

COURSE DESCRIPTION

Dept., Number	CSC 151	Course Title	Compiler Construction
Semester hours	3	Course Coordinator	Anne-Louise Radimsky
		URL (if any):	http://gaia.ecs.csus.edu/~radimsky/

Catalog Description

A practical approach to compiler design and implementation. Organization of a compiler, algorithms for lexical, syntactic and semantic analysis, recursive descent and/or LALR parsing, organization of symbol tables, error detection and recovery, object code generation. Modular design will be emphasized. Prerequisites: Full major status in CSC or CPE and at least a C-grade in CSC 136; however, it may be taken concurrently.

Textbook

Kenneth C. Louden, Compiler Construction: Principles and Practice, Course Technology, 1997.

Course Goals

To develop an understanding of the practical aspects of compiler design and construction, including:

1. At least one parsing technique and its conditions of application.
2. Error recovery.
3. Code generation.
4. Translation of typical programming language constructs.

Prerequisites by Topic

Thorough understanding of:

- Advanced data structures.
- An object-oriented programming language.

Basic understanding of:

- Regular expressions, finite automata, context-free grammars, pushdown automata.
- Concept of parsing.
- Representation of programming language syntax, BNF, EBNF, syntax diagrams.

Major Topics Covered in the Course

1. Introduction to compilers (1 hour).
2. Source handling (1 hour).
3. Symbol tables (2 hours).
4. Lexical analysis, including the use of finite automata (3 hours).
5. Syntactic analysis, including simulation of push down automata (10 hours).

6. The use of compiler-writing tools: automated parsers and lexical analyzers (6 hours).
7. Error recovery (4 hours).
8. Semantic analysis (8 hours).
9. Translation of source to an intermediate language, e. g., Polish postfix, expression tuples (3 hours).
10. Translation of intermediate language to object code (3 hours).
11. Run-time environments (2 hours).
12. Optimization of object code (2 hours).

Outcomes

Thorough understanding of:

- The concepts of scanning and parsing.
- The organization of a compiler.
- Scanning and the use of scanner generators.
- Modular development of a significant programming system.

Basic understanding of:

- Parsing; bottom-up parsing techniques including recursive and LL (1); top-down parsing including SLR (1) and LALR (1).
- Error recovery associated with various parsing techniques.
- Semantic analysis, attribute grammars.
- Method of intermediate code representation.
- Translation of some basic language constructs such as recursion, parameter passing.

Exposure to:

- Code optimization.

Laboratory Projects

Implementation of several portions of a compiler. Such project(s) might include the implementation of:

1. A source handler (2 weeks).
2. A lexical analyzer (minimum 2 weeks).
3. A syntax analyzer (minimum 3 weeks).
4. A semantic analyzer (minimum 2 weeks).
5. A code generator (minimum 3 weeks).

Estimated Curriculum Category Content (Semester hours)

<i>Area</i>	<i>Core</i>	<i>Advanced</i>	<i>Area</i>	<i>Core</i>	<i>Advanced</i>
Algorithms		0.5	Data Structures		0.5
Software Design		0.5	Prog. Languages		1.0
Comp. Arch.		0.5			

Oral and Written Communications

No significant component.

Social and Ethical Issues

No significant component.

Theoretical Content

1. Regular expressions and finite automata, automatic generation of lexical analyzers (4 hours).
2. Formal techniques for parsing context-free languages (6 hours).

Problem Analysis

In order to write some portions of a compiler (lexical analyzer, syntax analyzer, semantic analyzer, or code generator), the student must analyze how the techniques discussed in class relate to the particular language to be processed.

Solution Design

Students implement a compiler for a simple programming language using object-oriented techniques and a highly modular design.

/aa