COMPILER DESIGN

Subject Code: **CS601PC**

Regulations : R16 - JNTUH

Class: III Year B.Tech CSE II Semester

Department of Computer Science and Engineering

BHARAT INSTITUTE OF ENGINEERING AND TECHNOLOGY Ibrahimpatnam - 501 510, Hyderabad

COMPILER DESIGN (CS601PC)
COURSE PLANNER

OBJECTIVE AND RELEVANCE

The objective of this course is to provide a student with an understanding of the fundamental principles in compiler design and to provide the skills needed for building compilers for various situations that one may encounter in a career in Computer Science. After the course a student should have an understanding, based on knowledge of the underlying machine architecture, the limitations and efficiency of various design techniques of compilers implementation. The students will also be exposed to System Software programming using Java and will be trained to develop programs in different parsers, optimization etc., It also covers programming in various tools like LEX and YACC for scanning and parsing etc.

COURSE PURPOSE

The purpose of this course is to provide students with an overview of the issues that arise in Compiler construction as well as to throw light upon the significant theoretical developments and tools that are deep rooted into computer science. This course will basically introduce the major phases of Compiler construction and also its theoretical aspects including regular expressions, context free grammars etc. As a part of this course the students are required to design a compiler passing through the phases namely Lexical Analysis, Syntax Analysis, Semantic Analysis and Intermediate code generation for a small language.

SCOPE OF COURSE

This course provides the concepts related Compiler Design. Here, A mastery of these areas is essential for us to develop Compiler Design that utilizes computer resources in an effective manner. A Compiler Design will certainly help the student to create good System. The Regular Languages are required to determine the quality of Representation Mechanisms. As computers become faster and faster, the need for programs that can handle large amounts of input becomes more acute. The objective of this course is to teach students good automation skills simultaneously so that they can develop such languages with the maximum amount.

PRE-REQUISITES

1. Formal Languages and Automata Theory
2. Computer organization.
3. Discrete mathematics
4. System software

Course Objectives:

- To understand the various phases in the design of a compiler.
- To understand the design of top-down and bottom-up parsers.
- To understand syntax directed translation schemes.

- To introduce LEX and YACC tools.
- To learn to develop algorithms to generate code for a target machine.

Course Outcomes:

- CO1:Ability to design, develop, and implement a compiler for any language.
- CO2:Able to use LEX and YACC tools for developing a scanner and a parser.
- CO3:Able to design and implement LL and LR parsers.
- CO4:Able to design algorithms to perform code optimization in order to improve the performance of a program in terms of space and time complexity.
- CO5:Ability to design algorithms to generate machine code

How Program Outcomes are Assessed:

Program Outcomes (PO)		Level	Proficiency assessed by
PO1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.	3	Assignments, Tutorials, Mock Tests
PO2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.	3	Assignments, Tutorials
PO3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.	3	Assignments, Tutorials, Mock Tests
PO4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.	3	Assignments, Tutorials, Mock Tests
PO5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.	3	Assignments, Tutorials, Mock Tests
PO6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.	-	-
PO7	**Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.	-	-
PO8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.	2	Quiz
PO9	**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.	2	Quiz

Program Outcomes (PO)		Level	Proficiency assessed by
PO10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.	-	-
PO11	**Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.	3	Hands on training
PO12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.	2	Hands on training

How Program Specific Outcomes are Assessed:

Program Specific Outcomes (PSO)		Level	Proficiency assessed by
PSO1	**Software Development and Research Ability:** Ability to understand the structure and development methodologies of software systems. Possess professional skills and knowledge of software design process. Familiarity and practical competence with a broad range of programming language and open source platforms. Use knowledge in various domains to identify research gaps and hence to provide solution to new ideas and innovations.	3	Assignments, Tutorials, Mock Tests
PSO2	**Foundation of mathematical concepts:** Ability to apply the acquired knowledge of basic skills, principles of computing, mathematical foundations, algorithmic principles, modeling and design of computer-based systems in solving real world engineering Problems.	3	Assignments, Tutorials
PSO3	**Successful Career:** Ability to update knowledge continuously in the tools like Rational Rose, MATLAB, Argo UML, R Language and technologies like Storage, Computing, Communication to meet the industry requirements in creating innovative career paths for immediate employment and for higher studies.	3	Assignments, Tutorials, Mock Tests

1: Slight (Low) **2: Moderate (Medium)** **3: Substantial (High)** **- : None**

COURSE CONTENTS

UNIT – I

Introduction: Language Processors, the structure of a compiler, the science of building a compiler, programming language basics.

