**17CS3104 - COMPILER DESIGN**

|  |  |  |  |
| --- | --- | --- | --- |
| **Course**  **Category:** | Program Core | **Credits:** | 4 |
| **Course**  **Type:** | Theory | **Lecture-Tutorial-Practical:** | 3-2-0 |
| **Prerequisite:** | Basics of Programming Languages and Theory of Computation. | **Sessional Evaluation:**  **Univ. Exam Evaluation:**  **Total Marks:** | 40  60  100 |
| **Objectives** | * To make the student to understand the process involved in compilation. * Creating awareness among students on various types of bottom up parsers. * Understand the syntax analysis, intermediate code generation, type checking, and the role of symbol table etc. | | |

|  |  |  |
| --- | --- | --- |
| **Course Outcomes** | Upon successful completion of the course, the students will be able to: | |
| CO1 | Understand the basics of Compiler Design and the role of Lexical Analyzer |
| CO2 | Study various Syntax analyzers, grammar rules, LR and CLR parsing techniques |
| CO3 | Get exposure on syntax translation and type checking mechanisms to be motivated to develop interpreters or compiles. |
| CO4 | Identify various storage allocation strategies, intermediate code generation and their applicability |
| CO5 | Acquire knowledge on code generation and Run-time storage Management |
| CO6 | Explore the principal sources of optimization and code Improving Transformations in a broader perspective. |
| **Course Content** | **UNIT-I**  **Introduction to Compiling**: Compilers, Analysis of the Source program. Phases of a compiler, Cousins of the Compiler. Grouping of phases, Compiler construction tools.    **Lexical Analysis:** Role of the analyzer. Input buffering, Specification of tokens, Recognition of tokens, A language for Specifying Lexical analyzer.  **UNIT-II**  **Syntax Analysis**: Role of the parser, Context-free grammars, Writing a grammar, Top-down parsing, Bottom-up parsing, Operator-precedence parsing, LR parsers. Using ambiguous grammars, Parser generators.  **UNIT-III**  **Syntax Directed Translation**: Syntax-directed definitions, Construction of syntax trees, Bottom-up evaluation of S-attributed definitions. L-attributed definitions. Top-down translations. Bottom-up evaluation of inherited attributes.  .  **Type Checking**: Type systems, Specification of simple type checker. Equivalence of type expressions, Type conversions, Overloading of functions and operators, Polymorphic functions  **UNIT-IV**  **Run-Time Environments**: Source Language issues, Storage organization, Storage-allocation strategies. Access to non-local names. Symbol tables, Language facilities for dynamic storage allocation. Dynamic storage allocation techniques.  **Intermediate Code generation**: Intermediate languages. Declarations, Assignment statements.  **UNIT-V**  **Code Generation**: Issues in the Design of a code generator, The target machine, Run-time storage management, Basic blocks and flow graphs, Next-use information, A simple code generator, Register allocation and assignment.  **UNIT-VI**  **Code Optimization**: Introduction. The principle source of optimization, Optimization of basic blocks, Loops in flow graphs, Introduction to global data-flow analysis, code improving transformations | |
| **Text Books &**  **References**  **Books** | |  | | --- | | **TEXT BOOKS** | | 1. Alfred V.Aho, Ravi Sethi, and Jeffrey D.Ullman, Compilers-Principles, Techniques and Tools, Pearson Education, 2004... | |  | | **REFERENCE BOOKS** |   1. Alfred V.Aho, Jeffrey D.Ullman, Principles of Compiler Design, Narosa Publications.  2. J.P.Benne, Introduction to compiling Techniques, 2nd Edition, Tata Mc Graw Hill. | |
| **E-Resources** | 1. <https://nptel.ac.in/courses> 2. https://freevideolectures.com/university//iitm | |