A Model Curriculum for K-12 Computer Science

Level III Objectives and Outlines

Bill Madden
Bergen Community College
400 Paramus Road
Paramus, NJ 07652
201-493-3573
bmadden@bergen.edu

Anita Verno
Bergen Community College
400 Paramus Road
Paramus, NJ 07652
201-447-7909
averno@bergen.edu

Debbie Carter
Lancaster County Day School
725 Hamilton Road
Lancaster, PA 17603
717-392-2916
carterd@e-lcds.org

Steve Cooper
Saint Joseph's University
5600 City Avenue
Philadelphia, PA 19131
610-660-1561
scooper@sju.edu

Thomas J. Cortina
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
412-268-3514
tcortina@cs.cmu.edu

Ron Cudworth
Monroe Career & Technical Institute
Laurel Lake Dr.
Bartonsville, PA 18321
(570) 629-2001
RCudwort@monroecti.org

Barb Ericson
Georgia Tech
801 Atlantic Drive
Atlanta, GA, 30332
ericson@cc.gatech.edu

Elizabeth Parys
Math/CS Teacher
Retired
eparys@earthlink.net

Table of Contents

Introduction

Overview....3

About the Level III Objectives and Outlines....4

A Note about the CSTA Source (Web Repository)....5

Topic 1: Program Design and Problem Solving....6

Topic 2: Data Structures....9

Topic 3: Discrete Mathematics – Logic, Functions and Sets....11

Topic 4: Design for Usability....13

Topic 5: Fundamentals of Hardware Design....15

Topic 6: Levels of Language, Software, and Translation....18

Topic 7: Limits of Computing....20

Topic 8: Principles of Software Engineering....22

Topic 9: Social Issues....24

Topic 10: Careers in Computing....26

Acknowledgements....28

Appendix: Facets of Computer Science....29

Computer Hardware....29

Using Computer Software....29

Solving Problems by Developing Software....30

Computers, People, and Society....31

Table....32

Introduction

Overview

The ACM Model Curriculum for K-12 Computer Science[*] provides a broad outline from which a K-12 computer science curriculum can be developed. The Model Curriculum was a response to the view that computer science education is not clearly defined or well-established at the K-12 level. A national computer science curriculum that stabilizes the objectives and content of computer science has implications beyond K-12 education. It will assist students with further study in computer science, information technology, information systems, and computer engineering, and will propel the larger national effort to build and maintain America's position as a global leader in technological knowledge and innovation.

The Model Curriculum is a framework, providing an overview of concepts, grouped together into four spiraling levels:

- Level I: Foundations of Computer Science – recommended for students in grades K-8
- Level II: Computer Science in the Modern World – recommended for students in grades 9 or 10
- Level III: Computer Science as Analysis and Design – recommended for students in upper grades
- Level IV: Topics in Computer Science – recommended for students in upper grades

The Model does not contain detailed objectives for each concept, hence the importance of supplemental implementation documents.

The Level III course, broadly described in the Model, focuses on introductory computer science analysis and design concepts. This Objectives and Outlines document continues efforts by the ACM and the Computer Science Teachers Association[†] (CSTA) to create a comprehensive body of resources to support the implementation of a national K-12 computer science curriculum. It provides learning objectives, detailed focus points, assessment measures and sample educational activities for each topic in the Level III course. We believe the standards suggested in the Level III course are both educationally appropriate and accessible regardless of economic pressures within individual educational systems. It is hoped that this document can be used to encourage a uniform level of course content across the United States.

The Level III course places analysis and design front and center in the study of computer science. As such, it is targeted to high school students who may be interested in pursuing further study in the computing disciplines beyond high school. It is not intended to be a first course in computer science. While it may be a final course for some students, the goal is to present the various facets of computer science in a broad and compelling manner so as to excite students and encourage continued study of the computing disciplines. Level III establishes a framework of topics that are typically introduced in depth at the undergraduate level. The authors feel it is important to move the clock forward, so that computer science students entering college have a real understanding of the computing disciplines and a solid background for further study.

---

[*] Tucker, Allen. (2006). A Model Curriculum for K-12 Computer Science: Final Report of the ACM K-12 Task Force Curriculum Committee, 2nd Edition. Association for Computing Machinery, New York, New York.

[†] The Computer Science Teachers Association is a membership organization that supports and promotes the teaching of computer science and other computing disciplines by providing opportunities for K-12 teachers and students to better understand the computing disciplines and to more successfully prepare themselves to teach and to learn.

The range of topics contained in the Level III course suggests a breadth-first approach, which is appropriate for students requiring an understanding of the scope of computer science. There has been substantial debate as to the relative value of breadth-first and depth-first approaches to learning a subject. Ideas for overcoming perceived deficiencies of an exclusively breadth-first approach might include a project-based methodology in which small workgroups would select from among the topics and investigate their chosen topics more deeply, or, very naturally, if the teacher has a particular expertise in one or more topic areas, s/he may wish to pursue those topics more deeply.

Level III is designed as a year-long standalone course taught by a qualified computer science teacher. This course treats the major design and analysis issues in computer science with a rigor and depth similar to that of any standard high school science or college-prep math course. Therefore, the ideal level of expertise of the teacher of a Level III course is the same as in any of these subject areas, i.e. that the teacher has majored in the particular subject area as an undergraduate / graduate student. This is only fitting since it is the intent of the K-12 Model that Computer Science should take its place as an essential high school course in the 21st century.

To assist teachers in delivering the specified objectives of the Level III course, CSTA has created the CSTA Source, a Web repository that contains activities, lesson plans, and assessments. These resources are matched to the suggested topics and can be used for both professional development and in-class activities. CSTA also provides oversight to organizations that are providing teacher professional development based upon the Model Curriculum recommendations.

About the Level III Objectives and Outlines

The document contains 10 main topics. For each main topic there is a general description, statement of necessary resources, learning objectives, assessment guidelines, a list of focus areas that fall within the main topic area, and a sample activity or two that suggests the anticipated level of student learning. The sample activities are meant to be *representative*; they are not to be taken as required implementations of the student learning objectives. Careful consideration must be given to choosing a set of activities appropriate to a given classroom situation. Additional activities, cross-referenced to the Level III topics, can be found in the CSTA Source, a Web repository.