Lexical Analysis: The Role of the Lexical Analyzer, Input Buffering, Recognition of Tokens, The Lexical-Analyzer Generator Lex, Finite Automata, From Regular Expressions to Automata, Design of a Lexical-Analyzer Generator, Optimization of DFA-Based Pattern Matchers.

UNIT – II:

Syntax Analysis: Introduction, Context-Free Grammars, Writing a Grammar, Top-Down Parsing, Bottom-Up Parsing, Introduction to LR Parsing: Simple LR, More Powerful LR

Parsers, Using Ambiguous Grammars, Parser Generators

UNIT – III:

Syntax-Directed Translation: Syntax-Directed Definitions, Evaluation Orders for SDD's, Applications of Syntax-Directed Translation, Syntax-Directed Translation Schemes, and Implementing L-Attributed SDD's.

Intermediate-Code Generation: Variants of Syntax Trees, Three-Address Code, Types and Declarations, Type Checking, Control Flow, Back patching, Switch-Statements, Intermediate Code for Procedures.

UNIT – IV:

Run-Time Environments: Storage organization**,** Stack Allocation of Space, Access to Nonlocal Data on the Stack, Heap Management, Introduction to Garbage Collection, Introduction to Trace-Based Collection.

Code Generation: Issues in the Design of a Code Generator, The Target Language, Addresses in the Target Code, Basic Blocks and Flow Graphs, Optimization of Basic Blocks, A Simple Code Generator, Peephole Optimization, Register Allocation and Assignment, Dynamic Programming Code-Generation.

UNIT – V:

Machine-Independent Optimizations: The Principal Sources of Optimization, Introduction to Data-Flow Analysis, Foundations of Data-Flow Analysis, Constant Propagation, Partial-Redundancy Elimination, Loops in Flow Graphs.

TEXT BOOKS

1. Compilers: Principles, Techniques and Tools, Second Edition, Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffry D. Ullman, Pearson.

REFERENCE BOOKS

1. Compiler Construction-Principles and Practice, Kenneth C Louden, Cengage Learning.
2. Modern compiler implementation in C, Andrew W Appel, Revised edition, Cambridge University Press.
3. The Theory and Practice of Compiler writing, J. P. Tremblay and P. G. Sorenson, TMH
4. Writing compilers and interpreters, R. Mak, 3rd edition, Wiley student edition.
5. lex & yacc – John R. Levine, Tony Mason, Doug Brown, O'reilly

NPTEL Web Course:

https://onlinecourses.nptel.ac.in/noc19_cs01/preview
https://nptel.ac.in/courses/106108052/

NPTEL Video Course:

https://www.youtube.com/watch?v=Qkwj65l_96I&list=PLEbnTDJUr_IcPtUXFy2b1sGRPsLFMghhS
https://www.youtube.com/watch?v=hzlwgvAdR-s&list=PLG9aCp4uE-s3XepZyd94jGic7qMFa7CW1

RELEVANT SYLLABUS FOR GATE

Lexical analysis, Parsing, Syntax directed translation, Runtime environments, Intermediate and target code generation, Basics of code optimization.

