Computers in Mathematics Education, Viggo P. Hansen and Marilyn J. Zweng (editors), pages 194–201. Yearbook of the National Council of Teachers of Mathematics, 1984

Advanced Placement Computer Science

Stephen J. Garland
Dartmouth College

1. Introduction

The Advanced Placement (AP) Program is a cooperative educational endeavor of the College Board. Based on the fact that many students can complete college level studies in their secondary schools, it represents a desire of schools and colleges to foster such experiences. Like other programs of the Board, this program is national, its policies are determined by representatives of member institutions, and its operational services are provided by Educational Testing Service.

Advanced Placement serves three groups: students who wish to pursue college-level studies while still in secondary school, schools that desire to offer these opportunities, and colleges that wish to encourage and recognize such achievement. It does this by providing practical descriptions of college-level courses to interested schools and the results of examinations based on these descriptions to colleges upon request by individual students. Participating colleges, in turn, grant credit and/or appropriate placement to students who have done well on the examinations.

The decision to include computer science among the AP offerings was made in 1981. Prior to that time three possibilities for new AP courses in the mathematical sciences were being considered: linear algebra, probability and statistics, and computer science. Separate panels met to discuss the feasibility of an AP course in each area. For a variety of curricular and economic reasons, it was decided that AP courses in the first two areas were not presently feasible. However, there was strong support for an offering in computer science.

The computer science panel cited two reasons in particular for introducing a new AP course. (1) A significant amount of computer-related activity in both elementary and secondary schools already existed and was on the rise. (2) The introduction of an AP Computer Science course would foster further development of both secondary and undergraduate curricula in computer science in much the same way as the introduction of an AP Calculus course 25 years ago fostered the development of the mathematics curriculum. At the secondary level, the visibility of an AP course would provide guidance for the development of the computing curriculum at lower grade levels. And at the college level, the presence of students having completed an AP course would provide an incentive for colleges to upgrade their introductory courses in keeping with curricular recommendations [3, 4] issued by professional societies in computing.

The College Board accepted the panel’s recommendations and appointed a Computer Science Development Committee to draft a course description and related materials. The initial membership of that committee consisted of the author as chairman, Sara Baase of San Diego State University, Horace Butler of the Maudlin (S. C.) High School, Phillip Miller of Carnegie Mellon University, Bruce Presley of the Lawrenceville School, David C. Rine of Western Illinois University, and Sally A. Sloan of the Minneapolis Public Schools. Staff support was provided by J. R. Jefferson Wadkins and James S. Braswell of ETS.

The College Board set an operational date for having the course in place by the 1983-84 school year with the first examination to be given in May, 1984. To date the Development Committee has published a detailed *Course Description* [1] and a separate *Teacher's Guide* [2]. The information that follows is based primarily on the contents of the *Course Description*.

2. Course Goals

The goals of an AP Computer Science course are comparable to those of a year-long introductory course offered in college and university computer science departments. Thus the course covers topics that normally comprise six or more semester hours of college-level work in computer science, and it is based on courses that could be credited toward a computer science major. As such, the course is more than an introduction to programming in a specific language such as Basic, Fortran, or Pascal; such courses at the college level rarely occupy a full year and often carry only one or two semester hours of credit.

It is not expected that all students in an AP Computer Science course will major in computer science at the college level. The course is intended to serve both as an introductory course for students interesting in a computer science major and as a substantial service course for students who will major in other disciplines that require significant involvement with computing.

The broad goals of an AP Computer Science course are as follows:

1. The student will be able to design and implement computer-based solutions to problems in several application areas.
2. The student will learn well-known algorithms and data structures.
3. The student will be able to develop and select appropriate algorithms and data structures to solve problems.
4. The student will be able to code fluently in a well-structured fashion using an accepted high-level language. (The first offerings of the AP Computer Science Examination will require knowledge of Pascal.)
5. The student will be able 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.
6. The student will be able to recognize the ethical and social implications of computer use.

