School of Computing
UNIVERSITY OF GEORGIA

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
Required, Elective or Selected Elective	Required Course
Approved Textbook	Crafting A Compiler, by Charles N. Fischer, Ron K. Cytron, and Richard J. LeBlanc, Jr., Pearson, 2010. ISBN-13: 978-0136067054
Specific Learning Outcomes (Performance Indicators)	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; design and implement a lexical analyzer using a scanner generator. 3. Explain the role of a parser in a compiler; apply an algorithm for a top-down and a bottom-up parser construction. 4. Design and implement a parser for a small programming language using a parser generator; create of a syntax tree. 5. Explain the role of a semantic analyzer and type checking; create a syntax-directed definition and an annotated parse 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; design and implement an intermediate code generator.

ABET Learning Outcomes

A. Graduates of the program will have an ability to: Analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions.
B. Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline.
C. Communicate effectively in a variety of professional contexts.
D. Recognize professional responsibilities and make informed judgments in computing practice based on legal and ethical principles.
E. Function effectively as a member or leader of a team engaged in activities appropriate to the program's discipline.
F. Apply computer science theory and software development fundamentals to produce computing-based solutions.

Relationship Between Student Outcomes and Learning Outcomes

	ABET Learning Outcomes						
Specific Learning Outcomes		**A**	**B**	**C**	**D**	**E**	**F**
	1		●	●			●
	2	●	●				●
	3	●	●				●
	4	●	●				●
	5	●	●				●
	6	●	●				●
	7			●			●

Major Topics Covered

1. Compiler fundamentals (Familiarity)
   a. Identify the role of compilers in computing
   b. Contrast the source code and the target code (low-level code)
   c. Identify and explain the purpose of the phases of a typical compiler
2. Lexical analysis (Knowledge level: varies by topic)
   a. Identify and create regular expressions for classes of tokens, in typical programming languages (Usage)
   b. Apply scanner generator construction algorithm for a set of tokens (Assessment)
   c. Design and implement a lexical analyzer for a small programming language (Assessment)
3. Parsing (Knowledge level: varies by topic)
   a. Parse input string using a Context-free grammar; create a parse tree for an input string derivations (Usage)
   b. Define and apply a top-down parsing algorithm (Usage)
   c. Define and apply a bottom-up parsing algorithm (Usage)
   d. Design and implement a parser for a small programming language (Assessment)

e. Define syntax errors and design and implement syntax error detection and recovery (Assessment)
f. Design and implement the creation of a syntax tree for a small programming language (Assessment)

4. Semantic analysis (Knowledge level: varies by topic)
   a. Define the role of a semantic error analyzer (Usage)
   b. Define semantic errors for a small programming language (Usage)
   c. Define the role of a symbol table; design and implement (Assessment)
   d. Design and implement a semantic analyzer for a small programming language (Assessment)
5. Runtime environments (Usage)
   a. Explain the role of a runtime environment for executing programs
   b. Differentiate stack and heap allocation
6. Intermediate code generation (Knowledge level: varies by topic)
   a. Describe the role of intermediate languages and syntax-directed translation (Usage)
   b. Define the types of intermediate languages (Usage
   c. Design a strategy for memory allocation (Assessment)
   d. Describe the role and types of code optimization (Usage)
   e. Design and implement an intermediate code generator for a small programming language (Assessment)

Knowledge Levels

The following is the ACM's categorization of different levels of mastery: Assessment, Usage, and Familiarity. Note that Assessment encompasses both Usage and Familiarity, and Usage encompasses Familiarity.

Familiarity: The student understands what a concept is or what it means. This level of mastery concerns a basic awareness of a concept as opposed to expecting real facility with its application. It provides an answer to the question "What do you know about this?"

Usage: The student is able to use or apply a concept in a concrete way. Using a concept may include, for example, appropriately using a specific concept in a program, using a particular proof technique, or performing a particular analysis. It provides an answer to the question "What do you know how to do?"

Assessment: The student is able to consider a concept from multiple viewpoints and/or justify the selection of a particular approach to solve a problem. This level of mastery implies more than using a concept; it involves the ability to select an appropriate approach from understood alternatives. It provides an answer to the question "Why would you do that?"

Course Master Dr. Krzysztof J. Kochut

Modified

Approved Yes