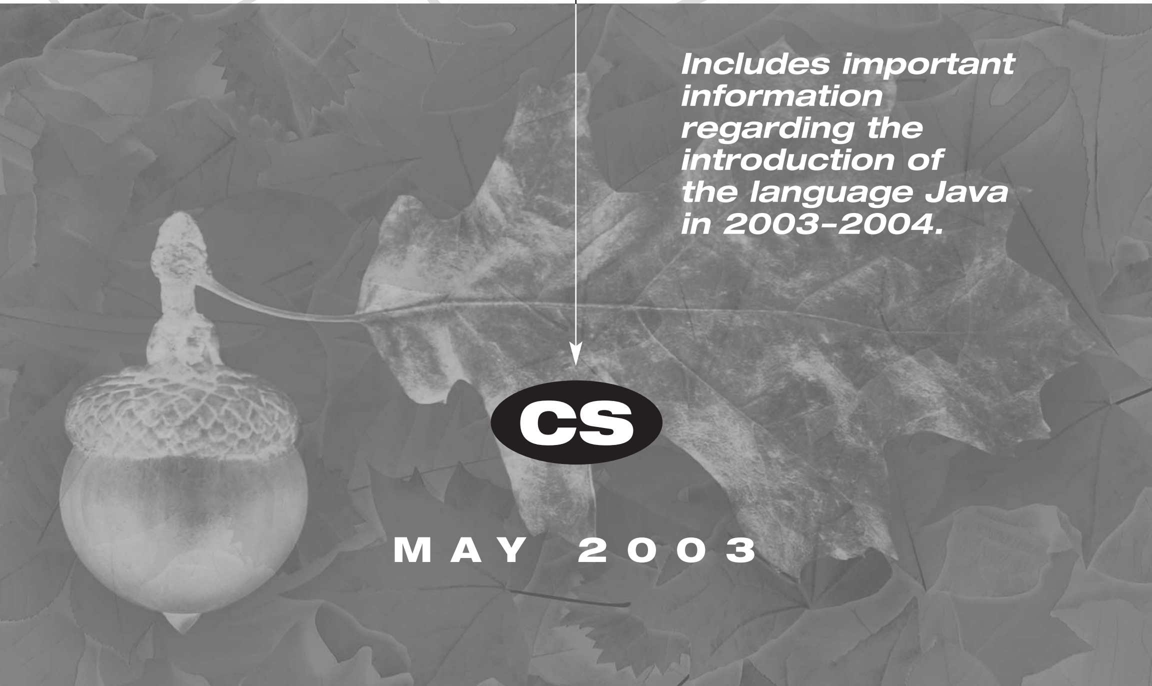
The College Board

AP® ADVANCED PLACEMENT PROGRAM®

Course Description

# COMPUTER SCIENCE



*Includes important information regarding the introduction of the language Java in 2003–2004.*

CS

MAY 2003

The College Board acknowledges that Dr. Tommy J. Boley, in his publication “New Trends in Teaching Composition” (1985) and in other writings has formulated the S.O.A.P. method of teaching composition in primary and secondary schools.

Dear Colleagues:

In 2001, more than one million high school students benefited from the opportunity of participating in AP® courses, and more than 840,000 then took the challenging AP Exams. These students felt the power of learning come alive in the classroom, and many earned college credit and placement while still in high school. Behind these students were talented, hardworking teachers — who collectively are the heart and soul of the AP Program.

The College Board is committed to supporting the work of AP teachers. This AP Course Description provides an outline of content and a description of course goals, while still allowing teachers the flexibility to develop their own lesson plans and syllabi, and to bring their individual creativity to the AP classroom. To support teacher efforts, a Teacher's Guide is available for each AP subject. Moreover, AP workshops and Summer Institutes held around the globe provide stimulating professional development for more than 60,000 teachers each year. The College Board Fellows stipends provide funds to support many teachers' attendance at these Institutes, and, in 2001, stipends were offered for the first time to teams of Pre-AP™ teachers as well.

Teachers and administrators can now also visit the official online destination for AP teachers and education professionals — AP Central™ (apcentral.collegeboard.com), which offers a new and unique set of resources, information, and tools. Here, teachers have access to a growing array of classroom resources, from textbook reviews to lesson plans, from surveys to the most up-to-date exam information. I invite all teachers, particularly those who are new to AP, to take advantage of these resources.

