FF No. : 654

CS3207::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: 3 Hours/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-1

Topics and Contents

Compilers: Introduction to compiler phases, introduction to cross compiler, features of machine-dependent and independent compilers, overview of types of compilers. Interpreters: compiler vs. interpreter, phases, and working, Preprocessor: header file and macro expansion.
Assembler: Elements of assembly language programming, design of the assembler, assembler design criteria, types of assemblers, two-pass assemblers, one-pass assemblers, assembler algorithms, multi-pass assemblers, variants of assemblers design of two-pass assembler, machine-dependent and machine-independent assembler features.
Linkers: Relocation and linking concepts, static and dynamic linker, subroutine linkages.
Loaders: Introduction to the loader, loader schemes: compile and go, general loader scheme, absolute loaders, relocating loaders, direct linking loaders, MS DOS linker.

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.
Syntax and Semantic Analysis: Bottom-Up Parsing, LR Parsers: constructing SLR parsing tables, constructing Canonical LR parsing tables, Constructing LALR parsing tables, using ambiguous grammars, YACC/BISON Type Checking, Type Conversion. Symbol Table Structure.

SECTION-II

Topics and Contents

Syntax-Directed Translation and Intermediate Code Generation: Syntax-Directed Definitions, Bottom-Up Evaluation, Top-Down Translation, Intermediate Representations, Intermediate Code Generation. Error Detection & Recovery: Lexical Phase errors, syntactic phase errors, semantic errors. More about translation: Array references in arithmetic expressions, case statements, introduction to SSA form
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.
Code Optimization and Run-Time Environments: Introduction, Principle Sources of Optimization, Optimization of basic Blocks, Introduction to Global Data Flow Analysis, Runtime Environments, Source Language issues. Storage Organization, Storage Allocation strategies, Access to non-local names, Parameter Passing
Machine Dependant Optimization: Instruction (Basic-Block) scheduling algorithm, Instruction selection algorithm, Register allocation techniques, peephole optimizations
Introduction to Data flow analysis: Introduction to constant propagation, live range analysis
Case studies: LLVM compiler Infrastructure, Power of SSA, compiling OOP features, Compiling in multicore environment, Deep learning compilation,

List of Practicals: (Any Six)

1) LEX/FLEX specification and programming regular expressions

2) Add line numbers to lines of text, printing the new text to the standard output using LEX/FLEX.

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

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

5) Implement LR/SLR/LALR Parser.

6) Implement Syntax directed Translator.

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

6) Change all numbers from decimal to hexadecimal notation, printing a summary

statistic (number of replacements) to stderr.

7) Implement Lexical Analyzer for language C-.

8) YAAC specifications and implement Parser for specified grammar.

9) Implement Parser for language C-.

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

List of Projects:

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

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

List of Course 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:

Design:

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

Case Study:

1. Algol Compiler

2. Compilation process(internals) of Functional Programming

3. Compilers for Mobile App development

4. LLVM compiler

5. Cross compiler

Blog

1. Decompilers: Ethical or Unethical?

2. Multiparadigm programming compiler

3. State of the Art tools for rapid compiler development

4. Compiler for parallel machines

5. Compiler for distributed computing

Surveys

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.

MSE(15)+ESE(15)+HA(10)+LAB(10)+CP(10)+CVV(20)+SEMINAR(10)+GD(10)

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:

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

CO1-PO2 - 2

CO2-PO3 - 3

CO3-PO4 - 3

CO5-PO11 - 2

CO6-PO12 - 1

CO4-PSO3 - 3

CO attainment levels

CO1 - 2

CO2 - 3

CO3 - 3

CO5 - 3

CO4 - 4

CO6 - 5

Future Courses Mapping:

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

Job Mapping:

Software Engineer, Compiler Developer