An AP Computer Science course is not intended to be the sole, or even the primary, course in computing at the secondary level. Just as secondary schools offering AP Calculus offer many other courses in mathematics, secondary schools may well teach other courses in computing to larger numbers of students. Though some students may be introduced to computing through an AP course, the course is intended (as are all Advanced Placement courses) as a first-year college course capable of being taught in those secondary schools that choose to do so.

3. Prerequisites

Since secondary schools differ widely in the ways they introduce students to computing, there are no required computing prerequisites for an AP Computer Science course. Instead, the course's prerequisites include a familiarity with mathematical notation at the level of a second course in algebra, experience in problem solving, and an appreciation of the need to structure and develop a given topic in a logical manner. Knowledge of any particular mathematical notation is less important than possession of an attitude toward notation that reflects a well-founded confidence that unfamiliar symbolism can be mastered and will eventually serve as an aid to understanding.

Mathematics courses have traditionally offered secondary school students the best available opportunity to acquire such an attitude. It is important that secondary school students and their advisors understand that any significant computer science course builds upon a foundation of mathematical reasoning that should be acquired prior to attempting such a course. Therefore, a computer science course is not a substitute for the usual college-preparatory mathematics courses.

4. Course Content

The major emphasis in an AP Computer Science course is on programming methodology, algorithms, and data structures. Applications of computing provide the context in which these subjects are treated; they are used to motivate the need for particular algorithms and data structures, as well as to provide topics for programming assignments to which students can apply their knowledge. A particular programming language constitutes the vehicle for implementing computer-based solutions to particular problems, but the language is not the focal point of the course. Treatments of computer systems and the social implications of computing are integrated into the course.

The specific topics to be covered in an AP Computer Science course are enumerated in the Topic Outline found in the appendix to this paper. Some comments about the major headings in that outline follow; more detailed comments can be found in the *Course Description* [1].

4.1. Programming methodology

Programming methodology or, more broadly, software methodology concerns itself with the development of high-quality software systems, that is, of useful, useable, reliable, maintainable, and extensible software systems. It is no accident that programming methodology is listed first in the Topic Outline. Since most of what students learn in an AP course will be applied to the process of creating computer programs, methodologies that enhance this process are of central concern.

Good programming methodology provides the means for producing quality software in an economical and timely fashion, whether that software be for a student's own use or for use in a commercial application. Modern programming languages have done much to enhance the quality of software by facilitating modular programming, structured programming, and top-down design. Disciplined approaches to the program development process have enabled programmers to make better use of the tools provided by programming languages.

But there is more to programming methodology than the use of a modern programming language and the application of disciplined techniques. Programming also involves many other skills: the recognition and selection of appropriate data structures; the selection of comprehensible, verifiable, and extensible algorithms over ones that simply minimize length of code or execution time; the ability to analyze an algorithm for speed, space, and clarity, and then to make reasonable trade-offs among these factors; and the knowledge of when and how to document a program. In short, programming in the truest sense of the word encompasses the entire subject matter of an AP course.

Programming methodology, then, is not meant to be taught as a self-contained package at the beginning of the course, but rather to be integrated throughout the course. Methodology is taught and assimilated best not by dictum, but by repeated applications of general principles to the construction and analysis of sample programs.

4.2. Features of programming languages

In order for students to practice principles of programming methodology and to apply their knowledge of algorithms and data structures to actual problems, it is essential that they spend a substantial amount of time writing computer programs. For this activity to be fruitful, students must be able, or must learn, to program well in a language which supports good methodology and which facilitates the implementation of specific algorithms and data structures.

The AP Computer Science Development Committee spent much time discussing which programming languages provided adequate support for an AP course, both in terms of the structures they provided for organizing data, the flow of control, and the modular structure of a program, and also in terms of their suitability for various applications. It also debated whether or not to constrain the choice of a language for an AP course, recognizing that good programmers can write well in almost any language, but that certain programming languages encourage good programming habits more than others. And it considered the support provided for teaching computer science using various languages by existing textbooks and by computer systems commonly available to secondary schools.

