Compiler Design

Course Title: Compiler Design **Credit: 3**
Course No: CSIT.313 **Number of period per week: 3+3**
Nature of the Course: Theory + Lab **Total hours: 45+45**
Year: Third, Semester: Fifth
Level: B. Sc. CSIT

1. Course Introduction

This course is designed to develop acquaintance with fundamental concepts of compiler design. The course starts with the basic concepts and also includes different phases of compilers like lexical analysis, syntax analysis, syntax-directed translation, type checking etc. in detail.

2. Objectives

On completion of this course, students will be able to

- develop their knowledge in compiler design
- develop lexical analyzers, parsers, and small compilers using different tools
- develop lexical analyzers, parsers, and small compilers by using general purpose programming languages.

3. Specific Objectives and Contents

Specific Objectives	Contents
• Identify phases of compiler design • Perform analysis of simple program statements • Demonstrate the concepts of symbol-table manager and error handler • Recognize different tools used in compiler design	**Unit One: Introduction [3 Hr.]** 1.1. Compilers, Analysis of the Source Program, Phases of a Compiler 1.2. Cousins of the Compiler, Compiler Construction Tools
• Exemplify lexical analysis and , input buffering and tokens • Understand role of regular expressions and Finite Automata in specification of tokens • Trace the algorithms used in implementing and optimizing pattern matchers	**Unit Two: Lexical Analysis [8 Hr.]** 2.1. The Role of the Lexical Analyzer, Input Buffering, Specification of Tokens, Recognition of Tokens 2.2. Finite Automata, From Regular Expression to an NFA, Optimization of DFA-Based Pattern Matches
• Understand and write context free grammars • Demonstrate different top down and	**Unit Three: Syntax Analysis [12 Hr.]** 3.1. The Role of Parser, Context Free Grammars, Writing a Grammar

bottom-up parsing techniques • Parse the statements using different variants of LR parsers • Handle ambiguity in context free grammars	3.2. Top-Down Parsing, Bottom-Up Parsing 3.3. Operator-Preceding Parsing, LR Parsers, Using Ambiguous Grammars
• Understand generalization of context free grammars • Construct syntax tree from syntax directed definitions • Exemplify bottom up evaluation of s-attributed definitions and l-attributed definitions • Demonstrate top-down translation and bottom-up evaluations of inherited attributes	**Unit Four: Syntax-Directed Translation [6 Hr.]** 4.1. Syntax-Directed Definition, Construction of Syntax Trees 4.2. Bottom-Up Evaluation of S-Attributed Definitions, L-Attributed Definitions 4.3. Top-Down Translation, Bottom-Up Evaluations of Inherited Attributes
• Understand the rules for assigning type expressions • Specify a type checker for a simple language • Exemplify type conversions and attribute grammar for type checking system	**Unit Five: Type Checking [3 hr.]** 5.1. Type Systems, Specification of a Simple Type Checker 5.2. Type conversions, Attribute Grammar for a Simple Type Checking System
• Understand idea behind intermediate languages • Understand declarations, assignment statements, Boolean expressions, and case statements • Demonstrate the concepts of backpatching and procedure call	**Unit Six: Intermediate Code Generation [4 Hr.]** 6.1. Intermediate Languages, Declarations, Assignments Statements 6.2. Boolean Expressions, Case Statements, Backpatching 6.3. Procedure Calls
• Recognize issues in the design of code generator • Understand target machine, its instruction set, and runtime storage management • Demonstrate basic blocks and flow graphs • Exemplify simple code generator, register allocation and assignment • Understand dag representation of basic blocks and code generation from dag	**Unit Seven: Code Generator [5 Hr.]** 7.1. Issues in the Design of a Code Generator, The Target Machine, Run-Time Storage Management 7.2. Basic Blocks and Flow Graphs, Next Use Information, A Simple Code Generator, Register Allocation and Assignment 7.3. The Dag Representation of Basic Blocks, Generating Code from Dags
• Understand some of the most useful code-improving transformations • Demonstrate Peephole optimization optimize basic blocks	**Unit Eight: Introduction to Code Optimization [4 Hr.]** 8.1. Introduction, The Principal Sources of Optimization

