Instructor's Manual

for the course on

Compiler Construction

Authors : **S. Biswas, D. M. Dhamdhere & A. Sanyal**

Target Audience : **Undergraduates in Computer Science & Engineering**

Prerequisites : **Courses on:**

1. **Data Structures**
2. **Programming Languages**
3. **Computer Organisation**
4. **Theory of Computation (desirable).**

Duration : **60 hours**

Date of Issue : **14th September 1994**

Prepared by : **IIT Bombay Resource Centre**

Course code : **IMPACT/LMD-3/CC**

Course Contents

1 Rationale

Language translation techniques play a vital role in many fields of Computer Technology. The need for language translators arises because the language in which one wants to express ones computational needs is significantly different from the language recognised by a real machine. Development of translators, therefore, have resulted in two important fallouts. On one hand it has a strong influence on the design of new programming languages, on the other the issue of efficient translation of programming languages is inherently linked with the design of new architectures for higher throughputs. Language translation also plays an important role in the design of command languages and user interfaces of software systems, which require knowledge of scanning and parsing techniques.

In the current context, it is essential for computer science undergraduate students to develop a thorough knowledge of the theory and techniques of language translation. This course is aimed at imparting the same through a detailed treatment of the theory of language translation, and a familiarity with the use of currently available software tools for automatic generation of various modules of a language translator.

2 Objectives

At the end of this course on Compiler Construction, the student should be able to

- Understand the two models of language processing, viz. compilers and interpreters, and realize their strengths and limitations.
- Understand the influence of various features of programming languages and machine architectures on the efficiency of language translation.
- Appreciate the role of lexical analysis in compilation, understand the theory behind design of lexical analysers and lexical analyser generators, and be able to use Lex to generate lexical analysers.

- Be proficient in writing grammars to specify syntax, understand the theories behind different parsing strategies – their strengths and limitations, understand how the generation of parsers can be automated, and be able to use Yacc to generate parsers.

- Understand the error detection capabilities of a compiler and how to report them in a user friendly manner.

- Understand the need for various static semantic analyses such as declaration processing, type checking, and intermediate code generation, and be able to perform such analysis in a syntax directed fashion through the use of attribute grammars.

- Understand the issues involved in allocation of memory to data objects, and the relation of such issues with higher level language features such as data types, scope and nesting rules, and recursion.

- Understand the key issues in the generation of efficient code for a given architecture.

- Understand the role played by code optimisation.

3 Course Module Nos., Title and Duration

Module	**Module Name**	**No. of lectures**
CC-1.	Introduction to Language Processing	5
CC-2.	Lexical Analysis	5
CC-3.	Syntax Analysis	11
CC-4.	Error Analysis	4
CC-5.	Static Semantics and Intermediate Code Generation	10
CC-6.	Runtime Environment	5
CC-7.	Code Generation	10
CC-8.	Code Optimisation	10

Course Modules

Module 1:Introduction to Language Processing

No. of lectures : 5

Topics Covered

1. Influence of programming language features on the translation tasks
2. The translation model – source language and target machine dependent issues
3. The pass structure of compilers
4. Overview of compilation
5. Interpreters

Objectives

At the conclusion of the module, the student should be able to :

1. Understand the influence of source language features on the efficiency and complexity of the language translation task.
2. Segregate the source language and target machine dependent issues.
3. Understand the need for multiple passes in a compiler.
4. Understand which language features require interpretation, and why.
5. Understand the fundamental tasks in the process of language translation, their need, and interfaces.

Module 2: Lexical Analysis

No. of lectures : 5

Topics Covered

1. Motivation
2. Basic Concepts and Issues
3. Handcoding a Lexical Analyser
4. Automatic generation of Lexical Analysers
5. Minimization of DFA
6. Efficient representation of DFA.

Objectives At the conclusion of the module, the student should be able to :

1. Understand the role of lexical analysis and the interface of a lexical analyser with the rest of the compiler.
2. Identify tokens in a language and the action to performed on discovery of a token.
3. Understand the issues involved in handcoding a lexical analyser.
4. Express tokens in a language through regular expressions and be able to specify action routines along with token specifications.
5. See a DFA as a recogniser of tokens. Understand the algorithms involved in converting regular expressions into a minimized DFA.
6. See clearly that the DFA, a driver routine and the action routines, combined appropriately, indeed constitute a lexical analyser.
7. Understand the need for an efficient representation of a DFA and see how it can be achieved by a particular representation using four arrays.

Module 3: Syntax Analysis

No. of lectures : 10

Topics Covered

1. Motivation and Issues
2. Basic Concepts and Definitions in Parsing
3. Principles of Top-Down parsing
4. Top-Down Parser Construction
5. Principles of Bottom-Up Parsing
6. Construction of LR Parsers
7. Automatic Generation of LR Parsers

Objectives

At the conclusion of the module, the student should be able to:

1. Understand the role of a parser, its interface with the scanner and the rest of the compiler.
2. Understand the basic concepts such as derivations, parse-trees, ambiguity and their relevance to parsing.
3. Construct equivalent Context Free Grammars using transformations such as left factoring, left recursion removal, etc. and know their relationship to the underlying parsing method.
4. Understand the algorithms that generate various useful information such as FIRST, FOLLOW, items and sets of items, etc., from a CFG.
5. Create LL(1), SLR(1), LALR(1) and LR(1) parsing tables manually for small grammars.