As a result of its deliberations, and after surveying secondary schools for their reaction to its options, the Development Committee selected Pascal as the sole language to be used in the AP Computer Science examination at this time. That selection was made fully cognizant of the fact that, in isolated instances, the goals of an AP Computer Science course could be accomplished in an existing secondary school computing environment with a language other than Pascal. However, the chances seemed too great that a large number of secondary schools might try to teach such a course with a less-than-adequate language and with disastrous consequences for their students taking the AP Computer Science examination. Consequently, it was decided that the interests of the students nationwide who will be taking the AP Computer Science examination are best served by being possibly too restrictive rather than being too permissive.

Because the programming language BASIC now enjoys such wide use at the secondary school level, some comments are probably in order to explain why the commonly available versions of BASIC are not acceptable vehicles for an AP course. Schools may still choose, however, to use BASIC in other, more elementary courses.

Most versions of BASIC have many inadequacies, and no attempt will be made to illustrate them all; but two examples should serve to give the flavor of the kind of shortcomings that permeate BASIC. In Pascal it is easy to distinguish the loop

```
while x < y do x := 2*x
```

which doubles a quantity @math(x) repeatedly until the resulting quantity either equals or exceeds a quantity @math(y), from the conditional

```
if x < y then x := 2*x
```

which simply doubles @math(x) once if it is less than @math(y) and does nothing otherwise. In standard BASIC, these constructs are expressed indirectly and are harder to distinguish. For example,

```
100 if x >= y then 130
110 let x = 2*x
120 goto 100
130 ...
```

performs the same task as the Pascal loop displayed above, but it differs from

```
100 if x >= y then 130
110 let x = 2*x
130 ...
```

only by the omission of one out of four lines, and from

```
100 if x >= y then 130
110 let x = 2*x
120 goto 130
130 ...
```

only by what might appear to be to be a superficial change in one out of four lines; yet the latter two constructs perform a different task entirely, namely, they perform the same task as the Pascal if-then statement displayed above. The experienced teacher of BASIC can easily imagine further complexities that would be introduced if there were other goto-statements leading out of the body of the loop; the ease with which one may write a program that is totally incomprehensible, even to its author, quickly becomes obvious in such situations.

As a second and more important example, some versions of BASIC widely available on personal computers do not provide a mechanism for passing parameters to subroutines, thereby limiting severely the usefulness of these versions for modular programming. Some dialects of BASIC do possess features akin to those of Pascal, but those dialects having at least the minimum features required for an AP course are not commonly available and are not supported by published texts.

4.3. Data types and structures

Data types and structures enable programmers to represent and organize the information manipulated by a program. They range from simple structures provided as primitive data types within a programming language to more intricate structures that may be provided by the programming language or that may be constructed by the programmer with the aid of more primitive language features.

A treatment of data types and structures cannot be divorced from a study of the algorithms used to manipulate various structures nor from a study of applications in which various structures prove useful. Hence material in this part of the Topic Outline, as with the material in the first part, is meant to be integrated throughout the course.

4.4. Algorithms

Algorithms and algorithmic thinking are at the heart of computer science. To program is to implement algorithms.

Searching and sorting algorithms in particular provide excellent examples for use in studying the design, development, and analysis of algorithms in general. Their study is motivated by applications to information storage and retrieval, and they can be used to illustrate the variety of ways in which one may approach a single problem. Furthermore, this variety of approaches helps illustrate that the efficiency of a particular algorithm is determined more by its approach to a problem than by the ingenuity of the way in which it is coded. For example, students often take great delight in inventing quite sophisticated "improvements" to intrinsically inefficient algorithms (such as the bubble sort), whereas a different approach altogether (such as repeated merging) is usually required to achieve a true gain in efficiency.