This document is not a curriculum. It provides topics which need to be included in the Level III course. The language of this document speaks to the educators who will actually deliver this material. This is done to stress the immediacy and usability of the material that comprises Level III. Each teacher will determine depth of coverage and shape the course to meet the needs of the students. The language also speaks to the curriculum planners, who can use the learning objectives to standardize curriculum across a school, district, and state.

The order of the topics in the document does not imply order of coverage. The topic numbers are to be used to cross-reference to resources in the CSTA Source Web repository. Teachers should use their own discretion in deciding how to order topics. In some cases, it may benefit students to tie related topics together and to embed and interweave topics; they do not all have to be treated discretely. A good example is Topic 8: Principles of Software Engineering, where the concept of team-oriented approaches to problem-solving is explored. This methodology, of course, can be included in the presentation of other topics and need not be treated individually. Likewise, individual student learning objectives and focus

points listed under each topic are not meant to be used in a rigid manner. Teachers may see inter-relationships among objectives and focus points that reach across topics. We encourage teachers to arrange the content based on student needs and personal teaching experiences.

The suggested time frame for each topic is provided to help map the curriculum to a year-long course of study. Available resources, students' prior knowledge, and desired depth of coverage will affect the amount of time devoted to each topic.

To the extent possible, no assumptions have been made regarding available hardware and software as this will vary. For certain topics, students will write code, but the particular language and environment have been left open to accommodate variations in available resources. The objectives can be achieved using purchased software, open source software, and in some cases, paper and pencil. Financial issues should not interfere with sound computing education.

Any course of study, any area of inquiry, anything worth investigating should be done in an open-ended and engaging way. Teachers should not be hobbled by a rigidly applied set of standards; it is hoped that the guidelines herein presented will be applied flexibly in ways that promote open-ended investigation and encourage learning about the computing disciplines.

A Note about the CSTA Source (Web Repository)

The goal of the CSTA Source Web repository is to create a Web-based multi-grade collection of appropriate materials (activities, lesson plans and other resources) for K-12 computer science teachers to support their teaching and professional development. The CSTA Source Web repository is the collective effort of computer science teachers everywhere. Teachers are encouraged to submit their own original materials for the benefit of the larger community. Additional information about the CSTA Source Web repository, its content, and submission guidelines is available on the CSTA Web site at http://csta.acm.org.

Topic 1: Program Design and Problem Solving

Topic Description
This topic provides an introduction to the fundamental ideas about problem solving and program development including style, abstraction, and discussion of correctness as part of the software design process. Students design algorithms and programming solutions to a variety of computational problems. While the choice of programming language is left to the instructor, the programming component should include control structures, functions, parameters, objects and classes, and structured programming and event-driven programming techniques.

Textbooks and Supplies:
A programming language, preferably with an interactive development environment.

Time to Complete:
3-5 weeks

Student Learning Objectives	Assessment Measures
The student will be able to:	
1. Given a problem statement, write a natural language procedure for solving the problem.	Written Activity
2. Use design tools (such as flowcharts or UML diagrams) to express a solution to a problem.	Written activity Lab activity
3. Solve a problem by applying a problem-solving process and translate the solution into a program that uses appropriate data types and control structures.	Written activity Lab activity
4. Illustrate the “is-a” and “has-a” object-oriented concepts.	Written activity
5. Test a solution to ensure that it meets its stated requirements.	Lab activity

Assessment Recommendations:	
An average of 60% from combined assessment measures is required to demonstrate proficiency in course material.	
Lab activities	50%
Written activities	50%

Detailed Outline	
Focus	**Sample Lab / Hands-on Activity**
1. Terminology	Identify and define key terms associated with programming.
2. Understanding the problem	Students are given several problems to review. Some have all the detail stated within the text. Others are missing detail. For each complete problem, students restate the problem, including input and output, and what classes will be involved. For each incomplete problem, students request an interview with the problem owner to ascertain the complete problem and reformulate it into a complete statement.
3. Exploring problems: problem-solving heuristics and strategies	Small teams of students discuss a problem and various alternative approaches to its solution. Example: how do you add the numbers from 1-10? From 1-1000? Another example: name two ways to check if a word is a palindrome (spelled the same forwards and backwards). They determine the best solution based upon speed and/or elegance.
4. Design creation and representation	Students design a solution to a given problem using a standard design tool and then present this solution. The diagram includes input, process, and output components as well as class diagrams.
5. Design re-evaluation and refinement	Students are given the solution to a problem that is only partially correct, or correct, but poorly designed. They correct flaws and produce an improved, complete design.
6. “Is- a” and “has-a” relationships	Students are given diagrams graphically illustrating a variety of classes of objects and instances of classes and are asked to identify all ‘is-a’ and ‘has-a’ relationships.
7. The power of stepwise refinement	The instructor presents a magic trick and works with the students to discover details about how it is accomplished. Appropriate notation should be used to clearly show how the problem is broken down into sub-parts and then steps.
8. Programming style	Students are given written code for a program that has no comments, one-letter non-descriptive variable names, multiple statements on a line, etc. They comment the program and make other stylistic changes as necessary.
9. Data types	Given a list of data, students determine the correct data type for each item in the list. The list should include data that has assorted types of text (numeric, character, and date), audio, video, links, and images.
10. Objects and classes	Example 1: Discuss blueprints for houses and their ability to be reused and customized. Have students identify other examples of classes and objects in real life. Example 2: Dogs and Cats inherit from Animal. Poodle and Collie can inherit from Dog. A pet named Rover is an instance of Poodle.

