### IT423 – Compiler Design (3-0-2-4)

<table>
<thead>
<tr>
<th>1. Instructor</th>
<th>Manish Khare</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Course Title</td>
<td>Compiler Design</td>
</tr>
<tr>
<td>3. Credit Structure</td>
<td>Lecture hours per week: 3</td>
</tr>
<tr>
<td></td>
<td>Tutorial hours per week: 0</td>
</tr>
<tr>
<td></td>
<td>Practical hours per week: 2</td>
</tr>
<tr>
<td></td>
<td>Total Credits:4</td>
</tr>
<tr>
<td>4. Course Code</td>
<td>IT423</td>
</tr>
<tr>
<td>5. Program /Semester</td>
<td>B Tech (ICT and CS) SEM VI</td>
</tr>
<tr>
<td>6. Category</td>
<td>Technical Elective</td>
</tr>
<tr>
<td>7. Prerequisite Courses</td>
<td>Programming Experience</td>
</tr>
</tbody>
</table>

#### Course Objective:
1. This course is designed to provide a comprehensive knowledge of Compiler Construction.
2. To learn how to construct compiler to translate High Level Languages to Machine Language.
3. To learn different phases of compiler and how to implement them.
4. To learn efficient machine Language Code Generation using the techniques of Optimization.

#### Course Outcomes:
Upon completion of the subject, students will be able to:
1. Understands compiler and various phases in compilation.
2. Understands Lexical Analysis and implement it using LEX tool.
3. Understands LL, LR, and SLR parsing techniques.
4. Implement parsing using YACC tool.
5. Understands Syntax Directed Translation, Symbol Tables and their applications.

#### Course Description
This self-paced course will discuss the major ideas used today in the implementation of programming language compilers, including lexical analysis, parsing, syntax-directed translation, abstract syntax trees, types and type checking, intermediate languages, dataflow analysis, program optimization, code generation, and runtime systems. As a result, you will learn how a program written in a high-level language designed for humans is systematically translated into a program written in low-level assembly more suited to machines.

The course lectures will be presented in short presentations / videos /lectures. To help you master the material, there will be in-lecture questions to answer, quizzes, and three exams: 2 in-sem exam and one final exam. There will also be homework in the form of exercises that ask you to show a sequence of logical steps needed to derive a specific result, such as the sequence of steps a type checker would perform to type check a piece of code, or the sequence of steps a parser would perform to parse an input string.

This course also contains Experimental Lab, which provide an understanding of the language translation peculiarities by designing complete translator for mini language and also provide an understanding of the design aspect of operating system.
### Detailed Course Contents

**Introduction to Compiler:**
Phases and passes, Bootstrapping, Finite state machines and regular expressions and their applications to lexical analysis, implementation of lexical analyzers, LEX-compiler, Formal grammars and their application to syntax analysis, ambiguity, The syntactic specification of programming languages: Context free grammars, derivation and parse trees, capabilities of CFG

**Basic Parsing Techniques:**
Parsers, Shift reduce parsing, operator precedence parsing, top down parsing, predictive parsers Automatic Construction of efficient Parsers: LR parsers, the canonical Collection of LR(0) items, constructing SLR parsing tables, constructing Canonical LR parsing tables, Constructing LALR parsing tables, using ambiguous grammars, an automatic parser generator, YACC tool.

**Syntax-directed Translation:**
Syntax-directed Translation schemes, Intermediate code, postfix notation, Parse trees & syntax trees, three address code, quadruple & triples, Translation of simple statements and control flow statements, Type checking, Type conversions, Equivalence of type expressions, Overloading of functions and operations

**Symbol Tables:**

**Code Generation and Code Optimization:**
Code Generation: Design Issues, the Target Language. Addresses in the Target Code, Basic Blocks and Flow Graphs, Optimization of Basic Blocks, Code Generator. Code optimization: Machine-Independent Optimizations, Loop optimization, DAG representation of basic blocks, value numbers and algebraic laws, Global Data-Flow analysis

### Grading Policy (Tentative)

- Surprise Quizzes - 10%
- Lab/Assignment and Viva - 20%
- In-Sem Exam (I and II) - 30%
- Final Exam - 40%