As we look to the future, the College Board's goal is to provide access to AP courses in every high school. Reaching this goal will require a lot of hard work. We encourage you to help us build bridges to college and opportunity by finding ways to prepare students in your school to benefit from participation in AP.

Sincerely,

Gaston Caperton

Gaston Caperton
President
The College Board

# Contents

Welcome to the AP® Program . . . 1
  AP Courses . . . 1
  AP Exams . . . 1

AP Computer Science . . . 3
  Introduction . . . 3
  The Courses . . . 4
    Goals . . . 4
    Computer Language . . . 5
    Equipment . . . 6
    Prerequisites . . . 7
    Teaching the Courses . . . 8
  Topic Outline . . . 9
  Case Studies . . . 14
  The Examinations . . . 15
    Computer Science A: Sample Multiple-Choice Questions . . . 17
      Answers to Computer Science A Multiple-Choice Questions . . . 32
    Sample Free-Response Questions . . . 33
      Suggested Solutions to Free-Response Questions . . . 40
    Computer Science AB: Sample Multiple-Choice Questions . . . 45
      Answers to Computer Science AB Multiple-Choice Questions . . . 54
    Sample Free-Response Questions . . . 55
      Suggested Solutions to Free-Response Questions . . . 60
  References . . . 63
  Appendix A . . . 64
    Defining the AP Computer Science C++ Subset: A Philosophy . . . 64
    AP Computer Science: The C++ Subset . . . 65
  Appendix B . . . 77
    AP Computer Science Java Subset . . . 77
    Language Features . . . 78
    Summary Table . . . 84
  Appendix C . . . 88
    Quick Reference – A Exam . . . 88
  Appendix D . . . 90
    Quick Reference – AB Exam . . . 90

AP Program Essentials . . . . . . . . . . 98
The AP Reading . . . . . . . . . . 98
AP Grades . . . . . . . . . . 98
Grade Distributions . . . . . . . . . . 98
AP and College Credit . . . . . . . . . . 98
Why Colleges Give Credit for AP Grades . . . . . . . . . . 99
Guidelines on Granting Credit for AP Grades . . . . . . . . . . 99
Finding Colleges That Accept AP Grades . . . . . . . . . . 100
AP Awards . . . . . . . . . . 100
AP Calendar . . . . . . . . . . 100
Test Security . . . . . . . . . . 100
Teacher Support . . . . . . . . . . 101
Pre-AP™ . . . . . . . . . . 102
Preparing Students; Preparing Teachers . . . . . . . . . . 102
Building Success . . . . . . . . . . 102
Setting the Cornerstones Workshops . . . . . . . . . . 102
AP Publications and Other Resources . . . . . . . . . . 103
Ordering Information . . . . . . . . . . 103
Print . . . . . . . . . . 104
Multimedia . . . . . . . . . . 106

# Welcome to the AP® Program

The Advanced Placement Program® is sponsored by the College Board, a nonprofit membership association. AP offers 34 college-level courses and exams in 19 subject areas for highly motivated students in secondary schools. Its reputation for excellence results from the close cooperation among secondary schools, colleges, and the College Board. Most U.S. colleges and universities grant credit, advanced standing, or both to students who have performed satisfactorily on the exams, and almost 1,500 institutions grant sophomore standing to students who meet their requirements. Approximately 13,700 high schools throughout the world participate in the AP Program; in May 2001, they administered more than 1.4 million AP Exams.

You will find more information about the AP Program at the back of this Course Description and at AP Central. The AP Central Web site is maintained for the AP Program by collegeboard.com, a destination Web site for students and parents.

## AP Courses

AP courses are available in the subject areas listed on the next page. (Unless noted, an AP course is equivalent to a full-year college course.) Each course is developed by a committee composed of college faculty and high school AP teachers. Members of these Development Committees are appointed by the College Board and serve for overlapping terms of up to four years.

## AP Exams

For each AP course, an AP Exam is administered at participating schools and multischool centers worldwide. Schools register to participate by completing the AP Participation Form and agreeing to its conditions. For more details, see the *AP Program Guide*; information about ordering a print copy or downloading this publication can be found at the back of this Course Description.