Detailed Outline	
Focus	**Sample Lab / Hands-on Activity**
	Discuss the concepts of inheritance, polymorphism, and encapsulation within the framework of the blueprint/house and dog/pet.
11. Input and Output	Write a program that uses input from the keyboard and from a file and produces output to the monitor and to a file.
12. Expressions	Students are asked to solve problems that require evaluation of relational and logical expressions and demonstrate knowledge of order of operations.
13. Selection	Students write pseudocode for all of the decisions they made when they first got up this morning. (get up or sleep five more minutes, what color socks to wear, what goes into pocket/purse, what to eat for breakfast, etc.). Which decisions are simple? Which are complex? Which involve nested decision-making? Which are best expressed using Select-Case (or switch) structures?
14. Iteration	Use real-life examples that can be pseudocoded as various kinds of loops: For loop (definite repetition): For Day = 1 to 7; While loop (indefinite repetition): While Summer = NotHereYet; Recursion: Function CountSheepToGetToSleep { … CountSheepToGetToSleep … }. Include counters, accumulators, and exit conditions for each kind of loop.
15. Interactive programming	Class discusses the advantages to the user of event-driven programs vs. procedural programs. Students run a command-line-driven program and describe how they interact with the program. They do the same with an event-driven program and then create a chart that highlights the different interaction.
16. Method (functions) and parameters	Write a program using an existing class and its methods. Example: using methods of a Graphics object, display squares of different side lengths. Then have students create their own methods (both with and without parameters) as part of their own class creation/modification, and call their methods.
17. Properties	Students discuss the properties of objects in the real world and then use properties in the creation/modification of their own classes.
18. Array variables and/or other aggregate types	Write a program using an array that accepts a list of names and then displays them in reverse order. Discuss how other structures might be used to handle the same task.

Topic 2: Data Structures

Topic Description:
This unit introduces data structures, including arrays, vectors, stacks, and queues, and their associated components, operations, and uses. Benefits and limitations of different data structures are presented. The concept that analysis and understanding of data structures can be used as a fundamental organizing principle in the design of solutions will be explored.

Textbooks and Supplies:
A programming language and assorted supplies as needed to support your chosen exercises

Time to Complete:
3-4 weeks

Student Learning Objectives	Assessment Measures
The student will be able to:	
1. Write a program to process a range or all elements in one- and two-dimensional arrays.	Lab activity Written activity Project
2. Describe the difference in the processing of arrays, stacks, and queues	Lab activity Written activity Project
3. Select the appropriate data structure – queues, stacks, and/or arrays – to solve a given problem	Written activity

Assessment Recommendations:	
An average of 60% from combined assessment measures is required to demonstrate proficiency in course material.	
Lab activities	50%
Written activities	50%

Detailed Outline	
Focus	**Sample Lab / Hands-on Activity**
1. One-dimensional arrays	Write a program that accepts the names of the presidents in chronological order. Display the list in reverse order. Use the same array to display a select group of presidents (ex: any two presidents with a common first name. Example: John Adams, John Quincy Adams). Display a count of presidents with the same last name. Process all the sound values in a sound file to increase or decrease its volume.
2. Two-dimensional arrays	Write a program to play Tic-Tac-Toe. The program uses a 2-D array to represent every square in a tic-tac-toe board. Use a value of 0 for empty squares, -1 for squares with Xs, and +1 for squares with Os. Process pixels in a picture. Store RGB pixel information in a 2D array and then reduce/increase red. Process a range of data by flipping the image horizontally or vertically.
3. Lists	Write a program that displays a slide show given a data file that contains a list of pictures. The picture data file is maintained by the user so the program must provide the ability for the user to add/delete from the picture list. Show each picture for a set amount of time.
4. Stacks	Draw the call stack for a given recursive problem. Write a program that will calculate the result of any postfix notation algebraic expression using a stack. Write a simulation of a card game and use a stack of cards.
5. Queues	Draw a printer queue and show how the jobs are added and removed from the queue. Write a program that uses a queue to simulate orders in a restaurant. Write a program to decode a message. Use a queue to hold the key. Each time you use an item of the queue add it to the back of the queue till all of the letters in the message have been processed.
6. Data Structure Algorithms	Present (or have students develop) algorithms for: ordering from a fast food drive-through (queue), washing a stack of dishes (stack), and selecting a blue blouse/shirt to wear (array). Which steps in each algorithm uniquely identify a queue, stack, or array process? Which steps are not unique?
7. Selecting appropriate data structures	Provide a description of various simple-to-complex problems. Students select a data structure that is well-suited to the problem and defend their choice.

Topic 3: Discrete Mathematics – Logic, Functions and Sets

Topic Description:
This unit includes selected topics in discrete mathematics including (but not limited to) Boolean logic, functions, sets, and graphs. Students construct complex expressions based on fundamental Boolean operations and learn how to relate the mathematical notion of functions to its counterparts in computer programming. They learn basic set theory and its application in computer science. Students are introduced to graphs using puzzles. Suitable exercises are presented that illustrate the value of mathematical abstraction in solving programming problems.

Textbooks and Supplies:
Internet resources, mathematics books, computer science texts.

Time to Complete:
3-4 weeks

Student Learning Objectives	Assessment Measures
The student will be able to:	
1. Write programs that use simple and complex logic statements (relational operators and Boolean operators).	Written activity Lab activity
2. Analyze truth tables and recognize logical equivalencies.	Written activity
3. Understand set terminology and describe data structures that can house these sets.	Written activity Lab activity
4. Write an algorithm that uses mathematical functions.	Written activity Lab activity
5. Apply simple graph concepts in problem solving.	Written activity Lab activity

Assessment Recommendations:	
An average of 60% from combined assessment measures is required to demonstrate proficiency in course material.	
Written activities	50%
Lab activities	50%

