School of Computing
UNIVERSITY OF GEORGIA

Course Information Sheet
CSCI 1301
Introduction to Computing and Programming

Brief Course Description (50-words or less)	Algorithms, programs, and computing systems. Fundamental techniques of program development and supportive software tools. Programming projects and applications in a structured computer language.
Extended Course Description / Comments	This course is a rigorous introduction to problem solving using fundamental programming techniques: variables, operators, expressions, decision statements, loops, nested statements, arrays, methods, objects, classes, inputs, and outputs. This course includes programming projects incorporating algorithm design and implementation with a structured computer language and hands-on experience creating, testing, and debugging software. This course is typically the first major-related course taken by computer science majors or anyone interested in learning how to program.
Pre-Requisite	MATH 1113: Precalculus
Required, Elective or Selected Elective	Required Course
Approved Textbooks (if more than one listed, the textbook used is up to the instructor's discretion)	Author(s): Walter Savitch Title: Java: An Introduction to Problem Solving and Programming Edition: 7th Edition or 8th Edition ISBN-13: 978-0133841081 (7th Edition) 978-0134448398 (8th Edition)
Specific Learning Outcomes (Performance Indicators)	This course presents fundamental programming topics in a structured programming language. At the end of the semester, all students will be able to do the following: 1. Explain and describe basic computing concepts required for programming. 2. Utilize software development tools, including tools for editing, compiling, testing, running, and debugging software solutions. 3. Describe and utilize basic language constructs, including data types, input, output, variables, constants, assignment statements, arithmetic and boolean expressions. 4. Trace, design, and implement software solutions to non-trivial problems using control flow structures. 5. Trace, design, and implement software solutions to non-trivial problems using basic data structures. 6. Trace, design, and implement software solutions to non-trivial problems using object-oriented programming techniques. 7. Combine control flow statements, basic data structures, and object-oriented programming techniques to create an interactive software solution to a problem.
ABET Learning Outcomes	Graduates of the program will have an ability to:

A. 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.

NOTE: In the construction of the student learning outcomes for this course, the instructors interpreted "computing requirements" in (B) as the functional requirements for a software solution and not as specific hardware requirements for the target platform; likewise, the phrase "[a]pply computer science theory" in (F) was interpreted as using computer science principles.

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. Computing Basics (Knowledge level: Familiarity)
   a) Explain the difference between computer software and hardware.
   b) Describe various types of memory (primary, secondary), give examples of each, and understand the importance of the various levels of the memory hierarchy.
   c) Explain encoding schemes for various types of data such as: whole numbers, characters, images, videos, etc.
2. Development Tools and Accepted Practices (Knowledge level: varies by topic)
   a) Explain the relationships between: JRE, JDK, JVM, IDE, and the Java compiler. (Familiarity)
   b) Utilize the debugger to trace and identify logical errors in a software solution. (Usage)
   c) Use an integrated development environment (IDE), such as Eclipse, to create, compile, and execute a non-trivial software solution. (Usage)

d) Create source code that adheres to established style guidelines. (Usage)
e) Create class, interface, and inline documentation for a complete object-oriented software solution. (Usage)
f) Create a set of dedicated test methods for classes and other methods. (Usage)
g) Read and interpret UML diagrams for a set of classes (Usage)

3. Language Basics (Knowledge level: varies by topic)
Specific topics that need to be covered include, but are not limited to, data types, input, output, variables, constants, assignment statements, arithmetic and boolean expressions.
a) Declare and initialize variables of different types. (Assessment)
b) Explain the need for multiple whole number and floating-point data types. (Familiarity)
c) Given non-trivial arithmetic and boolean expressions containing various operations and data types, calculate the resulting value and data type. (Usage)
d) Perform basic input and output operations. (Usage)
e) Explain the differences between primitive and reference types. (Familiarity)

4. Control Flow (Knowledge level: Usage)
Specific topics that need to be covered include, but are not limited to, if-else statements, switch statements, for-loop statements, while-loop statements, do-while-loop statements, and variable scope.
a) Draw a complete execution trace (memory map) for non-trivial code examples containing combinations of the major control flow statements involving at least three levels of nesting.
b) Design and implement non-trivial software solutions that require the use of combinations of the major control flow statements involving at least three levels of nesting.

5. Basic Data Structures (Knowledge level: varies by topic)
Specific topics that need to be covered include, but are not limited to, Arrays and Strings.
a) Draw a complete execution trace (memory map) for non-trivial code examples containing combinations of the basic data structures. For example, 2-D arrays, ragged arrays, and arrays containing references to objects. (Usage)
b) Design and implement non-trivial software solutions that require the use of combinations of the basic data structures. (Usage)
c) Linear search, binary search, and quadratic comparison-based sorting algorithms. (Familiarity)

6. Object-Oriented Programming (Knowledge level: Usage)
Specific topics that need to be covered include, but are not limited to, classes, objects, methods, constructors, static variables and methods, method overloading, access modifiers, mutability, information hiding and encapsulation, access

modifiers, and pass-by-value.

a) Draw a complete execution trace (memory map) for non-trivial code examples incorporating object-oriented concepts. Specifically, object composition and aggregation must be covered. (Usage)
a) Design and implement non-trivial software solutions that incorporate object-oriented concepts. Specifically, object composition and aggregation must be covered. (Usage)

7. Aggregate (Knowledge level: Assessment)
   a) Combine control flow statements, basic data structures, and object-oriented programming techniques to create an interactive software solution to a problem.

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. Bradley Barnes

Modified 8/10/2019 by Dr. Cotterell and Dr. Barnes