RELEVANT SYLLABUS FOR IES : NA

LESSON PLAN

Session	Week	Topic	Reference
1	Week – 1	**Introduction:** Language Processors	T1
2.		Structure of a compiler	
3.		Science of building a compiler	
4.		Programming language basics	
5.		**Lexical Analysis:** The Role of the Lexical Analyzer	
6.	Week – 2	Input Buffering, Recognition of Tokens	
7.		The Lexical-Analyzer Generator Lex	
8.		Finite Automata, From Regular Expressions to Automata	
9.		Design of a Lexical-Analyzer Generator	
10.		Optimization of DFA-Based Pattern Matchers.	
11.	Week-3	*Types of Compiler	
12.		*Bootstrapping	
13.		**Mock Test-1**	T1
14.		**Bridge Class**	
15.		**UNIT-II : Syntax Analysis:** Introduction	
16.	Week – 4	Context-Free Grammars	
17.		Context-Free Grammars	
18.		Writing a Grammar	
19.		Top-Down Parsing	
20.		Top-Down Parsing…1	
21.	Week – 5	Top-Down Parsing…2	
22.		Bottom-Up Parsing.	
23.		Bottom-Up Parsing..1	
24.		Bottom-Up Parsing..2	T1
25.		Introduction to LR Parsing	
26.	Week – 6	Simple LR	
27.		Simple LR..1	
28.		Simple LR..2	
29.		More Powerful LR Parsers	
30.		Using Ambiguous Grammars	
31.	Week – 7	Parser Generators	
32.		**Bridge Class -2**	
33.		**Syntax-Directed Translation:** Syntax -Directed Definitions	
34.		Evaluation Orders for SDD's	
35.		Applications of Syntax-Directed Translation	

36.	Week – 8	Syntax-Directed Translation Schemes	
37.		Implementing L-Attributed SDD's	
38.		*Implementing S-Attributed SDD's	
39.		**Intermediate-Code Generation:** Variants of Syntax Trees	
40.		Three-Address Code	T2
41.	Week-10	Types and Declarations	
42.		Type Checking	
43.		Control Flow	
44.		Back patching,	
45.		Switch-Statements	
46.	Week -11	Intermediate Code for Procedures.	
47.		**Bridge Class -3**	
48.		**Run-Time Environments:** Storage organization	
49.		Stack Allocation of Space	
50.		Access to Nonlocal Data on the Stack	
51.	Week – 12	Heap Management	
52.		Introduction to Garbage Collection	
53.		Introduction to Trace-Based Collection.	
54.		**Code Generation:** Issues in the Design of a Code Generator	
55.		The Target Language	
56.	Week – 13	ddresses in the Target Code	
57.		Basic Blocks and Flow Graphs	T2
58.		Optimization of Basic Blocks	
59.		A Simple Code Generator	
60.		Peephole Optimization	
61.	Week-14	Register Allocation and Assignment	
62.		Dynamic Programming Code-Generation	
63		**Mock-Test2**	
64.		**Bridge Class-4**	
65.		**Machine-Independent Optimizations:**	
66.	Week – 15	The Principal Sources of Optimization	
67.		Introduction to Data-Flow Analysis	
68.		Foundations of Data-Flow Analysis	
69.		Constant Propagation	
70.		Partial- Redundancy Elimination	
71.	Week-16	Loops in Flow Graphs.	
72.		Revision- Unit1	
73.		Revision – Unit2	
74.		Revision – Unit3	
75.		Revision – Unit4	

Mapping Course Outcomes Leading to the Achievement of Program Outcomes and

Program Specific Outcomes:

Course Outcomes	Program Outcomes (PO)												Program Specific Outcomes (PSO)		
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	3	3	3	3	-	-	1	2	-	2	1	3	3	2
CO2	3	3	3	3	3	-	-	1	2	-	2	1	3	3	2
CO3	3	3	3	3	3	-	-	1	2	-	2	1	3	3	2
CO4	3	3	3	3	3	-	-	1	2	-	3	1	3	3	2
CO5	3	3	3	3	3	-	-	1	2	-	3	1	3	3	2
AVG	3.00	3.00	3.00	3.00	3.00			1.00	2.00		2.40	1.00	3.00	3.00	2.00

1: Slight (Low) **2: Moderate (Medium)** **3: Substantial (High)** **- : None**

QUESTION BANK

Descriptive Questions

UNIT-I:

Long questions

1. What are various phases of compiler .Explain each phase in detail. Write down the output of each phase for expression a:= b+c *50. [L2:Understand]
2. Explain how lex program will perform the lexical analysis for the following patterns in C: identifiers, comments, numerical constants and arithmetic operators [L2:Understand]
3. Construct minimum state DFAs for the following regular expressions.
   (a+b)* a (a+b)
   (a+b)* a (a+b) (a+b) [L5:Evaluate]
4. What is LEX? Explain, in detail, different sections of LEX program. [L2:Understand]
5. Consider the following grammar
   E->TE'
   E'-> +TE'| ^
   T-> FT'
   T'->*FT'| ^
   F-> (E)|id

   Construct the predictive parsing table and show the stack implementation for the input string id+id * id. [L5:Evaluate]

Short questions

1. Write regular expressions for the set of words having a,e,i,o,u appearing in that order, although not necessarily consecutively [L2:Understand]
2. Explain about parser and its types [L2:Understand]
3. Define Pass and Phase [L1:Remember]
4. Explain ambiguous grammar with examples [L2:Understand]
5. Explain the issues of lexical analyzer. [L2:Understand]

Objective questions

1. A finite sequence of symbols drawn from alphabet is
   a) Symbol b) String c) Node d) None
2. Lexical Analyzer divides the source string into
a) Tokens b) Pattern c) Symbols d) Strings
3. The action of passing the source program into the proper syntactic classes is known as [ ]
A) Lexical analysis B) Syntax analysis C) Interpretation analysis D) Parsing
4. FOLLOW is applicable for [ ]
   A) Only non-terminals B) Either terminals or non terminals
   C) Only terminals D) Neither terminals nor non-terminals
5. System program such as compiler are designed so that they are [ ]
   A) Re-entrable B) Serially usable C) Recursive D) non re-usable

1. _____ phase of the compiler reads the source program
2. _____ file consists of tabular representation of the transmission diagram constructed for expressions of specification file.
3. Semantic analysis is related to _____ phase.
4. In _____ approach , a parse tree is generated form Parent node
5. A _____ is a tree in which node represents an operator and children of that node represent operands.

UNIT-II:
Long Questions

1. Design LALR(1) parser for the following grammar?
   S → aAdj bBdjaBcjbAc
   A → e
   B → e. [L6:Create]
2. Consider the following grammar: [L6:Create]
   S → A
   A → BA|E
   B → aB|b.
   (a) Construct LR(1) parser.
   (b) Find the moves made by the LR(1) parser on the input string: aabb.
3. Explain about error recovery in parsing. [L2:Understand]
4. Eliminate ambiguity if any from the following grammar for boolean expressions.
   Bexpr→ bexpr or bterm|bterm
   bterm → bterm and bfactor|bfactor
   bfactor →nst factor|(bexpr)|true|false. [L3:Evaluate]
   Where or, and, not (,), true, false are terminals in the grammar
5. Give the LALR parsing table for the below grammar [L3:Evaluate]
   S→L=R
   S→R
   L→ *R
   L→id
   R→L

Short questions

1. Define LR(0) items in bottom up parsing? [L1:Remember]
2. List down the conflicts during shift-reduce parsing. [L1:Remember]
3. Explain about handle pruning? [L2:Understand]
4. Define goto function in LR parser with an example? [L1:Remember]
5. Explain types of LR parsers? [L2:Understand]

Objective type questions

1. The most efficient among the following is [ ]
   a) CLR b) LALR c) SRP d) SLR
2. A grammar will be meaning less [ ]

A) If the left hand side of the production is a single terminal
B) If the terminal set and non terminal sets are disjoint
C) If the left hand side production has more than two non terminals
D) If the left hand side production has non terminal

3. The minimum value of k in LR(k) is [ ]
   A) 1 B) 0 C) 2 D) 3
4. Which of the following is true? [ ]

A) LALR(1) requires less space compare with LR(1)
B) LALR(1) is more powerful than LR(1)
C) LALR(1) requires more space compare with LR(1)
D) LALR(1) is a powerful as an LR(1)