Except for Studio Art — which is a portfolio assessment — all exams have a free-response section (either essay or problem-solving) and another section consisting of multiple-choice questions. The modern language exams contain a speaking component, and the Music Theory exam includes a sight-singing task.

| **AP Subject Areas** | **AP Courses and Exams** |
|---|---|
| Art | Art History; Studio Art: Drawing Portfolio; Studio Art: 2-D Design Portfolio; Studio Art: 3-D Design Portfolio |
| Biology | Biology |
| Calculus | AB; BC |
| Chemistry | Chemistry |
| Computer Science | A*; AB |
| Economics | Macroeconomics*; Microeconomics* |
| English | Language and Composition; Literature and Composition |
| Environmental Science | Environmental Science* |
| French | Language; Literature |
| German | Language |
| Geography | Human Geography* |
| Government and Politics | Comparative*; United States* |
| History | European; United States; World |
| Latin | Literature; Vergil |
| Music | Music Theory |
| Physics | B; C: Electricity and Magnetism*; C: Mechanics* |
| Psychology | Psychology* |
| Spanish | Language; Literature |
| Statistics | Statistics* |

* This subject is the equivalent of a half-year college course.

# AP Computer Science

## Introduction

Shaded text indicates important new changes in this subject.

The Advanced Placement Program offers two computer science courses: Computer Science A and Computer Science AB. The content of Computer Science A is a subset of the content of Computer Science AB. Computer Science A emphasizes programming methodology with a concentration on problem solving and algorithm development and is meant to be the equivalent of a first-semester course in Computer Science. It also includes the study of data structures and abstraction, but these topics are not covered to the extent that they are covered in Computer Science AB. For a comparison of the topics covered, see the AP Computer Science topic outline on pages 9-13. Computer Science A may be appropriate for schools offering an AP Computer Science course for the first time, for schools whose faculty members have not yet developed sufficient expertise to cover the material in Computer Science AB, or for schools wishing to offer a choice of courses.

Computer Science AB includes all the topics of Computer Science A, as well as a more formal and in-depth study of algorithms, data structures, and abstraction. For example, binary trees are studied in Computer Science AB but not in Computer Science A. These additional topics are listed in the right-hand column of the topic outline.

The nature of both AP courses is suggested by the words "computer science" in the titles. Their presence indicates a disciplined approach to a more broadly conceived subject than would a descriptor such as "computer programming." There are no computing prerequisites for either AP course. Each is designed to serve as a first course in computer science for students with no prior computing experience.

Because of the diversity of introductory computer science courses currently offered by colleges and universities, the outline of topics described here may not match any sequence of courses exactly. The Association for Computing Machinery (ACM) and the Institute of Electrical and Electronic Engineers (IEEE) Computer Society have published standards for the content of a college-level program in computer science that include recommendations for topics to be covered in the first two years of college. The AP Computer Science A course is compatible with those topics that are covered in a typical CS1 course as described in the example curricula in the ACM/IEEE guidelines. The additional topics of the AP

Computer Science AB course are consistent with a CS2 course in those sample curricula. Some colleges and universities may organize their curricula in alternative ways so that the topics of the AP Computer Science A and AB courses are spread over the first three or four college courses, with other topics from computer science interspersed.

Either AP Computer Science course can be offered by any secondary school that has faculty who possess the necessary expertise and have access to appropriate computing facilities. It should be emphasized that these courses represent college-level achievement for which most colleges and universities can be expected to grant advanced placement and credit. Placement and credit are granted by institutions in accordance with their own policies, not by those of the College Board or the AP Program.

## The Courses

The AP Computer Science courses are introductory courses in computer science. Because the development of computer programs to solve problems is a skill fundamental to the study of computer science, a large part of the course is built around the development of computer programs or parts of programs that correctly solve a given problem. The course also emphasizes the design issues that make programs understandable, adaptable, and, when appropriate, reusable. At the same time, the development of useful computer programs and program modules is used as a context for introducing other important concepts in computer science, including the development and analysis of algorithms, the development and use of fundamental data structures, and the study of standard algorithms and typical applications. In addition, an understanding of the basic hardware and software components of computer systems and the responsible use of these systems are integral parts of the course. The topic outline on pages 9-13 summarizes the content of the AP Computer Science curriculum.

