Course Title: Compiler Design and Construction
Course No: CSC – 352 **Full Marks: 60+20+20**
Credit Hours: 3 **Pass marks: 24+8+8**

This course is an introductory course to compiler construction. This course covers the important basic elements of compilation and use the material effectively to design and build a working compiler. Topics include language theory, syntax-directed translation, lexical analysis, symbol tables, bottom-up LR(k) parsing, top-down LL(k) parsing, Yacc/Bison and Lex/Flex specifications, intermediate code generation, code generation, error detection, and error recovery.

Course Content:

Unite 1:
1.1 Introduction to compiling: Compilers, analysis of source program, compilers phases, compiler construction tools (Chapter 1) 4hrs
1.2 A simple one pass Compiler: syntax definition, syntax directed translation, parsing, translation for simple expression, symbol table, abstract stack machine. (Chapter 2) 5 hrs

Unite 2:
2.1 Lexical analysis: Role of lexical analyzer, input buffering, specification and recognition of tokens, finite automata, Conversion regular expression to NFA – Thomson's Construction, NFA to DFA –subset construction, regular expression to DFA, State minimization in DFA, Flex/lex introduction. (Chapter – 3) 8hrs
2.2 Syntax Analysis: Role of Parser, Context Free Grammar, writing a grammar, Top-down Parsing – recursive decent parsing, non-recursive predictive parsing, error recovery mechanism, LL grammar, Bottom up parsing – handles, shift reduced parsing, LR Parsers – SLR, LALR, LR, LR/LALR Grammars, Parser Generator. (Chapter 4.1 -4.5, 4.7 & 4.9) 10hrs

Unite 3:
3.1 Syntax directed translation: Syntax directed definitions, syntax tree construction, synthesized and inherited attributes, dependency graph, S-attributed definitions, L-attributed definition, Translations schemes, Top-down and bottom-up evaluation. (Chapter 5.1 – 5.6) 5hrs
3.2 Type Checking: Type system, Specification simple type checker, equivalence of type expression, Type conversion. Type checking Yacc/Bison (Chapter 6.1 -6.4) 3hrs

Unite 4:
4.1: Intermediate languages, Three address code, Declarations, assignment statement, addressing array elements, Boolean expressions, case statements, procedure calls, backpatching. (Chapter 8.1 – 8.7) 4hrs
4.2: Code Generation and Optimization: Code generator design issues, target machine, runtime storage management, basic blocks and flow graphs, next use information, simple code generator, Peephole optimization. (Chapter 9.1 – 9.6 & 9.9) 6hrs

Laboratory works:

1. Simple expression translation program using C/C++
2. Writing C program to scan and identifying token type objects
3. Writing Flex/Lex program to identify source file tokens
4. Implement simple parsing like recursive decent parsing in C/C++
5. Parser writing using parser generator.
6. Writing grammar for intermediate representation in YACC/Bison
7. Writing grammar for type conversion in YACC/Bison
8. Implement Code generation algorithm for simple abstract machine
9. A final project to show the different aspect of compiler design

Text book: Compilers principles, Techniques and Tools, By A.V. Aho, R. Sethi, & J. D. Ullman, 1st edition, Addison Wesley.

Note: The topics not covered in this syllabus and included in the original syllabus are requested to cover in introductory way only.

Model Question

Course Title: Compiler Design and Construction
Course No: CSC – 352 **Full Marks: 60**
Examination Time: 3 Hours **Pass Marks: 24**

(There may be 10 questions each of caring 6 marks or 5 questions with partitions each of caring 12 marks in total)

Attempt all questions

Q.1 Discuss the phases of compiler construction briefly. 6

Q.2 Discuss the role of symbol table in compiler design 6

Q.3 Why regular expression are used in token specification? Write the regular expression to specify the identifier like in C. 6

Q.4 Consider the grammar 6

E -> TE'
E' ->+TE' | ε
T -> FT'
T' -> *FT' | ε
F -> (E) | **id**

Compute the FIRST and FOLLOW for each symbol.

Q.5 Discuss with a suitable example the operation of stack implementation of shift-reduce parsing. 6

Q.6. Define the L-attributed definitions. How L-attributed definitions are evaluated? 6

Q.7 Define the process for Bottom-Up Evaluation of Inherited Attributes. 6

Q.8 Consider the grammar: E -> E + T | T
T -> num . num | num

The grammar generates the expression of + to integer or real. Give a syntax-directed definition to determine the type of expression. When two integers are added, the resulting type is integer otherwise, it is real. 6

Q.9 Write the grammar with semantic rules that translate the C like ***while statement*** into three address code representation. 6

Q.10 How next-use information is useful in code generation? Explain steps of computing next-use information. 6