The University of Georgia

Department of Computer Science

Course Information Sheet

CSCI 4570

Compilers

Brief Course Description
(50-words or less)

Design and implementation of compilers for high-level programming languages. Topics include all phases of a typical compiler, including scanning, parsing, semantic analysis, intermediate code generation, code optimization, and code generation. Students design and develop a compiler for a small programming language. Emphasis is placed on using compiler development tools.

Extended Course Description / Comments

In this course, the students study the principles of compiler design and implementation. The primary emphasis is placed on the organization of a typical compiler pipeline, especially focusing on the stages of a compiler front-end. The course begins with lexical analysis and the construction of scanners, then moves on to various top-down and bottom-up parsing algorithms, semantic analysis and type checking, syntax-directed translation, and intermediate code generation. The course content also includes symbol tables, error recovery, and runtime systems. Furthermore, the course includes an overview of code optimization and target code generation.
Each student implements a compiler for a small programming language, usually a subset of a well-known high-level programming language. The students learn to use compiler-compiler tools, including scanner and parser generators. The programming project is split into a few parts to make the development a larger program manageable.

Pre-Requisites and/or Co-Requisites

CSCI 4720
Computer Architecture

Approved Textbooks
(if more than one listed, the textbook used is up to the instructor's discretion)

Author(s): Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman
Title: Compilers: Principles, Techniques, and Tools, Addison-Wesley, 2007.
Edition: 2-nd edition
ISBN-13: 978-0321486813

Specific Learning Outcomes (Performance Indicators)

This course presents a survey of topics in compiler construction most relevant to students studying computer science. At the end of the semester, all students will be able to do the following:

1. Define the phases of a typical compiler, including the front- and back-end.
2. Identify tokens of a typical high-level programming language; define regular expressions for tokens and design; implement a lexical analyzer using a typical scanner generator.
3. Explain the role of a parser in a compiler and relate the yield of a parse tree to a grammar derivation; design and implement a parser using a typical parser generator.
4. Apply an algorithm for a top-down or a bottom-up parser construction; construct a parser for a small context-free grammar.
5. Explain the role of a semantic analyzer and type checking; create a syntax-directed definition and an annotated parse tree; describe the purpose of a syntax tree.
6. Explain the role of different types of runtime environments and memory organization for implementation of typical programming languages.
7. Describe the purpose of translating to intermediate code in the

compilation process.

8. Design and implement an intermediate code generator based on given code patterns.

Relationship Between Student Outcomes and Learning Outcomes

		Student Outcomes										
		a	b	c	d	e	f	g	h	i	j	k
Learning Outcomes	1			•								•
	2			•						•	•	•
	3			•						•	•	•
	4			•						•	•	•
	5			•						•	•	•
	6			•						•		•
	7			•								•
	8			•						•	•	•

Program Outcomes

a. An ability to apply knowledge of computing and mathematics appropriate to the discipline.
b. An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution.
c. An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs.
d. An ability to function effectively on teams to accomplish a common goal.
e. An understanding of professional, ethical, legal, security and social issues and responsibilities.
f. An ability to communicate effectively with a range of audiences.
g. An ability to analyze the local and global impact of computing on individuals, organizations, and society.
h. Recognition of the need for and an ability to engage in continuing professional development.
i. An ability to use current techniques, skills, and tools necessary for computing practice.
j. An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices.
k. An ability to apply design and development principles in the construction of software systems of varying complexity.

Major Topics Covered
(Approximate Course Hours)

3 credit hours = 37.5 contact hours
4 credit hours = 50 contact hours

Note: Exams count as a major topic covered

Organization of a typical compiler (1.5-hours)
Token identification and specification (4-hours)
Scanner generator construction (3-hours)
Lexical analyzer design and implementation (3-hours)
Context-free grammar, derivations, parse trees (4-hours)
Algorithms for top-down parsing (3-hours)
Algorithms for bottom-up parsing (4-hours)
Syntax error detection and recovery (2-hours)
Parser specification using a typical parser generator (3-hours)
Syntax trees and symbol tables (2.5-hours)
Semantic analysis, attribute grammars, type checking (3-hours)
Semantic errors detection (4-hour)
Runtime environments (4-hours)
Intermediate languages and syntax-directed translation (3-hours)
Code patterns and intermediate code generation (3-hours)
Target code optimization and generation (3-hours)

Assessment Plan for this Course

Each time this course is offered, the class is initially informed of the Course Outcomes listed in this document, and they are included in the syllabus. At the end of the semester, an anonymous survey is administered to the class where each student is asked to rate how well the outcome was achieved. The choices provided use a 5-point Likert scale containing the following options: Strongly agree, Agree, Neither agree or disagree, disagree, and strongly disagree. The results of the anonymous survey are tabulated and results returned to the instructor of the course.

The course instructor takes the results of the survey, combined with sample student responses to homework and final exam questions corresponding to course outcomes, and reports these results to the ABET committee. If necessary, the instructor also writes a recommendation to the ABET committee for better achieving the course outcomes the next time the course is offered.

How Data is Used to Assess Program Outcomes

Each course Learning Outcome, listed above, directly supports one or more of the Student Outcomes, as is listed in "Relationships between Learning Outcomes and Student Outcomes". For CSCI 4570, Student Outcomes (c) (i) (j) and (k) are supported.

Course Master

Dr. Krzysztof Kochut

Course History

01/2003 Course Uploaded into CAPA
02/2012 Course Information Sheet Updated