Detailed Outline	
Focus	**Sample Lab / Hands-on Activity**
1. Logic	Students are given formal propositions (simple and complex) and complete truth tables. Some statements should be negations of propositions. Encourage students to break more complex propositions into smaller components to aid in the solution. Both mathematical symbols (Λ , V and ~) and typical computer language symbols (&& , || and !) should be used. Students should translate the formal proposition into set notation and a set of programming statements to test and verify the logic.
2. Basic Sets	Discuss set terminology and fundamentals: elements, union, intersection, set size, and null sets. Engage students in activities that focus on conceptual sets as well as sets of items familiar to the students (colored blocks, shapes, or snack items). Examples: Students draw Venn diagrams showing RGB primary colors singly and in any combination, or to help solve problems such as: All my friends go to my school. Rita is my friend. All the kids on my block have graduated. Does Rita live on my block? Students describe data structures that can be used to handle sets.
3. Concepts of Functions	Students explore a variety of functions in spreadsheet software: (sum, average, count, minimum, maximum, as well as some string-handling functions). Ask students to write theoretical program statements to call these functions and report the result. Have students explain the algorithm for one or more functions and discuss the advantages of coding with functions.
4. De Morgan's Laws	Students practice re-writing logical equivalencies by applying De Morgan's Laws. The equivalencies should be written in formal logic notation, set notation, and in program code to test and verify the logic.
5. Graphs	Students are presented with a connect-the-dots problem such as The Bridges of Konigsberg. The objective is to connect the dots in the most efficient manner possible (ex: no repeat traversals or minimum number of points visited). Solutions are discussed and validity is ensured. Students generate algorithms and discuss which are most efficient. Advanced students should be encouraged to implement the solution in program code.

Topic 4: Design for Usability

Topic Description:
This section prepares students to take the role of a developer by expanding their knowledge of programming and Web page design and applying it to the creation of Web pages, programs, and documentation for users and equipment. Students learn to create user-friendly manuals, Web sites, and program interfaces. Fundamental notions of Human Computer Interaction (HCI) and ergonomics are introduced. Code documentation and hardware and software limitations are also explored.

Textbooks and Supplies:
Textbooks, hardware, and software appropriate for Web page design and programming.
Optional: resource-limited hardware, such as PDAs.

Time to Complete:
2-3 weeks with appropriate reinforcement in other topics

Student Learning Objectives	Assessment Measures
The student will be able to:	
1. Create user-friendly and functional Web sites and programs that apply good HCI practices.	Written activity Lab activity
2. Create Web sites and programs that recognize hardware and software constraints of potential client machines and/or environments.	Written activity Lab activity
3. Prepare documentation that follows professional standards.	Written activity
4. Apply good code documentation techniques to Web sites and programs.	Written activity Lab activity

Assessment Recommendations:	
Assessment Recommendations: An average of 60% from combined assessment measures is required to demonstrate proficiency in course material.	
Written activities	50%
Lab activities	50%

Detailed Outline	
Focus	**Sample Lab / Hands-on Activity**
1. Fundamental HCI concepts	Working in small groups, students research HCI on the Internet. They prepare a short presentation on some aspect of HCI: definition/terminology, emerging technologies, fields affected by HCI, history of HCI, software/hardware aspects of HCI, etc.
2. Identify elements of user-friendly Web sites	Students work in small groups to find one poorly-designed informational Web site and one well-designed informational Web site. If possible, the sites are to be viewed in multiple current browsers and on various hardware platforms. Students prepare a short presentation explaining their criteria. Students visit W3C and other Web sites to research issues related to Web site design and accessibility. Students provide suggestions to improve the sites.
3. Design a user-friendly Web site	Students work in small groups to develop a storyboard and navigation plan for a four-page informational Web site on a topic of their choice (school, community, softball team, common cause, etc.).
4. Create a user-friendly Web site	Students work in small groups to transform a storyboard and navigation plan into a finished Web site. Students learn about various coding standards for Web pages. They embed comments in the Web pages.
5. Identify elements of user-friendly software	Working in small groups, students investigate several media player programs and rate them in terms of usability (how easy they are to learn, how easily they access commonly-used features, how easy it is to get help for features, size/location of buttons/commands, etc.).
6. Design a user-interface for a program	Students work in small groups to storyboard and design the user interface and the user experience for a hypothetical game program. What will the user first see? What subsequent screens and interactions will occur? What will happen as the program terminates?
7. Documentation techniques	Students are given a short program that does not follow professional coding standards (poor variable names, no documentation), but which will compile and run. They modify the program so that it meets professional standards (rename variables and objects, provide comments where appropriate, add a preamble with author, title, description, version, and environment information) and provide user documentation.

Topic 5: Fundamentals of Hardware Design

Topic Description:
This unit introduces logic gates as a hardware implementation of Boolean logic and binary arithmetic. Students explore how basic logic gates can be combined to build components and systems of any complexity. Basic machine architecture (processor registers, the control unit, ALU, and memory) are explored. Students learn how data is encoded, stored, manipulated, and moved in this context.

Textbooks and Supplies:
Internet and assorted textbooks

Time to Complete:
3-4 weeks

Student Learning Objectives	Assessment Measures
The student will be able to:	
1. Demonstrate knowledge of binary and hexadecimal (hex) number systems, addition and subtraction in binary, and an understanding of two's complement representation.	Written activity
2. Represent Boolean logic in table form and circuit diagrams.	Written activity Lab activity
3. List the four stages of the machine cycle and describe each stage.	Written activity Lab activity
4. State the components of the CPU and describe how they operate.	Written activity
5. Match a list of various types of memory with the primary purpose of each type.	Written activity Lab activity
6. List ways to increase computer performance.	Written activity Lab activity

Assessment Recommendations:	
An average of 60% from combined assessment measures is required to demonstrate proficiency in course material.	
Written activities	50%
Lab activities	50%

Detailed Outline	
Focus	**Sample Lab / Hands-on Activity**
1. Review: Conversion among decimal, binary and hex number systems	In pairs, students practice encoding short ASCII messages into binary or hex. Each student decodes another student's message. Given a set of addition/subtraction problems written in binary and hex, students translate them into decimal, solve them and translate the answers back into binary and hex.
2. Binary counting and switching	Students write a 1 on the face-up side of several index cards and a 0 on the face-down side of each card. Place 8 cards in a line, some facing up and some down. Determine how many possible ways the cards can be manipulated (unique combinations of 1s and 0s). (Each change from 1 to 0 is a change in state of the switch.) Challenge students to count to more than 5 by 1s with the fingers of one hand. Each finger can only be up or down.
3. Encoded data and integrated circuits	Introduce character sets and have students arrange the cards to represent specific characters. Discuss the limitation of 8 bits and the need for more characters in an international society. Add 8 more index cards to the set. How many unique combinations can be made now? Point out the spatial problem with the cards as more are added. Help students suggest that each card needs to be smaller so you can pack more together. Introduce integrated circuits. Ask students to arrange eight index cards in a binary switch pattern corresponding to "A". Ask students to arrange the remaining eight index cards in a binary switch pattern corresponding to the integer number 65. Point out that the on/off patterns of the 8 switches are the same and that the decision as to "A" vs. 65 will be determined by software.
4. Representation of numbers	Students describe the state of the switches for a given negative number and its absolute value. Introduce the concept of a sign bit. Discuss two's complement. Given a list of decimal numbers, students write the positive binary value and the negative representation.
5. Adding and subtracting binary numbers	Students add binary numbers of any length and subtract binary numbers using two's complement representation.
6. Drawing logic gates and circuit diagrams	Use a circuit diagram to introduce the graphical representation of the logical *and*, *or*, *not, nor, nand,* and *xor*. Given a second diagram, students identify each type of gate. Then they trace the flow of data through a simple combinational circuit diagram.

