AP® Computer Science AB

Syllabus 1

Course Resources


Video: *Sorting Out Sorting*, Ronald Baecker, University of Toronto, 1981

AP GridWorld Case Study

Institute for Mathematics and Computer Science Online Courses (eIMACS)

www.eimacs.com

Supplemental Handouts

- **The Espresso Maker**, Brewing Coffee, Brewing Coffee Interactively (taken from ACSE Symposium, University of Toronto, February 2002)
  - Focus: Review design of objects and classes, creation of classes, and using simple interacting classes.

- **Evaluation of Expressions**
  - Focus: Learn how to evaluate arithmetic expressions using infix, prefix and postfix notation.

- **The Stair Counting Problem** (modified from Main and Savitch, *Data Structures and Other Objects Using C++*, 1997)
  - Focus: Describe three different ways of solving a problem and compare the efficiency of each method.
  - Introduces Big-O Notation, time analysis (linear, logarithmic, quadratic, etc.) and case analysis (worst, best and average).

  - Focus: Describe sorting algorithms (bubble sort, insertion sort, selection sort, shell sort, quick sort, heap sort, merge sort) and searching algorithms (sequential search, binary search).

- **Trees Theory**
  - Focus: Binary tree traversals (preorder, inorder, postorder), heaps (min and max), binary search trees and binary expression trees
  - Includes a list of common tree terminology
Units of Study

Unit 1: Basic Programming Structures

Students learn about the components of a computer system, networks and the Internet. We explore basic programming structures. Topics include conditionals (if, if-else), repetition (for, while, do while), design and creation of objects, arrays and ArrayLists.

Resources:
- Chapters One, Two, Three, Four, Five and Six – Lewis, Loftus, Cocking
- Chapters One and Two – Lewis, Chase, Sudol
- Supplemental Handouts
  - The Espresso Maker, Brewing Coffee, Brewing Coffee Interactively

Unit 2: Efficiency of Algorithms

Students learn how to evaluate the efficiency of algorithms. They evaluate expressions using prefix, infix and postfix notation. Students will use Big-Oh to classify and compare algorithms. They will also implement recursion and various sorting (insertion sort, selection sort, merge sort, quick sort, heap sort, bubble sort, shaker sort and shell sort) and searching (linear and binary) algorithms.

Sample Student Activities for Unit 2: Evaluation of Expressions, Big-Oh Assignment, Sorting and Searching

Resources:
- Chapters Six, Seven and Eight – Lewis, Loftus, Cocking
- Chapters One, Six and Seven – Lewis, Chase, Sudol
- Sorting Out Sorting
- Supplemental Handouts
  - Evaluation of Expressions
  - The Stair Counting Problem
  - Sorting and Searching Algorithms
Unit 3: Abstract Data Types

Students work with advanced data structures. Topics include trees, heaps, stacks, queues, sets, maps, hashing and linked lists.

Resources:
- Chapters Nine and Ten – Lewis, Loftus, Cocking
- Chapters Three, Four, Five, Eight, Nine, Ten, Eleven and Twelve – Lewis, Chase, Sudol
- Supplemental Handouts
  - Trees Theory

Unit 4: AP GridWorld Case Study

Students work on Parts 1, 2, 3 and 4 of the AP GridWorld Case Study and study Part 5.

Resources:
- AP GridWorld Case Study

Unit 5: The Computer and Society

Students explore how the computer has affected society through various assignments. They look at case studies and a variety of issues pertaining to environmental concerns, hardware and software, careers, computer crimes and miscellaneous items. Students will investigate and analyze careers created by the introduction of computers to society and examine the impact of rapid technology changes on local businesses. They will also have opportunities to reflect on whether they would choose a computer science related career and create a code of computing ethics.

Students will submit a series of written assignments and create a website.

Resources:
- Assignment handouts
  - Impact of Computers: Case Studies—written assignment
  - Impact of Computers: Issues—written assignment
  - Computer Careers—Web site
  - Facing Change—written assignment

C3—The course teaches students to design and implement computer-based solutions to problems in a variety of application areas.

C4—The course teaches students to use and implement commonly used algorithms and data structures.

C5—The course teaches students to develop and select appropriate algorithms and data structures to solve problems.

C6—The course teaches students to code fluently in an object-oriented paradigm using the programming language Java. The course teaches students to use standard Java library classes from the AP Java subset delineated in Appendices A and B of the AP Computer Science Course Description. (Note: Students who study a language other than Java in AP Computer Science must also be taught to use Java, as specified in the AP Java subset.)

C7—The course teaches students to read and understand a large program consisting of several classes and interacting objects, and enables students to read and understand the current AP Computer Science Case Study posted on AP Central®.

C8—The course teaches students to identify the major hardware and software components of a computer system, their relationship to one another, and the roles of these components within the system.

C9—The course teaches students to recognize the ethical and social implications of computer use.
Unit 6: Preparing for the AP Exam

Students work through sample multiple choice questions and free response questions. They will write a mock exam to simulate the timing and type of questions to expect on the actual exam. Students review Java theory and the AP GridWorld Case Study using eIMACS.

