construction) and Lab 6(ICG) file for if and if-else loop.
Use the following productions
S → if (C) { S } S
| if (C) { S } S else { S } S
C → T_ID rel T_ID
rel → < | > | <= | >= | == | !=</td><td></td><td></td></tr>
<tr><td>67</td><td>Unit #5</td><td>Code Generation:</td><td></td><td></td></tr>
</table>

PES University, Bangalore

(Established under Karnataka Act No. 16 of 2013)

Department of Computer Science & Engineering
Session : Jan-May, 2023
Course Information (75 Hours)

UE20CS353 – COMPILER DESIGN (4-0-0-0-4)

	T1: Chapters 7 Run Time Environments, 7.1–7.3 **T1: Chapter 8 Code Generation, 8.1– 8.3, 8.6**	- Issues in the design of a Code Generator.	**17.86**	**100**
68		The Target Language: - Discuss the ISA - Discuss examples - Discuss how one can generate Target code for procedures		
69		Storage allocation Strategies : - Static Allocation (target code generation), - Stack Allocation : Activation tree		
70		- Activation Record, - Calling Sequence, - Return Sequence, - Nested Procedures : Access Links, - Displays. - Discuss the ML Language syntax		
71		Code Generation for Procedures (stack allocation)		
72		A Simple Code generator - The Code generation algorithm. -		
73-74		Lab 7 : Self-Study : Introduction to LLVM, Clang. Simple exercises		
74		Unit 5 Revision		

75		ISA-V		

Reference Textbooks:

Book Type	Code	Title & Author	Publication Information		
			Edition	Publisher	Year
Text Book	T1	**Compilers–Principles, Techniques and Tools** Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffery D. Ullman	2^{nd}	Pearson Education	2009
Reference Book	R1	"Modern Compiler Design", Dick Grune, Kees van Reeuwijk, Henri E. Bal, Ceriel J.H. Jacobs, Koen Langendoen,	2^{nd}	Pearson Education	2012

PES University, Bangalore

(Established under Karnataka Act No. 16 of 2013)

Department of Computer Science & Engineering
Session : Jan-May, 2023
Course Information (75 Hours)

UE20CS353 – COMPILER DESIGN (4-0-0-0-4)

ISA-ESA Policy

	Details	Conducted for	Scaled down to	Total
ISA	ISA-1 (Unit 1)	20	7.5	30 Best of 4 will be selected
	ISA-2 (Unit 2)	20	7.5	
	ISA-3 (Unit 3)	20	7,5	
	ISA-4 (Unit 4)	20	7,5	
	ISA-5 (Unit 5)	20	7.5	
Hands-on	7 Lab exercises	70	10	20
	2 assignments	40	10	
ESA		100	50	50
		Total	100	100

Course Plan Summary

#	Description	Hours
1	Lectures	49
2	Unit Revision	5
3	Unit ISA	5
4	Labs	12
5	Assignments	4
	Total	75

PES University, Bangalore

(Established under Karnataka Act No. 16 of 2013)

Department of Computer Science & Engineering
Session : Jan-May, 2023
Course Information (75 Hours)

UE20CS353 – COMPILER DESIGN (4-0-0-0-4)

Laboratory Experiments
(10 Marks each)

#	Description
1	Create a lexer for C language using the lex tool.
2	To validate the syntax of a C program which consists of simple type declaration, if, if-else, and while constructs, Arithmetic/relational expressions.
3	To implement a Symbol Table. The symbol table must contain necessary information i.e. line number, token-name, type, value, storage required, line-no, scope information. The Grammar must contain rules for variable declaration and expression grammar
4	Expression Evaluation : Given for example an input that contains statements such as, a=10, b=20, c= a+b, update the symbol table with the value of c=30. Check for Variable not declared error, mismatch types in expression and display relevant error messages.
5	Yacc to construct an Abstract Syntax tree for expressions. Define the structure of the node and print the output in postorder traversal.
6	Yacc to construct an Intermediate Code Generation for expressions. Define a function to generate temporaries and print the output in the quadruple format.
7	Introduction to LLVM, Clang. - Simple exercises

PES University, Bangalore

(Established under Karnataka Act No. 16 of 2013)

Department of Computer Science & Engineering
Session : Jan-May, 2023
Course Information (75 Hours)

UE20CS353 – COMPILER DESIGN (4-0-0-0-4)

Assignment Details
(20 Marks each)

<table>
<tr><td>1.</td><td>To extend the file from Lab 2 to handle:
- for loop(eg: for(i = 0, j=10; (i<n || j < i), i++,j--)),
- do-while loop,
- initialization within declaration(eg: int a=5, b, c=10;),
- array declarations(eg: int a[2]; int a[2][3]; int a[2][3][4][5]; int a[2][3], b[4];)
- logical expressions(||, &&, true, false, !),
- unary operators : (-, ++, --)</td></tr>
<tr><td>2.</td><td>To extend the Lab 5(AST construction) and Lab 6(ICG) file for
- if and if-else loop.
- Use the following productions
S → if (C) { S } S
| if (C) { S } S else { S } S
C → T_ID rel T_ID
rel → < | > | <= | >= | == | !=</td></tr>
</table>