Detailed Outline	
Focus	**Sample Lab / Hands-on Activity**
7. Understanding the major component parts of the microprocessor.	Draw a representation of the parts of the computer, including RAM, ALU, CU, cache, and busses. Identify how the parts interact and how data travel from one part to another. Compare and contrast CPU specifications in two computer ads.
8. The machine cycle	Each student in a group of four is assigned a role of Fetch, Decode, Execute, or Store. Provide a short list of coded statements on cards. The appropriate student fetches a card and hands it to Decoder (who has a translation table). Decoder rewrites the translation in everyday words and hands the card to Execute. Execute performs the action and gives Store the result. Store writes the result and places the card in the "done" stack. Then Fetch can begin with the next card. A variation of the exercise above can be used to illustrate pipelining and dual-core architecture. Ask students to research Von Neumann architecture. The history of computer hardware can be added to this topic as time allows.
9. Processing speed	Students obtain the technical specifications for two modern computers and provide an analysis as to which computer will operate the fastest. The analysis should also address how both computers could be upgraded and the relative speed vs. expense to modify the hardware.

Topic 6: Levels of Language, Software, and Translation

Topic Description:
This unit examines the relationships among high-level languages, assembly language and machine language, as well as the role that compilers and byte-code generators play in code development. Students explore the significance of operating systems and networking in the development process.

Textbooks and Supplies:
Internet, assorted textbooks, computer lab with appropriate software

Time to Complete:
2 weeks

Student Learning Objectives	Assessment Measures
The student will be able to:	
1. State the advantages and disadvantages of high-level and assembly language.	Written activity
2. Describe the process by which high-level source code is compiled into machine language in broad terms,	Written activity
3. State the characteristics that distinguish compilers, byte-code generators, and interpreters.	Written activity Lab activity
4. Describe the impact of operating systems on the development process.	Written activity Lab activity

Assessment Recommendations:	
An average of 60% from combined assessment measures is required to demonstrate proficiency in course material.	
Written activities	50%
Lab activities	50%

Detailed Outline	
Focus	**Sample Lab / Hands-on Activity**
1. Translation between levels of software	Given a variety of code snippets in both a high-level language and assembly language, students match the high-level snippet with its assembly language counterpart. Given a translation table for an assembly language code sample, students hand-compile machine language code.
2. Fundamental functions of high-level languages	Given source code in a high-level language and a set of equivalent structures and keywords in another high-level language, students translate from one to the other. When available, the student should enter and test the code in the second language until it successfully compiles and runs.
3. Compilers and byte-code generators.	If access to multiple platforms (Windows and Linux, for example) is available, have students observe the portability of byte code vs. compiled code. Give students source code. They will compile and execute the source code on one platform and then move it to another platform. What happens when they attempt to run the compiled code? Have students generate byte code and run on one platform and then move this code to the second platform. What happens when it runs on the second platform? Help students understand the differences between a compiler and a byte-code generator and why byte-code generators are important in networked environments such as the Internet.
4. Operating System (OS)	Students discuss OS features which allow software to access hardware, i.e. statements that generate system calls. Students outline the steps they envision occurring as a statement is executed. Example: an input statement from disk requires OS intervention to access the File Allocation Table (FAT) and locate the required content on disk.
5. Network environments	Students explore the operation of scripts in a networked environment. Students write short client-side scripts for Web pages such as a script to request a user's first name and birth date. The script then calculates how many days, weeks and months old the person is. Students visit the Web page using various available platforms. When Internet access and an appropriate language are available, students implement Web services, which are freely available online. Example: a client-side script could request the person's first name and zip code. The script contacts a Web service that returns a town name.

Topic 7: Limits of Computing

Topic Description:
This unit provides an elementary introduction to computational complexity theory to encourage an appreciation for the relative efficiency of various algorithms. Students are introduced to examples of computationally "hard" problems, computationally unsolvable problems, and problems that are made difficult by the complexity of the realities they attempt to model (air traffic control, human intelligence, weather).

Textbooks and Supplies:
Internet and assorted textbooks

Time to Complete:
2-3 weeks

Student Learning Objectives	Assessment Measures
The student will be able to:	
1. List activities in which humans excel over computers and activities in which computers excel over humans.	Lab activity Written activity
2. Calculate the number of steps required to execute a given algorithm.	Lab activity Written activity
3. Define parallel processing. Describe how to solve a problem using parallel processing.	Lab activity Written activity
4. Describe and run computationally intensive problems.	Written activity Lab activity
5. Describe at least one problem computers cannot solve.	Written activity
6. Describe at least one computationally hard (NP) problem.	Written activity

Assessment Recommendations:	
An average of 60% from combined assessment measures is required to demonstrate proficiency in course material.	
Written activities	50%
Lab activities	50%