Resources:
- AP GridWorld Case Study
- eIMACS
- Supplemental handouts – multiple choice and free response questions

Correlation to AP Topic Outline – Computer Science AB

I. Object-Oriented Program Design
The overall goal for designing a piece of software (a computer program) is to correctly solve the given problem. At the same time, this goal should encompass specifying and designing a program that is understandable, can be adapted to changing circumstances, and has the potential to be reused in whole or in part. The design process needs to be based on a thorough understanding of the problem to be solved.

A. Program design
1. Specify the purpose and goals for a problem. Unit #1
2. Apply data abstraction and encapsulation. Unit #1
3. Decompose a problem into classes; define responsibilities of those classes. Unit #1
4. Understand and implement a given class hierarchy. Unit #1
5. Identify reusable components from existing code using classes and class libraries. Unit #1

B. Class design
1. Design and implement a set of interacting classes. Unit #1
2. Design an interface. Unit #1
3. Choose appropriate advanced data structures and algorithms. Unit #1
4. Apply functional decomposition. Unit #1
5. Extend a given class using inheritance. Unit #1 and 2

II. Program Implementation
The overall goals of program implementation parallel those of program design. Classes that fill common needs should be built so that they can be reused easily in other programs. Object-oriented design is an important part of program implementation.

A. Implementation techniques
1. Methodology
   a. Object-oriented development Unit #1
   b. Top-down development Unit #1
   c. Encapsulation and information hiding Unit #1
   d. Procedural abstraction Unit #1

B. Programming constructs
1. Primitive types vs. objects Unit #1
2. Declaration
   a. Constant declarations Unit #1
### III. Program Analysis

The analysis of programs includes examining and testing programs to determine whether they correctly meet their specifications. It also includes the analysis of programs or algorithms in order to understand their time and space requirements when applied to different data sets.

#### A. Testing

1. Test classes and libraries in isolation.  
   Unit #1
2. Identify boundary cases and generate appropriate test data.  
   Unit #1
3. Perform integration testing.  
   Unit #1

#### B. Debugging

1. Categorize errors: compile-time, run-time, logic.  
   Unit #1 and 2
2. Identify and correct errors.  
   Unit #1 and 2
3. Employ techniques such as using a debugger, adding extra output statements, or hand-tracing code.  
   Unit #1 and 2

#### C. Understand and modify existing code

Unit #1, 2 and 3

#### D. Extend existing code using inheritance

Unit #1 and 2

#### E. Understand error handling

1. Understand runtime exceptions.  
   Unit #1
2. Throw runtime exceptions  
   Unit #1

#### F. Reason about programs

1. Pre- and post-conditions  
   Unit #1
2. Assertions  
   Unit #1

#### G. Analysis of algorithms

1. Informal comparisons of running times  
   Unit #2
2. Exact calculation of statement execution counts  
   Unit #2
3. Big-Oh notation  
   Unit #2
4. Worst-case and average-case time and space analysis  
   Unit #2

#### H. Numerical representations and limits

1. Representations of numbers in different bases  
   Unit #1
2. Limitations of finite representations (e.g., integer bounds, imprecision of floating-point representations, and round-off error)  
   Unit #1

### IV. Standard Data Structures

Data structures are used to represent information within a program. Abstraction is an important theme in the development and application of data structures.

#### A. Simple data types (int, boolean, double)

Unit #1

#### B. Classes

Unit #1

#### C. One-dimensional arrays

Unit #1 and 2

#### D. Two-dimensional arrays

Unit #1 and 2

#### E. Linked lists (singly, doubly, circular)

Unit #3

#### F. Stacks

Unit #3

#### G. Queues

Unit #3

#### H. Trees

Unit #3

#### I. Heaps

Unit #3
J. Priority queues
K. Sets
L. Maps

<table>
<thead>
<tr>
<th>V. Standard Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard algorithms serve as examples of good solutions to standard problems. Many are intertwined with standard data structures. These algorithms provide examples for analysis of program efficiency.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A. Operations on AB-level data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Traversals</td>
</tr>
<tr>
<td>2. Insertions</td>
</tr>
<tr>
<td>3. Deletions</td>
</tr>
<tr>
<td>4. Iterators</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B. Searching</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Sequential</td>
</tr>
<tr>
<td>2. Binary</td>
</tr>
<tr>
<td>3. Hashing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C. Sorting</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Selection</td>
</tr>
<tr>
<td>2. Insertion</td>
</tr>
<tr>
<td>3. Mergesort</td>
</tr>
<tr>
<td>4. Quicksort</td>
</tr>
<tr>
<td>5. Heapsort</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VI. Computing in Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>A working knowledge of the major hardware and software components of computer systems is necessary for the study of computer science, as is the awareness of the ethical and social implications of computing systems. These topics need not be covered in detail but should be considered throughout the course.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A. Major hardware components</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Primary and secondary memory</td>
</tr>
<tr>
<td>2. Processors</td>
</tr>
<tr>
<td>3. Peripherals</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B. System software</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Language translators/compilers</td>
</tr>
<tr>
<td>2. Virtual machines</td>
</tr>
<tr>
<td>3. Operating systems</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C. Types of systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Single-user systems</td>
</tr>
<tr>
<td>2. Networks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D. Responsible use of computer systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. System reliability</td>
</tr>
<tr>
<td>2. Privacy</td>
</tr>
<tr>
<td>3. Legal issues and intellectual property</td>
</tr>
<tr>
<td>4. Social and ethical ramifications of computer use</td>
</tr>
</tbody>
</table>