4.5. Applications of computing

Simple examples of the applications listed in the Topic Outline should be studied in order to acquaint students with the general nature of these applications and to introduce students to the data structures and algorithms inherent in these applications. Though it should not be the intent of an AP Computer Science course to develop expertise in particular application areas, students should nonetheless become familiar with typical applications in several areas and they should appreciate how a knowledge of computer science can be brought to bear upon problems in these areas.

4.6. Computer systems

A first course in computer science should provide students with a working knowledge of the major hardware and software components of computer systems. At this level only a general understanding of the purposes and general characteristics of these components is required; a more detailed understanding of how such components operate or how they are constructed should be left to later courses.

4.7. Social implications

Given the tremendous impact computers and computing have on almost every aspect of society, it is important that intelligent and responsible attitudes about computers be developed as early as possible. Students in a first course in computer science should come to regard computer systems not as opponents to be bested by whatever wiles they themselves possess, but rather as powerful tools whose nature and use they in part determine.

Attitudes are acquired, not taught. Hence the importance of considering the ethical and social implications of computing should be stressed whenever an opportunity is available. However, no systematic coverage of these implications is required.

5. The Examination

A three-hour examination will seek to determine how well a student has mastered the concepts and techniques of the course. The examination consists of (1) a multiple-choice section and (2) a free-response section requiring students to demonstrate their ability to design, write, and document programs and procedures. Topics such as programming methodology may be tested less thoroughly by the multiple-choice questions and more thoroughly by the free-response questions.

Knowledge of computer systems and the social implications of computing will usually be tested in questions on other topics, although there will occasionally be single questions in either section that are devoted exclusively to these topics.

In determining the grade for the examination, the two sections are given equal weight. Since the examination is designed for full coverage of the subject matter, it is not expected that many students will be able to answer correctly all the questions in either the multiple-choice section or the free-response section.

6. Conclusion

In summary, an AP Computer Science course seeks to develop its students' abilities to use computing in powerful, intelligent, and responsible ways. Mastery of programming methodology, algorithms, and data structures raises a student's ability to program considerably above the novice level. An acquaintance with major applications of computing suggests both where and how that programming ability can be put to use. And an awareness of the ramifications of computer use contributes to the ability of society at large to make responsible and intelligent use of computing.

7. Appendix: Topic Outline

This appendix necessarily contains technical terminology. Readers who have not yet reached a level of expertise sufficient to understand all this terminology may be aided by reading the *Course Description* or by a brief study of some computer science texts employing the terminology.

```
A.  Programming methodology
    1.  Specification
        a.  Problem definition and requirements
        b.  Functional specifications for programs
    2.  Design
        a.  Modularization
        b.  Top-down versus bottom-up methodologies
        c.  Stepwise refinement of modules and data structures
    3.  Coding
        a.  Structure
        b.  Style, clarity of expression
    4.  Program correctness
        a.  Testing
              i.  Relation to design and coding
             ii.  Generation of test data
            iii.  Top-down versus bottom-up testing of modules
        b.  Verification
              i.  Assertions and invariants
             ii.  Reasoning about programs
        c.  Debugging
    5.  Documentation

B.  Features of programming languages
    1.  Types and declarations
        a.  Block structure
        b.  Scope of identifiers
             i.  Local identifiers
            ii.  Global identifiers
    2.  Data
        a.  Constants
        b.  Variables
    3.  Expressions and assignments
        a.  Operators and operator precedence
        b.  Standard functions
        c.  Assignment statements
    4.  Control structures
        a.  Sequential execution
        b.  Conditional execution
```

- c. Iteration (loops or repetitive execution)
- 5. Input and output
  - a. Terminal input and output
  - b. File input and output
- 6. Procedures
  - a. Subroutines and functions
  - b. Parameters
    - i. Actual and formal parameters
    - ii. Value and reference parameters
  - d. Recursive procedures
- 7. Program annotation
  - a. Comments
  - b. Indentation and formatting