### Goals

The goals of an AP course in computer science are comparable to those in the introductory sequence of courses for computer science majors offered in college and university computer science departments. It is not expected, however, that all students in an AP Computer Science course will major in computer science at the university level. An AP Computer Science course is intended to serve both as an introductory course for computer science majors and as a course for people who will major in other disciplines that require significant involvement with computing.

The following goals apply to both of the AP Computer Science courses when interpreted within the context of the specific course.

1. Students should be able to design and implement computer-based solutions to problems in several application areas.
2. Students should learn well-known algorithms and data structures.
3. Students should be able to develop and select appropriate algorithms and data structures to solve problems.
4. Students should be able to code fluently in a well-structured fashion using the programming language C++. Students are expected to be familiar with and be able to use standard AP C++ classes.
5. Students should be able to read and understand a large program and a description of the design and development process leading to such a program. (Examples of such programs are the AP Computer Science Case Studies.)
6. Students should 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.
7. Students should be able to recognize the ethical and social implications of computer use.

## Computer Language

The content of the college-level introductory programming course has evolved significantly over the years. Starting as a treatment merely of language features, it eventually incorporated first the notions of procedures and procedural abstraction, then the use of modules and data abstraction. At many institutions, the current introductory programming course takes an object-oriented approach to programming that is based on encapsulating procedures and data. The AP Computer Science courses have also been evolving to incorporate this approach.

Current offerings of the AP Computer Science Examination require the use of C++. Those sections of the exam that require the reading or writing of actual programs will use C++. The exam will not cover all the features of C++; it will be consistent with the AP C++ subset and the standard AP C++ classes. (See Appendix A.) Students will be tested on the AP C++ classes. These classes can be found in the Computer Science section of AP Central.

The AP Computer Science Examinations will require knowledge of the programming language Java beginning with the 2003-04 academic year and the 2004 examinations. The exams will continue to cover the fundamentals of computer science taught in first-year college courses. However, those sections of the examination that require the reading or writing of actual programs will use Java rather than C++. The examination will not cover all the features of Java; it will be consistent with the AP Java subset (see Appendix B). A quick reference containing the required library classes and methods will be provided as part of the examination (see appendices C and D for the A and AB versions of the quick reference). The AP Java subset is also available at AP Central.

## Equipment

Students should have access to a computer system that represents relatively recent technology. The system should be able to compile, in a matter of seconds, programs of size comparable to the AP Computer Science Case Study, and response time should be reasonably rapid. This will require large hard disk drives either on individual machines or shared via a network.

Each student in the course should have a minimum of three hours per week alone on a computer throughout the academic year; additional time is desirable. This access can be made available at any time during the school day or after school and need not be made available to all students in the AP course simultaneously. It should be stressed that (1) this requirement represents a bare minimum of access; and (2) this time is not instructional time at a computer with the teacher or a tutor, but is time that the student spends alone at a computer in addition to the instructional time. Schools that do not allow their facilities to be used after school hours may wish to reevaluate such a policy in light of the needs of their students who take an AP Computer Science course.

Schools offering AP Computer Science will need to have C++ compiler software and enough memory in their lab machines so that students will be able to compile C++ programs efficiently. At a minimum, the hardware configuration will need large hard drives and sufficient memory to support current operating systems and compilers.

Beginning with the 2003-04 academic year, the AP Computer Science Examinations will require the programming language Java. Schools will have to acquire Java compiler software and, in some situations, upgrade the operating system. Schools with older hardware may have to increase the memory and/or upgrade the processors in their lab machines based on the requirements of the Java compiler.

## Prerequisites

The necessary prerequisites for entering either one of the AP Computer Science courses 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. A student in either AP Computer Science course should be comfortable with functions and the concepts often found in the uses of functional notation, such as `f(x) = x + 2` and `f(x) = g(h(x))`. It is important that secondary school students and their advisers understand that any significant computer science course builds upon a foundation of mathematical reasoning that should be acquired before attempting such a course.

Schools that offer Computer Science AB may have students who do well on the topics covering programming methodology but do not deal as well with the more advanced material on data structures and analysis of algorithms. Such students might be advised to concentrate on the topics listed for Computer Science A and to take the Computer Science A examination instead.

