<table>
<tr><td>1.</td><td colspan="3">Name of Course</td><td colspan="5">Compiler Design</td></tr>
<tr><td>2.</td><td colspan="3">Course Code</td><td colspan="5">CCPS4573</td></tr>
<tr><td>3.</td><td colspan="3">Name(s) of academic staff</td><td colspan="5"></td></tr>
<tr><td>4.</td><td colspan="3">Rationale for the inclusion of the course/module in the programme</td><td colspan="5"><u>Major</u>
The module deals with Compiler Principles and Techniques applied to general purpose programming language. The goal of this course is to give students a working knowledge of the foundations, tools, and engineering approaches used in building a compiler. The emphasis is on the construction of Compilers to position students to build translators for simple high level languages.</td></tr>
<tr><td>5.</td><td colspan="3">Semester and Year offered</td><td colspan="5">1/4</td></tr>
<tr><td rowspan="2">6.</td><td>Total Student Learning Time (SLT)</td><td colspan="3">Face to Face</td><td colspan="3">Total Guided and Independent Learning</td></tr>
<tr><td>L = Lecture
T = Tutorial
P = Practical
O= Others</td><td>L
28</td><td>T</td><td>P
28</td><td>O</td><td colspan="3">Independent=84
Total =140</td></tr>
<tr><td>7.</td><td colspan="3">Credit Value</td><td colspan="5">3</td></tr>
<tr><td>8.</td><td colspan="3">Prerequisite (if any)</td><td colspan="5"><u>CCPS1543</u> Computer Programming</td></tr>
<tr><td>9.</td><td colspan="8">Objectives:
• To provide students with a solid foundation of the major phases of a compiler.
• To introduce students to the theory behind the various phases, including regular expressions, context free grammars, and finite state automata.
• To introduce students to the design and implementation of a Compiler</td></tr>
<tr><td>10.</td><td colspan="8">Learning outcomes:
After undergoing this module, students will be able to:
• Demonstrate the phases of the compilation process and be able to describe the purpose and implementation approach of each phase.
• Proficiently explain the aspects of theoretical computer science including Languages, Grammars, and Machines.
• Apply prior programming knowledge with a non-trivial programming project to construct a compiler.</td></tr>
<tr><td>11.</td><td colspan="8">Transferable Skills:
1. Working knowledge of the foundations, tools, and engineering approaches used in building a compiler.
• Proficiently communicate about compiler construction.
• Practical experience in building a compiler</td></tr>
</table>

1. **Teaching-learning and assessment strategy**

A variety of teaching and learning strategies are used throughout the course, including:

- Classroom lessons. Lectures and Power Point presentations
- Laboratory sessions: Practice exercises
- brainstorming;
- student-Lecturer discussion
- collaborative and co-operative learning;
- Independent study.

Assessment strategies include the following:

- Ongoing quizzes
- Midterm tests
- Performance Assessment (project, Assigned exercises)
- Lecturer Observation

2. **Synopsis:**

The module deals with Compiler Principles and Techniques applied to general purpose programming language. Subjects include scanning and regular expressions, context-free grammars and parsing, syntax-directed translation, abstract syntax trees, scoping, symbol tables, code-generation, and code optimization., students will design lexical and syntax analyzer phases of complier.

3. **Mode of Delivery:**

- Classroom lessons. Lectures and Power Point presentations
- Laboratory sessions: Practice exercises

4. **Assessment Methods and Types:**

The assessment for this course will be based on the following:

Coursework **50%**

- Quizzes and Assignments 15%
- Group Project 15%
- Mid-Semester Exam 20%

Final Examination **50%**

100%

5. **Mapping of the course/module to the Programme Aims**

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12
4	2	2	0	2	1	0	2	0	4	3	0

6. **Mapping of the course/module to the Programme Learning Outcomes**

LO1	LO2	LO3	LO4	LO5	LO6	LO7	LO8	LO9	LO10	LO11	LO12
4	3	2	3	1	0	1	2	2	0	0	0

7. **Content outline of the course/module and the SLT per topic**

	Details	SLT			
		L	P	Indep.	Total

	Topic 1	**Introduction** • Overview of Compilers, • Mathematical Preliminaries, • Analysis of the source program • The phases of a compiler • Cousins of the compiler • The grouping of phases • Compiler-construction tools • High level Programming Languages, • Implementation of Programming Languages, • Interpreters, Real and Abstract Machines	4	4	12	**20**
	Topic 2	**Simple One-pass Compiler** • Overview • Syntax-directed translation • Parsing • A translator for simple expressions • Lexical analysis • Incorporating a symbol table	2	2	6	**10**
	Topic 3	**Lexical analysis** • The role of the lexical analyzer • Input buffering • Specification of tokens • Recognition of tokens • Finite automata • Design of a lexical analyzer generator • Optimization of DFA-based pattern matchers	4	4	12	**20**
	Topic 4	**Parsing Techniques** • The role of the parser • Context-free grammars • Writing a grammar • Top-down parsing • Bottom-up parsing • Operator-precedence parsing • LR parsers • ambiguous grammars • Parser generators	4	4	12	**20**
	Topic 5	**Syntax-Directed Translation** • Syntax-directed definitions • Construction of syntax trees • Bottom-up evaluation of S-attributed definitions • L-attributed definitions • Top down translation • Bottom-up evaluation of inherited attributes • Recursive evaluators • compiler-construction time	4	4	12	**20**

	Topic 6	**Type Checking** • Type Systems • Specification of a simple type checker • Equivalence of type expressions • Type conversions • Polymorphic functions	2	2	6	**10**
	Topic 7	**Run-Time Environments** • Source language issues • Storage organization and Storage-allocation strategies • Parameter passing • Symbol tables • Dynamic storage allocation techniques	2	2	6	**10**
	Topic 8	**Intermediate Code Generation** • Intermediate languages • Declarations • Assignment statements • Boolean expressions • Case statements • Procedure Calls	2	2	6	**10**
	Topic 9	**Code Generation And Optimization** • Problems in code generation • The target machine • Simple Code generator • Error detection and recovery • Run-time storage management • Sources of Optimization • DAG representation • Global Data flow Analysis	4	4	12	**20**
		Total	28	28	84	**140**

	Laboratory	Laboratory Details
		Exercises based on topics covered in each lecture. practical work must include the following:
		• Lexical Analyzer (Simulation)
		• Writing simple Parser programs
		  • Top-down parsing
		  • Bottom-up parsing
		  • Operator-precedence parsing
		• Developing applications with LEX and YACC
		• Designing simple imperative languages
		• Simple Intermediate Code Generation: Declarations, Boolean expressions , Case statements and Procedure Calls
		• Storage organization and Storage-allocation, Parameter passing ,Symbol tables ,Dynamic storage allocation				

19.	**Main references supporting the course:** • Alexander Meduna, **Elements of Compiler Design**, 1st edition, Auerbach Publications, 2007					
	Additional references supporting the course: 2. Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman **, Compilers: Principles, Techniques, and Tools,** 2nd Edition, Addison Wesley, 2006 3. Parsons, T. W.: **Introduction to Compiler Construction**. Freeman, New York, 1992.					
20.	**Other additional information** All materials will be available to the students online.					