5. Which of the following is not a bottom up parser?

(a)LALR (b) Predictive parser (c)Canonical LR (d)SLR1. LALR

1.Stands for _____
2. Among all the LR(1) parser, the strongest is _____
3. YACC specifications consist of _____ sections
4. _____parser is bottom up and efficient parser
5. In bottom up parser, shift always_____

. **UNIT-III:**

Long questions

1. Write the semantic actions to generate the three address code for function call and return statements in C language? [L2:Understand]
2. Consider the following three address code in a basic block. [L3:Evaluate]

(a) t1 = j -1
(b) t2 = 4 * t1
(c) temp = A[t2]
(d) t3 = j
(e) t4 = j + 1
(f) t5 = 4 * t3
(g) t6 = A[t5]
(h) t7 = j - 1

(i) t8 = 4 * t7
(j) A[t8] = t6
(k) t9 = j
(l) t10 = j+1
(m) t11 = 4*t9
(n) A [t11] = temp

(a) Perform copy propagation. You need to do necessary data ow analysis to make sure that the propagation can be done correctly.

(b) Perform dead code elimination. You need to do necessary analysis to make sure that the elimination can be done correctly. Also, assume that after the basic block, A[i], for all i, are alive and no other variables are alive.

3. Write semantic actions for the following? [L6:Create]

(a) for - loop
(b) repeat - until loop.

4. Illustrate the following techniques with suitable examples [L3:Apply]

(a) Constant folding
(b) constant propagation
(c) reduction in strength
(d) Elimination of induction variables.

5. Describe the use of symbol tables in the code generation process. That is, describe how symbol tables are used to determine scope of variables and how this acts the code that performs the run-time access to load/store values in subroutine frames or in static data space [L2:Understand]

Short questions

1.Define back patching? [L1:Remember]
2.List various forms of target programs? [L1:Remember]
3.Define basic blocks? [L1:Remember]
4.Explain general activation record? [L2:Understand]
5. State the difference between heap storage and hash table? [L4:Analyze]

Objective questions

1. Pick the odd man out.
(a) Syntax Tree
(b) Triples
(c) Quadruples
(d) Indirect Triples

2. Which of the following translation program converts assembly language programs to object program?
(a) Assembler
(b) Compiler
(c) Linker
(d) Pre-processor

3. In Quadruple notation _____ fields are used to represent operands.
(a) 3
(b) 4
(c) 2
(d) 1

4. Which of the following is not a type expression?
(a) Char
(b) Float
(c) Int
(d) Main

5. Synthesized attribute can be easily simulated by a _____.
(a) LL Grammar
(b) Not LR Grammar
(c) Ambiguous Grammar
(d) LR Grammar

1. The intermediate code can be in _____ form
2. _____ is performed in Semantic Analysis
3. In n tuple notation _____ fields are used to represent operands
4. An operator is _____ is the same operator name used in two different operations
5. The substitution of values for names whose values are constant is known as _____.

UNIT-IV:

Long Questions

1. Explain in detail about global optimization. [L2:Understand]
2. Discuss in detail about live variable analysis [L2:Understand]
3. Discuss in detail about data flow analysis [L2:Understand]
4. Determine the pre-dominant block of block B2 in the program flow graph from the following code
   …../* Block B0*/
   o to 100 /* Block B1*/
   100 go to 10 /* Block B2*/ [L4:Analyze]
5. Describe in detail about next use information about names in basic blocks. [L2:Understand]

Short questions

1. Define the 3 areas of code optimization? [L1:Remember]
2. Define local optimization? [L1:Remember]
3. Define constant folding? [L1:Remember]
4. Define Common Sub expressions? [L1:Remember]
5. Define peephole optimization? [L1:Remember]

Objective questions

1.DAG has .
(a) only one root (b) any number of roots (c) no root (d) no does at all