Detailed Outline	
Focus	**Sample Lab / Hands-on Activity**
1. Computers vs. humans	Students sort a list of names alphabetically by hand and using a computer program. Compare the time it takes to accomplish the task. A student records a short speech and then types the speech. Use speech recognition software to convert the same speech to text. Compare the results for speed and accuracy. Given a set of tasks, students predict which would do better at the task – a human or a computer?
2. Algorithm efficiency	Students count 'by hand' how many times a loop will execute for some given data. Then they use an incremental counter and print its value after the loop finishes. Is there a way to reduce the number of steps? Repeat with nested loops, indeterminate loops, recursive structures, selection structures, and sequential structures.
3. Computationally intensive problems	Discuss the definition of a prime number (a number not evenly divisible by any other numbers except itself and 1). Write an algorithm based on this definition to decide if any given number is prime. Extend the algorithm to test a range of numbers. Optimize the algorithm: divide each number by every number up to its square root. (Why is this sufficient?) Count steps when the range of numbers to be tested is 1,000,000 to 2,000,000. How many steps are saved with optimization? Are there further optimizations that should be considered?
4. Parallel processing	Students add up all the numbers from 1 to 100. Students are then divided into 10 teams. Each team is assigned 10 numbers to add, thus distributing the task of adding 100 numbers among the 10 teams. (One team will add up 1-10, the next 11-20, and so on.) One student adds up all the preliminary results. Visit sites that discuss massively parallel computing. If possible, set up a computer lab at school to participate in a massively parallel computing project. Present the idea that there may be alternate algorithms that will solve the problem more efficiently.
5. Unsolvable problem for the computer	Describe the halting problem. Ask students if they think the halting problem can be solved by a computer. Then read about the halting problem and the significance of it at a Web site of your choice.
6. Computationally hard problems	Consider the issues raised by the Traveling Salesman problem. What is the optimal path with 5 cities to visit? Why? What is the optimal path for 6 cities or 10 cities?

Topic 8: Principles of Software Engineering

Topic Description:
Students are introduced to software engineering concepts and team-oriented approaches for solving problems. They learn the essential methods of the software development life cycle and use these methods in one or more group projects.

Textbooks and Supplies:
A personal computer, and a software design tool (if available)

Time to Complete:
2 weeks with appropriate reinforcement in other topics

Student Learning Objectives	Assessment Measures
The student will be able to:	
1. Name the different phases of the software development process.	Written activity
2. Use a software process model (such as the waterfall, RAD, incremental, or XP) to solve a problem.	Lab Activity
3. Complete a project as a software design team with assigned roles and responsibilities for each member.	Lab activity
4. Complete programs using pair programming.	Lab activity

Assessment Recommendations:	
An average of 60% from combined assessment measures is required to demonstrate proficiency in course material.	
Team project	50%
Written activities	50%

Detailed Outline	
Focus	**Sample Lab / Hands-on Activity**
1. Software design team	Students research the various roles and responsibilities of members of a software design team. The students are then divided into small project teams. Each team member is assigned a role and lists his/her responsibilities for the project.
2. Break a problem statement into specific requirements	Students are divided into teams and each team is asked to develop a list of requirements for an ice cream stand that sells cones, cups, and floats. Items can have a choice of toppings and can have either one or two scoops of ice cream. The problem and requirements list are given to another team for evaluation and then updated based on feedback. The scenario can be made simpler or more elaborate as needed.
3. Design a solution to a problem	Given requirements, student teams develop a problem design. Design tools can be used if available. We recommend an object-oriented programming approach in which students design classes and specify the methods and properties for each class.
4. Code a solution from a design	Student teams implement a project design.
5. Test a solution to identify bugs	Student teams develop a unit test plan or a validation/verification test plan to search for bugs in their project solution. Their completed project is tested by another team and then updated as needed.
6. Pair programming	Encourage pair programming by assigning students to work together on one or more programs. (Provide specific guidelines: frequency of switching roles, logging of work, etc.)
7. Team Oral Presentations	Student teams present their projects. The presentation should include possible future enhancements.

Topic 9: Social Issues

Topic Description:
Students study the responsibilities of software users and software developers with respect to intellectual property rights, software failures, and the piracy of software and other digital media. They are introduced to the concept of open-source software development and explore its implications.

Textbooks and Supplies:
PC with an internet connection, the ACM code of ethics available at www.acm.org

Time to Complete:
1-2 weeks

Student Learning Objectives	Assessment Measures
The student will be able to:	
1. Define intellectual property, explain the rights of owners and end users, and provide rationale for the need to protect owners and end users.	Classroom discussion Written activity
2. Define software piracy and discuss its effect on software company profits and the price of software to the consumer.	Lab activity
3. List at least two ways in which software (and other digital media) is protected and state at least one current law to protect software and the makers of software.	Lab activity
4. Describe the responsibilities of software professionals to society and to each other.	Lab activity
5. List the advantages and disadvantages of open-source software.	Lab activity

Assessment Recommendations:	
An average of 60% from combined assessment measures is required to demonstrate proficiency in course material.	
Written activities	50%
Labs	50%

Detailed Outline	
Focus	**Sample Lab / Hands-on Activity**
1. Terminology	Use the Internet or printed literature to find definitions for the following terms: intellectual property, software license, copyright, patent, software piracy, digital watermark, and fair use.
2. Intellectual property rights	Students compute how much money can be lost by a software company due to software piracy. Provide an example scenario where a company charges $100 for a software package and sells 1000 units. Each unit is copied illegally by ten additional users without payment. Find three estimates on the Web regarding the percentage of computer software that is copied illegally. How were these estimates derived?
3. Software licensing agreements	Students read through a software license agreement from a popular software program (e.g. Windows) and describe each clause of the license in their own words. Students poll five friends or family members who use computer software and ask who has read the license agreement. Students should ask their friend/family members to describe their rights if the software fails. Can they put the software on more than one PC? If yes, how many?
4. Digital rights management (DRM)	Students describe two methods that companies use to protect digital media (software, music, video, art). They examine a software package, DVD, or music CD to see if contains some form of DRM software.
5. Intellectual property/fair use conflicts	Teams of students debate the positions of the Recording Industry Association of America (RIAA) and consumers on the measures to protect copyright on musical recordings. Students research arguments on both sides of the issue.
6. Current legislation and/or litigation	Given a list of several current laws and recent or pending legal cases that deal with intellectual property issues, each student selects one scenario and uses the Internet to find a concise explanation of the issues involved.
7. Responsibilities of a software professional	Students read through the Association of Computing Machinery (ACM) code of ethics. Divide students into eight teams. Each team prepares a skit that summarizes one of the eight major responsibilities of a software professional. The skits are presented to the class.
8. Software failures and responsibilities	Students research a major software failure and prepare a short oral presentation to the class describing the actual failure, how much it cost in lost dollars and/or lives, and why the failure occurred. Students also comment on whether such a failure is likely to happen again and whether software engineers should be held responsible if the failure occurs repeatedly.

