COURSE CODE: CS3207 **COURSE NAME: COMPILER DESIGN**

Course Prerequisites: Automata Theory (Grammar)

Course Objectives:

1. Understand the process of program execution cycle.
2. Understand the translation process from High Level Languages to Machine Level Language.
3. Know the syntax and semantic analysis approaches for efficient code/program verification.
4. Learn the methods of code generation which helps for the optimization.
5. Learn code optimization and runtime code synthesis.
6. Know the process of compiler design for emerging programming languages.

Credits: 4 **Teaching Scheme Theory: 2 Hours/Week**

Tut: 1 Hour/Week

Lab: 2 Hours/Week

Course Relevance: All high level programming languages are easy for users to understand but not understood by a computing machine. The computing machine knows only binary data. A translation is required, in this case, to convert higher level language into machine level, so that the intended program could execute. This translation is done by using a compiler. This course will give you detailed insights of how compilers function internally and design it efficiently. This gives freedom to design your own programming language with its compiler.

SECTION-I

Topics and Contents:

Unit 1: Introduction to Compilers, interpreters, Assembler, Linker and Loader: [CO1 ▯ PO2, PO3, Strength : 2]

Compilers: Introduction to compiler phases, features of machine-dependent and independent compilers, overview of types of compilers, introduction to cross compiler, Interpreters: compiler vs. interpreter, phases, and working, Preprocessor: header file and macro expansion, Assembler: Introduction to Assembler, overview of types of Assembler, Linker and Loader: Introduction to Linker and Loader, overview of types of Linker and Loader. **[4 Hrs.]**

Unit 2: Lexical Analysis and Introduction of Syntax Analysis:
[CO1 PO2, PO3, Strength : 2]

Lexical Analysis and Introduction to Syntax Analysis: Introduction to Compiler, Phases and Passes, Bootstrapping, Role of a Lexical Analyzer, Specification and Recognition of Tokens, LEX/FLEX, Expressing Syntax, Top-Down Parsing, Predictive Parsers. Implementing Scanners, operator precedence parsers. **[4 Hrs.]**

Unit 3: Syntax Analysis and Semantic Analysis:
[CO1, CO2 PO2, PO3, Strength : 2, 3]

Syntax and Semantic Analysis: Bottom-Up Parsing, LR Parsers: Overview of types of LR Parsers, Constructing LALR parsing tables, Introduction to YACC/BISON, Type Checking, and Type Conversion. Symbol Table Structure. **[5 Hrs.]**

SECTION-II

Topics and Contents:

Unit 4: Syntax-Directed Translation and Intermediate Code Generation:
[CO1, CO2 PO2, PO3, Strength : 2, 3]

Syntax-Directed Translation and Intermediate Code Generation: Syntax-Directed Definitions, Bottom-Up Evaluation, Intermediate Representations, and Intermediate Code Generation: Quadraples, Error Detection & Recovery: Lexical Phase errors, syntactic phase errors, semantic errors. More about translation: Array references in arithmetic expressions. **[5 Hrs.]**

Unit 5: Code Generation:
[CO1, CO4 PO2, PO3, PSO3, Strength : 2, 4]

Code Generation: Issues in Code Generation, Basic Blocks and Flow Graphs, Next-use information, A simple Code generator, DAG representation of Basic Blocks, Peephole Optimization. Generating code from DAGs
[4 Hrs.]

Unit 6: Code Optimization, Run-Time Environments and Data Flow Analysis:
[CO1, CO3, CO4, CO5, CO6 PO2, PO4, PSO3, PO11, PO12, Strength : 2, 3, 4, 5, 4]

Code Optimization and Run-Time Environments: Introduction, Principle Sources of Optimization, Optimization of basic Blocks, Introduction to Global Data Flow Analysis, Runtime Environments, and Source Language issues. Storage Organization, Storage Allocation strategies, Access to non-local names, Parameter Passing, Machine Dependent Optimization, Introduction to Data Flow Analysis**:** Introduction to constant propagation, live range analysis. **[4 Hrs.]**

Case studies: LLVM compiler Infrastructure, compiling OOP features, Compiling in multicore environment, Deep learning compilation, Parallel Compilers, Web Compilers. **[2 Hrs.]**