C. Data types and structures

- 1. Primitive data types
  - a. Numeric data
    - i. Floating point real numbers
    - ii. Integers
  - b. Character (symbolic) data
  - c. Logical (Boolean) data
- 2. Linear data structures
  - a. Arrays
  - b. Strings
  - c. Linked lists
  - d. Stacks
  - e. Queues
- 3. Tree structures
  - a. Terminology
    - i. Nodes: root, leaf, parent, child, sibling
    - ii. Branches and subtrees
    - iii. Ordered and unordered trees
  - b. Binary trees
  - c. General tree structures (optional)
- 4. Representation of data structures
  - a. Sequential representation of linear structures
  - b. Pointers and linked data structures

D. Algorithms

- 1. Classes of algorithms
  - a. Sequential algorithms
  - b. Iterative or enumerative algorithms
  - c. Recursive algorithms
- 2. Searching
  - a. Sequential (linear) search
  - b. Binary search
  - c. Hash coded search
  - d. Searching an ordered binary tree
  - e. Linear versus logarithmic searching times
- 3. Sorting
  - a. Selection sort
  - b. Insertion sort
  - c. Exchange or bubble sort
  - d. Merge sort
  - e. Sorting using an ordered binary tree
  - f. Quicksort (optional)
  - g. Radix sort (optional)

- h. Quadratic versus n*log(n) sorting times
4. Numerical algorithms
   - a. Approximations
     - i. Zeroes of functions by bisection
     - ii. Monte Carlo techniques
     - iii. Area under a curve (optional)
   - b. Statistical algorithms
     - i. Measures of central tendency
     - ii. Measures of dispersion
   - c. Numerical accuracy
     - i. Roundoff effects
     - ii. Precision of approximations
5. Manipulation of data structures
   - a. String processing
     - i. Concatenation
     - ii. Substring extraction
     - iii. Matching
   - b. Insertion and deletion in linear structures, trees
   - c. Tree traversals

E. Applications of computing
1. Text processing
   - a. Editors
   - b. Text formatters
2. Simulation and modeling
   - a. Continuous simulation of physical processes
   - b. Discrete simulation of probabilistic events
3. Data analysis
   - a. Statistical packages
   - b. Graphical display of data
4. Data management
   - a. Information storage and retrieval
   - b. Typical business systems
5. System software
   - a. File management routines (e.g., mail systems)
   - b. Graphical software
   - b. Syntax analysis routines
     - i. Command scanners
     - ii. Evaluation of arithmetic expressions
   - c. Graphical software
6. Games
   - a. Simple puzzles (e.g., Tower of Hanoi)
   - b. Simple games (e.g, tic-tac-toe)
   - c. Searching game trees (optional)

F. Computer systems
1. Major hardware components
   - a. Primary and secondary memory
   - b. Processors
   - c. Peripherals
2. System software
   - a. Language processors
   - b. Operating systems
   - c. Graphical output facilities
3. System configuration
   - a. Microprocessor systems
   - b. Time-sharing and batch processing systems

```
        c.  Networks

G.  Social implications
    1.  Responsible use of computer systems
    2.  Social ramifications of computer applications
        a.  Privacy
        b.  Values implicit in the construction of systems
        c.  Reliability of systems
```

8. Bibliography

[1] *Advanced Placement Course Description: Computer Science*, The College Board, Advanced Placement Program, Box 2899, Princeton, NJ 08541.

[2] *Teacher’s Guide to Advanced Placement Courses in Computer Science*, The College Board, Advanced Placement Program, Box 2899, Princeton, NJ 08541.

[3] “Curriculum ’78. Recommendations for the undergraduate program in computer science,” *Communications of the Association for Computing Machinery*, Vol. 22, No. 3 (March 1979), pp. 147-166.

[4] *A Curriculum in Computer Science and Engineering*, IEEE Computer Science Education Committee Report (rev 1), IEEE Computer Society, November 1976.

Table of Contents