Topic 10: Careers in Computing

Topic Description:
Students identify and describe careers in computing and careers that employ computing. Information is provided about the required technical skill set, soft skills, educational pathways, and ongoing training required for computing careers. Students also explore how computers are used in other career choices. Finally, students are made aware of which additional secondary-level courses might be needed in preparation for particular careers.

Textbooks and Supplies:
Newspapers, Parents, Internet, Guest Speaker(s)

Time to Complete:
1-2 weeks

Student Learning Objectives	Assessment Measures
The student will be able to:	
1. List five careers related to computers.	Written activity
2. List three or more skills needed to succeed in at least three computer-related careers.	Written activity
3. State the level of education and ongoing training needed for at least three careers.	Written activity

Assessment Recommendations:	
An average of 60% from combined assessment measures is required to demonstrate proficiency in course material.	
Written activities	100%

Detailed Outline	
Focus	**Sample Lab / Hands-on Activity**
1. List careers related to computers	Students compile a list of careers through brainstorming and/or group activities. Each item on the list should include a brief description of the position. The list should minimally include computer scientist, computer engineer, software engineer, information technologist, game developer, and database manager.
2. Personal career choices	Students research a possible personal career choice. The research should include a brief job description, a list of minimal educational qualifications, and a description of likely computer skills needed to perform on the job successfully.
3. Skills and academic background	Students research the skills needed and the academic background for three computer positions. They complete a table with the careers in the left column and the skills, academics, and ongoing training along the top. Students include a list of resources used (URLs, job postings, interviews, college catalogs, etc.) and indicate the entry-level position for each career.
4. Choose a computing career	Students select an entry-level computer position (advertised in print or online) in which they might like to begin their career. Each student creates a resume that includes the academic background and professional skills s/he expects to have upon graduation from high school or college. Conduct mock interviews for the indicated position. Ask a local computer company to assist with the interviews. Invite guest speakers to address the class. They should describe what they do, what the challenges are, and what specific education, training, and experience are needed, what it takes to get the job, what it takes to hold onto the job, what non-technical (soft) skills are needed, and what it takes to move ahead.

Acknowledgements

Feedback[‡] towards improvement of this document was provided by:

- Attendees at SIGCSE 2007 "Developing Resources to support a National Computer Science Curriculum for K-12"
- Daniel Frost, University of California, Irvine
- Philip East, University of Northern Iowa, Cedar Falls
- Michelle Hutton, The Girls' Middle School, Mountain View, CA
- Judith Gal-Ezer, The Open University of Israel

[‡] Feedback does not necessarily indicate endorsement.

Appendix

Level III Curriculum: Facets[§] of Computer Science

Computer science can be seen as comprising four facets:

- Computer Hardware
- Using Computer Software
- Solving Problems by Developing Software
- Computers, People, and Society

Each of these facets has principal concepts and underlying themes, which are described below.

Computer Hardware

The development of the electronic computer has been one of the technological marvels of the last century. Research and development of computers and peripherals actively continues.

Principal concepts and themes of this facet are:

1. At a fundamental level, all computers are collections of circuits.
2. The most common architecture for computers is based on a central processor, memory, and peripherals.
3. Memory storage devices (including punch cards, paper tape, cassette tapes, hard disk drives, floppy disks, CD-ROMs and DVDs, memory sticks, RAM, ROM, cache, and video memory) have a variety of characteristics.
4. Memory storage devices usually store information in units called bytes, and each byte has a numeric address.
5. Computer processors (chips) are almost ubiquitous in cars, cell phones, traffic signal controllers, and other embedded devices.
6. Peripherals serve two main functions, input and output. Specialized peripherals are used in non-PC devices such as robots, satellites, cell phones, and digital cameras.
7. Computers are connected in networks via a wide variety of communication media, including telephones (with modems), Ethernet cable, and wireless.
8. Various tools (screwdrivers, Allen wrenches) are used to open computer cases and add or remove components.
9. Safety concerns must be kept in mind when assembling or fixing computers or any electronic equipment.

Using Computer Software

Some software is written to be tightly integrated with specific hardware, as in a cell phone or a digital camera. In other cases, a software application, such as a word processor or a Web browser, is primarily designed to run on standard hardware. Sometimes software works "behind the scenes" and can be almost

[§] These categories were defined by Daniel Frost, Donald Bren School of Information and Computer Sciences, University of California, Irvine and initially used in A Model Curriculum for K-12 Computer Science: Level II Objectives and Outlines.

invisible, as in the Internet or many parts of an operating system. Familiarity with a variety of computer software programs and with the basic concepts underlying many of them is a prerequisite for many jobs and for understanding a large part of 21st century culture.

Principal concepts of this facet are:
1. Software may be tightly or loosely coupled with the computer hardware on which it runs.
2. By representing information in digital format, computers can store, manipulate, and transmit that information.
3. The instructions a computer follows are in software, which means that they can be changed easily.
4. Related data is often stored in a "file." A file (usually) corresponds to a particular range of locations in a memory storage device. The data in the file often has a specific format. A file has a name, and often the name has an extension that is associated with a specific program that knows how to use the contents of the file.
5. Files are organized in a hierarchy of folders or directories.
6. Files should be backed up periodically.
7. The same data can often be displayed in multiple ways.
8. Personal computers' operating systems often have graphical user interfaces.
9. Commercial "shrink-wrapped" software includes categories such as word processors, Web browsers, presentation and slide managers, spreadsheets, databases, graphic and music content creators and editors, and e-mail clients.