2.The method that merges the bodies of two loops is.
(a) loop nesting (b) costant folding (c) loop ramming (d) loop unrolling
3.Movement of the code from inside to outside is.
(a) coding (b) frequency reduction (c) constant folding (d) costant substitution

4.The following cannot be used to identify loops.
(a) depth first ordering (b) reducible graphs (c) dominator (d) flow chart

5.The modification that decreases the amount of code in a loop is.
(a) constant folding (b) constant reduction (c) code motion (d)reduction streng

1 The runtime representation of an object program in the logical address space consists of _____.
2. The static data objects are created at _____.
3. The activations of procedures during the running of an entire program by a tree called _____.
4. Activation records are sometimes called _____.

5. If a transformation of a program performed by locking only at the statements in a basic blocks, called _____.

UNIT-V:
Long questions

1. Represent DAG for register allocation in detail [L4:Apply]
2. Describe different object code forms and illustrate this with an example [L2:Understand]
6. Explain in detail about register allocation and assignment of generic code [L2:Understand] generation algorithms.
3. Discuss in detail about the issues in the design of a code generator [L2:Understand]
4. Explain in detail about machine dependent code optimization. [L2:Understand]

Short Questions

1. Write in detail the issues in the design of code generator [L2:Understand]
2. Explain the role of code generator in a compiler? [L2:Understand]
3. How will you map names to values [L1:Remember]
4. How do you calculate the cost of an instruction? [L2:Understand]
5. Mention the properties that a code generator should possess [L1:Remember]

Objective Questions
1.Which of the following is related to synthesis phase?
A) Syntax analysis B) Code generation

C) Lexical analysis D) Semantic analysis

2.Tree Translation scheme is _____.
(a) Set of tree rewriting rules.
(b) Set of all traversals
(c) Translating all the operations of the tree
(d)Translation of tree in to arrays

3. The advantage of generating a code from DAG is _____.
(a) Easily see how to rearrange the order of the final computation sequence
(b) Eliminating sub expressions.
(c) Eliminating loop invariant variables
(d) Eliminating induction variables

4. Peephole optimization is to eliminate _____.
(a)Constant folding
(b) Jumps over jumps
(c) Copy propagation
(d) Elimination of dead variables

5. The input to the code generator is a_____.
(a) Sequence of tree at lexical level
(b) Sequence of tree at semantic level
(c) Sequence of assembly language instruction
(d) Sequence of machine idioms

1. The relative address for a field name is relative to the _____ for that record.
2. The advantage of generating a code from DAG_____
3. The input to the code generator is a_____
4. Addressing modes involving registers have_____
5. The output of code generator is_____

Websites

- Free Compiler e-books : http: // more-compiler.blogspot.com/
- Lex and Yacc Links
- A Compact Guide to Lex and Yacc by Tom Niemann
- The Lex and Yacc Page
- Assembly Language
- http://www.drpaulcarter.com/pcasm/

Expert Details

Sebastian Hack

Prof. Dr.Sebastian Hack is a professor of computer science at Saarland University. His work focuses on compiler construction, especially code generation, automatic vectorization and parallelization. Before, he was an assistant professor at Saarland University, a Post-Doc at EPFL, Switzerland in the LAMP lab and a Post-Doc at ENS Lyon, France in the COMPSYSproject. He received his PhD in 2006 from Karlsruhe University, Germany and his Diploma degree also from Karlsruhe University in 2004.

Journals

Optimal Register Allocation for SSA-form Programs in polynomial Time

Hack, S. and Goos, G.

Information Processing Letters, 98 (4): 150–155, 2006.

Seminar Topics

1) Phases Of Complier
2) Ll(1) Parsing
3) CLR Parsing
4) Intermediate Code Generation
5) Context Sensitive Language
6) Storage Allocation Strategies
7) Data Flow Analysis Of Flow Graphs
8) Object Code Forms

Case Studies/Small Projects

Project 1 – Lexical Analyzer using the Lex Unix Tool

Project 2--Static Analysis and Compiler Design for Idempotent Processing