**20CS3601- COMPILER DESIGN**

**Micro Syllabus**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Offering Branches** | | Computer Science & Engineering | | | | |
| **Course Category:** | | Program Core | **Credits:** | | | 3 |
| **Course Type:** | | Theory | **Lecture-Tutorial- Practical:** | | | 3-0-0 |
| **Prerequisites:** | | Formal Languages and Automata Theory | **Continuous Evaluation:** | | | 30 |
| **Semester End Evaluation:** | | | 70 |
| **Total Marks:** | | | 100 |
| **Course Outcomes** | | | | | | |
| Upon successful completion of the course, the student will be able to: | | | | | | |
| **CO1** | | Understand the fundamental concepts of Compiler Design. | | | L2 | |
| **CO2** | | Apply top-down parsing techniques to generate the parse trees. | | | L3 | |
| **CO3** | | Apply bottom-up parsing techniques to generate parse tree for the given grammar. | | | L3 | |
| **CO4** | | Apply various code optimization techniques for intermediate code forms and Code Generation. | | | L3 | |
| **CO5** | | Analyze the given grammar and apply suitable parsing techniques. | | | L4 | |
| **Course Content** | | | | | | |
| **Unit-1** | **Language Processors:** Overview of language processing system: – preprocessors – compiler – assembler – Linkers & loaders, difference between compiler and interpreter- structure of a compiler:–phases of a compiler.  **Lexical Analysis:** - Role of Lexical Analysis: Lexical analysis Versus Parsing – Tokens, Patterns, and Lexemes – Attributes for Tokens – Lexical errors - Input Buffering: Buffer Pairs – Sentinels  **Specification of Tokens:** Strings and Languages – Operations on Languages – Regular Expressions – Regular Definitions  **Recognition of Tokens:** Transition Diagrams – Recognition of Reserved Words and Identifiers - Completion of the Running Example – Architecture of a Transition–Diagram-Based Lexical Analyzer  **The Lexical Analyzer Generator (LEX):** Use of Lex – Structure of Lex Programs | | | CO1 | | |
| **Unit-2** | **Syntax Analysis:**  **Introduction:** The Role of the parser – Representative Grammars – Syntax Error Handling – Error Recovery Strategies.  **Context Free Grammars:** The formal definition of a CFG – Notational Conventions – Derivations – Parse trees and derivations – Ambiguity  **Writing Grammar:** Lexical Versus Syntax Analysis – Eliminating Ambiguity – Elimination of Left Recursion – Left Factoring  **Top Down Parsing:** Recursive Descent Parsing-FIRST and FOLLOW - LL(1) Grammars – Non recursive Predictive Parsing- Error Recovery in Predictive Parsing. | | | CO1,CO2, CO5 | | |
| **UnIt-3** | **Bottom up Parsing:** Reductions – Handle Pruning - Shift Reduce Parsing – Conflicts During Shift–Reduce Parsing.  **Introduction to simple LR Parsing:**  Why LR Parsers – Items and LR(0) Automaton - The LR-Parsing Algorithm - Constructing SLR–Parsing Tables  . | | | CO1,CO3, CO5 | | |
| **Unit-4** | **More powerful LR parsers:** Canonical LR(1) items - Constructing LR(1) Set of Items – Canonical LR(1) Parsing Tables - Constructing LALR Parsing tables  **Runtime Environments:**  Storage organization : Static versus Dynamic storage allocation  Stack allocation of space: Activation Trees – Activation Records  Heap management: Introduction to Garbage Collection: Design goals for Garbage Collectors – Reachability – Reference Counting Garbage Collectors  **Intermediate code:**  Variants of Syntax Trees: Directed Acyclic Graphs for Expressions  Three address code: Addresses and Instructions- Quadruples - Triples - Indirect Triples. | | | CO1,CO3, CO4, CO5 | | |
| **Unit-5** | **Code Generation: -**  **Basic Blocks and Flow Graphs**-Basic Blocks, Next use Information, Flow Graphs, Representation of Flow Graph, Loops.  **Optimization of Basic Blocks:**  The DAG representation of basic blocks –Finding Local common subexpressions – Dead code elimination – The use of Algebraic Identities -  **Machine independent code optimization -** Principle sources of Optimization - Causes of Redundancy, Running example: Quick Sort, Semantic Preserving transformations, Global common sub expressions, copy propagation, dead code elimination, code motion, induction variables, and reduction in strength.  **Machine dependent code optimization**:  Peephole optimization: Eliminating redundant loads and stores – Eliminating Unreachable code – Flow-of-control optimizations – Algebraic simplification and Reduction in strength - Use of Machine idioms  Register allocation: Global Register Allocation | | | CO1,CO4 | | |
| **Learning Resources** | | | | | | |
| **Text Books** | 1. Compilers: Principles, Techniques and Tools, Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, Second Edition, Pearson Education | | | | | |
| **Reference Books** | 1. Modern Compiler Implementation in C- Andrew N. Appel, Cambridge University. 2. Principles of compiler design, V. Raghavan, Second edition, 2011, TMH.  3. Compiler Design, Muneeswaran K. First Edition, 2012, Oxford University Press. | | | | | |
| **e- Resources & other digital material** | 1.http://www.nptel.iitm.ac.in/downloads/106108052/  2.http://www.vssut.ac.in/lecture\_notes/lecture1422914957.pdf | | | | | |

Course Coordinators HOD