Solving Problems by Developing Software

Computer software solutions are created by identifying a need or opportunity, analyzing how it can be addressed with software, designing and coding the program, carefully testing the program, and in many cases writing documentation and training the users. Gaining a basic understanding of how software is created gives students a deeper understanding of what computers can do.

Principal concepts of this facet are:
1. Creating software involves several common phases:
   (a) Identifying the requirements from the user's perspective
   (b) Planning how to write the program (particularly important when the program is large or more than one programmer will collaborate)
   (c) Following the plan by writing code in a computer programming language
   (d) Testing the program to assure it meets the original requirements, runs acceptably quickly, is user-friendly, and has other desired qualities
   (e) Turning the program over to the user, which involves training, documentation, and designing procedures

   Often, steps (d) and (e) are repeated after beta-testing by end users.
2. Computers follow programs, which are written by humans.
3. Computer programs are written in computer languages, which have rigid syntax utilizing a limited number of key words and symbols.
4. Computers have no "understanding" beyond what is explicitly coded into a computer program.
5. For a problem to be solved by a computer, every step must be defined in detail.
6. Information used by the computer must be represented as digital data.

7. Writing a computer program involves selecting or creating algorithms and data structures, and analyzing their performance and other characteristics. Algorithms to solve a specific problem vary widely, and often involve different trade-offs of space and time.
8. Algorithms implemented in computer programs are made up of elementary control structures, such as conditionals, loops, and subroutines.
9. Computer science has been developed in the 20th and 21st centuries, but the philosophic roots of the "laws of thought" and algorithmic thinking originated with Plato and the pre-Socratics.

Computers, People, and Society

Technical advances have driven social changes throughout history, and tools have shaped culture in many ways. The rapid development of computers, networks, and peripherals has an ongoing impact on society.

Principal concepts of this facet are:
1. Computer technology and software changes more quickly than ethics and laws, thus creating a constant tension in society.
2. The ubiquity of data in digital format presents new issues of privacy and security.
3. Computerized data is often copied and rarely deleted, raising issues of privacy, ethics, and ownership rights.
4. Humans are best at recognition, making connections between similar things, and learning by doing. Computers are best at following small instruction steps and processing digital data quickly and consistently.
5. A human-computer interface is the meeting point of the human and computer realms. A good interface minimizes the human's short-term memory load, is compatible with a diverse set of users, and prevents errors[**].
6. Computers are tools with several functions: to process data (to compute), to store data, to acquire and display data, and to move data from one computer to another (to communicate).

Beginning on the next page, we've included a cross-reference between these facets of computer science and the Level III curriculum's topics and focuses.

[**] Shneiderman, Ben. *Designing the User Interface.* Third Ed. Addison Wesley, 1998.

Topic	Focus	Computer Hardware	Using Software	Developing Solutions	People & Society
Topic 1: Program Design and Problem Solving	1. Terminology			X	
	2. Understanding the problem			X	
	3. Exploring problems: problem-solving heuristics and strategies			X	
	4. Design creation and representation			X	X
	5. Design re-evaluation and refinement			X	
	6. “Is- a” and “has-a” relationships			X	
	7. The power of stepwise refinement			X	
	8. Programming style			X	
	9. Data types			X	
	10. Objects and classes			X	
	11. Input and Output	X	X	X	
	12. Expressions			X	
	13. Selection			X	
	14. Iteration			X	
	15. Interactive programming			X	X
	16. Method (functions) and parameters			X	
	17. Properties			X	
	18. Array variables and/or other aggregate types			X	
Topic 2: Data Structures	1. One-dimensional arrays			X	
	2. Two-dimensional arrays			X	
	3. Lists			X	
	4. Stacks			X	
	5. Queues			X	
	6. Data Structure Algorithms			X	
	7. Selecting appropriate data structures			X	
Topic 3: Discrete Mathematics – Logic, Functions and Sets	1. Logic			X	
	2. Basic Sets			X	
	3. Concepts of Functions			X	
	4. De Morgan's Laws			X	
	5. Graphs			X	
Topic 4: Design for Usability	1. Fundamental HCI concepts		X		X
	2. Identify elements of user-friendly Web sites		X		X
	3. Design a user-friendly Web site			X	X
	4. Create a user-friendly Web site			X	X
	5. Identify elements of user-friendly software		X		X
	6. Design a user-interface for a program			X	X
	7. Documentation techniques			X	X
Topic 5: Fundamentals of Hardware Design	1. Review: Conversion among decimal, binary and hex number systems	X	X		
	2. Binary counting and switching	X	X		
	3. Encoded data and integrated circuits	X	X		
	4. Representation of numbers	X	X	X	

	5. Adding and subtracting binary numbers		X		
	6. Drawing logic gates and circuit diagrams	X	X		
	7. Understanding the major component parts of the microprocessor.	X			
	8. The machine cycle	X			
	9. Processing speed	X		X	
Topic 6: Levels of Language, Software, and Translation	1. Translation between levels of software		X	X	
	2. Fundamental functions of high-level languages		X	X	
	3. Compilers and byte-code generators.		X		
	4. Operating System		X		
	5. Network environments	X	X	X	
Topic 7: Limits of Computing	1. Computers vs. humans		X	X	
	2. Algorithm efficiency			X	
	3. Computationally intensive problems			X	
	4. Parallel processing			X	
	5. Unsolvable problem for the computer			X	
	6. Computationally hard problems.			X	
Topic 8: Principles of Software Engineering	1. Software design team			X	
	2. Break a problem statement into specific requirements			X	
	3. Design a solution to a problem			X	
	4. Code a solution from a design			X	
	5. Test a solution to identify bugs			X	
	6. Pair programming			X	
	7. Team Oral Presentations			X	
Topic 9: Social Issues	1. Terminology				X
	2. Intellectual property rights				X
	3. Software licensing agreements		X		X
	4. Digital rights management (DRM)		X		X
	5. Intellectual property/fair use conflicts		X		X
	6. Current legislation and/or litigation				X
	7. Responsibilities of a software professional				X
	8. Software failures and responsibilities				X
Topic 10: Careers in Computing	1. List careers related to computers				X
	2. Personal career choices				X
	3. Skills and academic background				X
	4. Choose a computing career				X