6. Relate the theoretical results of LR parsing to understand the behaviour of these parsers and also their capabilities.
7. Write grammar rules and actions in yacc to construct a parser for a subset of some programming language and test the same.

Module 4: Error Analysis

No. of lectures : 4

Topics Covered

1. Introduction to Error analysis; detection, reporting and recovery from syntax errors.

2. Error detection and reporting in LR parsers.

3. Error recovery in YACC; error recovery algorithm and its illustration.

4. Guidelines for using *error productions* and *yyerrok* in the grammar rules.

Objectives At the conclusion of the module, the student should be able to:

1. Understand the issues in error analysis, particularly detection, reporting and recovery from syntax errors.

2. Understand the issues in incorporating error messages in LR parsers.

3. Understand the issues in performing error recovery from syntax errors.

4. Understand the features supported in yacc for inclusion of error recovery actions.

5. Understand the basic algorithm employed in yacc for syntax directed error recovery. And use this knowledge to construct parsers that may perform acceptable recovery in the presence of syntax errors.

Module 5: Static Semantics and Intermediate Code Generation

No. of lectures : 10

Topics Covered

1. Basic Concepts and Issues
2. Syntax Directed Analysis
3. S-attributed Definitions
4. Semantic analysis using S-attributes. This involves writing attribute grammars for declaration processing, type checking and generating intermediate code for different language constructs.
5. L-attributed Definitions

Objectives At the conclusion of the module, the student should be able to:

1. Understand the issues in static semantic analysis.
2. Understand how to perform such analyses in a syntax directed fashion by augmenting the CFG with semantic rules and the parsing process by attribute evaluation.
3. Understand what S-attributed definitions are and see how the evaluation of attributes in a S-attributed definition can be integrated with parsing.
4. Understand the issues involved in performing semantic analysis such as processing of declarations in block structured languages, type checking and type coercion and intermediate code generation for assignment statement and other language constructs. Should also be able to write attribute grammars to implement the required translation.
5. Understand L-attributed definitions, their advantage over S-attributed definitions and the evaluation of attributes in a L-attributed definition.

Module 6: Run-time Environments

No. of lectures : 5

Topics Covered

1. Basic Concepts and Issues
2. Activation Records
3. Parameter Passing
4. Compiling Procedure Calls
5. Handling Dynamic Data Structures
6. Conclusion

Objectives At the conclusion of the module, the student should be able to :

1. Understand the need for runtime storage management and see the issues involved in terms of source language features.
2. Understand the concept of an environment.
3. Understand the reason for broadly classifying the available memory into code, data, stack and heap.
4. Understand the notion of an activation record and the factors which influence its contents.
5. Understand why static allocation suffices for some languages, whereas others need stack allocation.
6. Understand the issues involved in handling different parameter passing mechanisms.
7. Understand memory allocation techniques for dynamic data structures.

Module 7: Code Generation

No. of lectures : 10

Topics Covered

1. Simple Code generation for expressions — operand descriptors and the handling of partial results. Code generation algorithm for a simple machine architecture like Intel 8086.

2. Issues in efficient code generation — instruction costs, register utilization and evaluation order. The Sethi-Ullman algorithm.

3. Optimal code generation for expression trees — The machine model, instruction costs. Cover of an expression tree. The Aho-Johnson dynamic programming algorithm for optimal code generation.

Objectives

At the conclusion of the module, the student should be able to :

1. Understand the role of operand descriptors in the process of code generation,

2. Understand the techniques for handling partial results in unoptimized code,

3. Understand the importance of and the issues involved in generating optimal code for expressions.

4. Understand the assumptions behind Sethi-Ullman algorithm – the expressions considered, the machine model and the the cost measure of the code generated.

5. Understand Sethi-Ullman algorithm, especially the influence of register requirements on the optimal evaluation order for an expression.

6. Understand the proof of optimality and complexity of the algorithm.

7. Understand the assumptions behind Aho-Johnson algorithm.

8. Understand Aho-Johnson algorithm along with its optimality proof and complexity.

Module 8: Code Optimisation

No. of lectures : 10

Topics Covered

1. Fundamental optimising transformations.
2. Local and global optimisation.
3. Control flow analysis - useful concepts from graph theory.
4. Data flow analysis - specification and solution of data flow equations.
5. Implementation of global optimising transformations.
6. Register allocation.

Objectives At the conclusion of the module, the student should be able to :

1. Understand the fundamental optimising transformations and the cost-effectiveness of local and global optimisation.
2. Understand the key notions in control flow analysis.
3. Understand the importance of data flow analysis and be able to devise data flow equations for simple problems in optimisation.
4. Understand the fundamentals of the solution techniques in data flow analysis.
5. Understand the fundamental approaches in register allocation.