• Exemplify loop optimization	8.2. Peephole Optimization, Optimization of Basic Blocks, Loops in Flow Graphs

Evaluation System

Undergraduate Programs							
External Evaluation	**Marks**	**Internal Evaluation**	**Weightage**	**Marks**	**Practical**	**Weightage**	**Mark**
End semester examination	60	Assignments	20%	20	Practical Report copy	25%	20
(Details are given in the separate table at the end)		Quizzes	10%		Viva	25%	
		Attendance	20%		Practical Exam	50%	
		Internal Exams	50%				
Total External	60	Total Internal	100%	20		100%	20
Full Marks 60+20+20 = 100							

External evaluation

1. **End semester examination:**
   It is a written examination at the end of the semester. The questions will be asked covering all the units of the course. The question model, full marks, time and others will be as per the following grid.

2. **External Practical Evaluation:**
   After completing the end semester theoretical examination, practical examination will be held. External examiner will conduct the practical examination according to the above mentioned evaluation. There will be an internal examiner to assist the external examiner. Three hours time will be given for the practical examination. In this examination Students must demonstrate the knowledge of the subject matter.

Full Marks: 100, Pass Marks: 45, Time: 3 Hrs

Nature of question	Total questions to be asked	Total questions to be answered	Total marks	Weightage
Group A: multiple choice	20	20	20×1 = 20	60%
Group B: Short answer type questions	8	6	6×8 = 48	60%
Group C: Long answer type question/long menu driven programs	3	2	2×16 =32	60%
			100	100%

Each student must secure at least 50% marks in internal evaluation in order to appear in the end semester examination. Failed student will not be eligible to appear in the end semester examinations.

Internal evaluation

Assignment: Each student must submit the assignment individually. The stipulated time for submission of the assignment will be seriously taken.

Quizzes: Unannounced and announced quizzes/tests will be taken by the respective subject teachers. Such quizzes/tests will be conducted twice per semester. The students will be evaluated accordingly.

Attendance in class: Students should regularly attend and participate in class discussion. Eighty percent class attendance is mandatory for the students to enable them to appear in the end semester examination. Below 80% attendance in the class will signify NOT QUALIFIED (NQ) to attend the end semester examination.

Presentation: Students will be divided into groups and each group will be provided with a topic for presentation. It will be evaluated individually as well as group-wise. Individual students have to make presentations on the given topics.

Mid-term examination: It is a written examination and the questions will be asked covering all the topics in the session of the course.

Discussion and participation: Students will be evaluated on the basis of their active participation in the classroom discussions.

Instructional Techniques: All topics are discussed with emphasis on real-world application. List of instructional techniques is as follows:

- Lecture and Discussion
- Group work and Individual work
- Assignments
- Presentation by Students
- Quizzes
- Guest Lecture

Students are advised to attend all the classes and complete all the assignments within the specified time period. If a student does not attend the class(es), it is his/her sole responsibility to cover the topic(s) taught during that period. If a student fails to attend a formal exam/quiz/test, there won't be any provision for re-exam.

Laboratory Work

The laboratory work develops practical knowledge on different concepts of compiler design. Students should be able to develop a project using lexical analyzer generator to specify lexical

analyzer, using parser generator to facilitate the construction of the front end of a compiler and using general purpose programming languages like C/C++

Prescribed Text

- Compilers Principles, Techniques, and Tools, Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman; Pearson Education

References

- Compiler Design, Sandeep Saxena, Rajkumar Singh Rathore, S.Chand
- Introduction to Automata Theory, Languages, and Computation, Johne E. Hopcroft, Rajeev Motwani, Jeffrey D. Ulman, Pearson Education