Tutorials:

List of Tutorials (Any Eleven)

Unit 1: Introduction to Compilers, interpreters, Assembler, Linker and Loader:
[CO1 PO2, PO3, Strength : 2]

1. Single and two pass Assembler

2. Two pass Macro processor

3. Types of Linkers

4. Types of Loaders

Unit 2: Lexical Analysis and Introduction of Syntax Analysis:
[CO1 PO2, PO3, Strength : 2]

5. Examples on First and Follow

6. Examples on Lex/Flex regular expressions

7. Construction of LL(1) parser

Unit 3: Syntax Analysis and Semantic Analysis:
[CO1, CO2 PO2, PO3, Strength : 2, 3]

8. Construction of SLR parsing table

9. Construction of Canonical LR parsing table

10. Examples on YACC/Bison grammar rules

Unit 4: Syntax-Directed Translation and Intermediate Code Generation:
[CO1, CO2 PO2, PO3, Strength : 2, 3]

11. Translation Scheme

12. Examples of Intermediate code generation by Quadraples

Unit 5: Code Generation:
[CO1, CO4 PO2, PSO3, Strength : 2, 4]

13. Examples of DAG representation

Unit 6: Code Optimization, Run-Time Environments and Data Flow Analysis:
[CO1, CO3, CO4, CO5, CO6 PO2, PO3, PO4, PSO3, PO11, PO12, Strength : 2, 3, 4, 5, 4]

14. Examples of Code optimization techniques.

Practicals:

List of Practicals (Any Six)

Unit 1 & 2: Introduction to Compilers, interpreters, Assembler, Linker and Loader, Lexical Analysis:
[CO1 PO2, PO3]

1) LEX/FLEX specification and programming regular expressions.

Unit 2: Lexical Analysis and Introduction to Syntax Analysis:
[CO1 PO2, PO3]

2) Implement LEX/FLEX code to count the number of characters, words and lines in an input file.

3) Implement LEX/FLEX code to select only lines that begin or end with the letter ‘a’ and delete everything else.

4) Convert all uppercase characters to lowercase except inside comments.

5) Change all numbers from decimal to hexadecimal notation, printing a summary statistic (number of replacements) to stderr.

6) Implement Lexical Analyzer for language C.

Unit 2 & 3: Lexical Analysis and Introduction to Syntax Analysis, Syntax Analysis and Semantic Analysis:
[CO1, CO2 PO2, PO3]

7) Implement LR/SLR/LALR Parser.

8) YAAC specifications and implement Parser for specified grammar.

9) Implement Parser for language C.

Unit 4: Syntax-Directed Translation and Intermediate Code Generation:
[CO1, CO2 PO2, PO3]

10) Implement Syntax directed Translator.

11) Implement an Intermediate code generator (three address code and Quadruples)

Unit 6: Code Optimization, Run-Time Environments and Data Flow Analysis:

[CO1, CO3, CO4, CO5, CO6 PO2, PO3, PO4, PSO3, PO11, PO12, Strength : 2, 3, 4, 5, 4]

12) Implement a code optimizer for C/C++ subset.

Unit 5: Code Generation:
[CO1, CO4 PO2, PO3, PSO3, Strength : 2, 4]

13) Implement a code generator for C/C++ subset.

Course Projects:

List of Course Project Topics

1. Compiler for subset of C using Lex and YAAC.

2. Compiler for Subset of Java programming Language.

3. Intermediate Code generator.

4. Code Optimizer.

5. Develop an Editor for Assembly programming. (Use available Assembler MASM/TASM to compile the code and execute in editor).

6. Design a system to check syntax and semantics of English Language.

7. Design a system to check syntax and semantics of a subset of Logical programming Language.

8. Design a System to check syntax and semantics of a subset of Python programming language.

9. Compiler for subset of C++ programming language.

10. Compiler for a subset of Algol programming language.

Seminars:

List of Course Seminar Topics

1. Tools complementary to Lex

2. Tools complementary to YAAC

3. Semantic Analyser

4. Obsolete programming Language compiler advantage and issues

5. Android App program compiler

6. Approaches of Intermediate Code generation

7. Recent Trends in Compiler

8. Recent Trends in Interpreter

9. Decompilation

10. Compilation in multicore machines

11. iberg tool

Group Discussion:

List of Group Discussion Topics

1. Compiler Vs Interpreter
2. Multi Language Compiler
3. Tree structure for parsing
4. Decompilers: Good or Bad?
5. Universal Compiler
6. Cross compiler
7. Alternate to parsers
8. Compiler challenges in mobile app development.
9. Online Compilers.
10. Compilers in field of Game development

List of Home Assignments:

List of Design Based Home Assignments

1. Recent methodologies in Intermediate Code Generator
2. Recent methodologies in Code Optimizer
3. Universal Compiler
4. Compiler for Deep learning
5. Recent trend in parsers

List of Case Study Based Home Assignments

1. Algol language Compiler
2. Compilation process (internals) of Functional Programming
3. Compilers for Mobile App development

4. LLVM compiler

5. Cross compiler

List of Blog Based Home Assignment

1. Decompilers: Ethical or Unethical?

2. Multi-paradigm programming compiler

3. State of the Art tools for rapid compiler development

4. Compiler for parallel machines

5. Compiler for distributed computing

List of Survey Based Home Assignments

1. Obsolete Programming Language Compilers

2. Obsolete Programming Language Interpreter

3. Compilers for various programming paradigms

4. Online compilers

5. Mobile app cross compiler

Suggest an assessment Scheme:

Suggest an Assessment scheme that is best suited for the course. Ensure 360 degree assessment and check if it covers all aspects of Bloom's Taxonomy.

Text Books: ***(As per IEEE format)***

1. Aho, A.V., Lam, M.S., Sethi, R., & Ullman, J.D. (2006). Compilers: Principles, Techniques, and Tools, Addison Wesley, ISBN 978-81317-2101-8 (2nd Edition).

2. Cooper, K., & Torczon, L. (2011). Engineering a compiler. Morgan Kaufmann, ISBN 155860-698-X.

3. Appel, A. W. (2004). Modern compiler implementation in C. Cambridge university press.

4. Appel, A. W., & Jens, P. (2002). Modern compiler implementation in Java. In ISBN 0–521–58388–8. Cambridge University Press.

5. Appel, A. W. (1998). Modern Compiler Implementation in ML, In ISBN 0-521-60764-7. Cambridge University Press.

6. Raghavan, V. (2010). Principles of Compiler Design. Tata McGraw-Hill Education

Reference Books: *(As per IEEE format)*

1. Muchnick, S. (1997). Advanced compiler design implementation. Morgan Kaufmann, ISBN 8178672413.

2. Levine, J. R., Mason, J., Levine, J. R., Mason, T., Brown, D., Levine, J. R., & Levine, P. (1992). Lex *& yacc. "O'Reilly Media, Inc".*

MOOCs Links and additional reading material: www.nptelvideos.in

https://swayam.gov.in/nd1_noc20_cs13/preview
https://www.udacity.com/course/compilers-theory-and-practice--ud168
https://online.stanford.edu/courses/soe-ycscs1-compilers

Course Outcomes:

On the completion of course, student will able to-

1. Design basic components of a compiler including scanner, parser, and code generator.

2. Perform semantic analysis in a syntax-directed fashion using attributed definitions.

3. Apply local and global code optimization techniques.

4. Synthesize machine code for the runtime environment.

5. Develop software solutions for the problems related to compiler construction.

6. Adapt themselves to the emerging trends in language processing.

CO-PO Map:

CO	Program Outcomes (PO)												PSO			
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4
1	0	**3**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	**3**	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	**3**	0	0	0	0	0	0	0	0	0	0	0	0

4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**3**	0
5	0	0	0	0	0	0	0	0	0	0	**2**	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	**2**	0	0	0	0
Total	**0**	**3**	**3**	**3**	**0**	**0**	**0**	**0**	**0**	**0**	**2**	**2**	**0**	**0**	**3**	**0**

CO attainment levels:

CO1 – 2, CO2 – 3, CO3 – 3, CO4 – 4, CO5 – 5, CO6 - 4

Future Course Mapping:

Mention other courses that can be taken after completion of this course

Natural Language Processing

Job Mapping:

What are the Job opportunities that one can get after learning this course

Software Engineer, Compiler Developer