<table>
<tr><td>Course Title</td><td>Compiler Design</td><td>Course No</td><td colspan="6"></td></tr>
<tr><td rowspan="2">Department/ Specialization</td><td rowspan="2">Computer Science and Engineering</td><td rowspan="2">Credits</td><td>L</td><td>T</td><td>P</td><td colspan="2">C</td></tr>
<tr><td>3</td><td>1</td><td>0</td><td colspan="2">4</td></tr>
<tr><td>Faculty proposing the course</td><td>Faculty, Department of CSE</td><td>Status</td><td>Core</td><td>■</td><td>Elective</td><td>□</td></tr>
<tr><td>Offered for</td><td>B.Tech</td><td>Type</td><td>New</td><td>■</td><td>Revision</td><td>□</td></tr>
<tr><td>To take effect from</td><td>July 2021</td><td rowspan="2">Submitted for approval</td><td colspan="4" rowspan="2">44th Senate</td></tr>
<tr><td>Prerequisite</td><td>Nil</td></tr>
<tr><td>Learning Objectives</td><td colspan="7">The objective of this course is to train students to design various phases of compiler such as Lexical analyzer, syntax analyzer, semantic analyzer, intermediate code generator, code optimizer and code generator. Students are also exposed to design compiler construction tools such as Lexical Analyzer generator and parser generator. Applications of finite state machine and pushdown automation in compiler design are also taught in this course.</td></tr>
<tr><td>Learning Outcomes</td><td colspan="7">● At the end of the course, students will be able to design a programming language and compiler for the same.
● Students will also be able to write large programs.</td></tr>
<tr><td>Course Contents (with approximate breakup of hours for lecture/ tutorial/practice)</td><td colspan="7">Need of compiler-cross compiler-Introduction to phases of compiler –Lexical Analyzer Design using DFAs —regular expression and its application to give syntax of word –Automatic design of Lexical Analyzer from regular expression, Construction of NFA without epsilon moves from regular expression- Efficient Lexical analyzer using Minimization of automata- limitation of recognition capability of Lexical analyzer using Pumping lemma (12L,3T)
Context free grammar & its application to give syntax of program statement – Types of parsing – Top down & bottom up–Recursive descent– Predictive–Shift reduce–Operator precedence–SLR (10L,3T)
Semantic analysis - Intermediate code generation: Declaration – Assignment statements – Boolean expressions– looping and branching statements (7L,2T)
Back patching and procedure calls code generator design issues – Runtime storage management – Code Optimization: Basic blocks – Flow graphs – Next use information – Code generator case study – Directed acyclic graph representation of basic blocks – Peephole optimization technique Introduction to code optimization (10L,3T)
Storage optimization & allocation strategies).Assembly Code Generation: from syntax tree and Directed acyclic graph - from three address code. (5L,1T)</td></tr>
<tr><td>Essential Reading</td><td colspan="7">1. Alfred Aho, Ravi Sethi and Jeffrey D Ullman, Compilers Principles, Techniques and Tools, Pearson Education, 2003. ISBN: 9780321491695</td></tr>
<tr><td>Supplementary Reading</td><td colspan="7">1. Levine J.R, Mason T, Brown D, Lex & Yacc, OReilly Associates, 1992 ISBN: 9781565920002.
2. Allen I. Holub, Compiler Design in C, Prentice Hall, 2003. ISBN: 9780131550452</td></tr>
</table>