Some schools may offer only Computer Science A and encourage students who move at a faster pace to study the topics covered by the Computer Science AB outline. They can then take the AP Examination in Computer Science AB rather than the examination in Computer Science A. Other schools may offer both courses and restrict enrollment in Computer Science AB to students who have some prior programming experience. Some schools may offer each course for a full year, while others cover each in one semester. This will vary according to the background of the students and the teacher.

The curricular organization necessary to prepare students to enter an AP Computer Science course is compatible with that recommended to prepare students for the AP courses in calculus. It should be emphasized, however, that calculus is not a prerequisite for an AP Computer Science course. In fact, no relationship need exist between AP offerings in calculus and computer science. A computer science course is not a substitute for the usual college-preparatory mathematics courses or a substitute for a course in calculus.

One prerequisite for an AP Computer Science course, competence in written communication, deserves special attention. Documentation plays a central role in the programming methodology that forms the heart of an AP Computer Science course. Students should have already acquired facility in written communication before entering such a course.

## Teaching the Courses

The teacher should be prepared to present a college-level first course in computer science. Each AP Computer Science course is more than a traditional programming course. The emphasis in these courses is on procedural and data abstraction, object-based programming methodology, algorithms, and data structures.

Because of the dynamic nature of the computer science field, AP Computer Science teachers will continually need to update their skills. Some resources that may assist teachers in professional development are AP Computer Science workshops and summer institutes, and Web sites such as AP Central. For information on workshops, teachers should contact their College Board Regional Office or go to AP Central.

One particular area of change is the evolution of programming languages and programming paradigms. Teachers should endeavor to keep current in this area by investigating different programming languages. Future AP Examinations will be based on the language Java, and teachers are encouraged to start preparing for the change. The AP Java subset and auxiliary materials will be accessible on the Web.

## Topic Outline

Following is an outline of the major topics covered by the AP Examinations in Computer Science. The ordering here is intended to define the scope of the course, but not necessarily the sequence. The topics in the right-hand column will not be tested on the Computer Science A examination.

### I. 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.

| ***Computer Science A and AB*** | ***Computer Science AB only*** |
|---|---|
| A. Problem definition<br>1. Specification of purpose and goals<br>2. Identification of objects and classes (abstract data types)<br>3. Identification of class responsibilities (operations on abstract data types)<br>B. Program design<br>1. Design of user/client interface<br>2. Choice of data structures and algorithms<br>3. Function decomposition<br>4. Identification of reusable components from existing code | |

## II. Program Implementation

The overall goals of program implementation parallel those of program design. Modules of the program that fill common needs should be built so that they can be reused easily in other programs. Procedural and data abstraction are important parts of program implementation.

| ***Computer Science A and AB*** | ***Computer Science AB only*** |
|---|---|
| A. Implementation techniques | |
|   1. Methodology | |
|     a. Object-based development | |
|     b. Top-down development | |
|   2. Use of abstraction | |
|     a. Abstract data types (including encapsulation and information hiding) | |
|     b. Procedural abstraction | |
| B. Programming constructs | |
|   1. Declaration | |
|     a. Constant declarations | |
|     b. Variable declarations | |
|     c. Class declarations | |
|     d. Function declarations | |
|     e. Parameter declaration | |
|       i. Value | |
|       ii. Reference | |
|       iii. Constant reference | |
|   2. Input and output | |
|     a. Interactive | |
|     b. Files | |
|   3. Control | |
|     a. Sequential | |
|     b. Conditional | |
|     c. Repetition | |
|       i. Iteration | |
|       ii. Recursion | |
|     d. Functions (including member functions) | |
| C. Generic data types and functions | |
|   1. AP classes | |
| | 2. Templates |

## III. Program Analysis

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

| ***Computer Science A and AB*** | ***Computer Science AB only*** |
|---|---|
| A. Testing | |
| 1. Testing classes and modules in isolation | |
| 2. Identifying boundary cases and generating appropriate test data | |
| 3. Integration testing | |
| B. Debugging | |
| 1. Categorizing errors: compile-time, run-time, logic | |
| 2. Identifying and correcting errors | |
| 3. Techniques: using a debugger, adding extra output statements, hand-tracing | |
| C. Understanding and modifying existing code | |
| D. Handling errors — robust behavior | |
| E. Reasoning about programs | |
| 1. Pre/post conditions | |
| 2. Assertions | |
| | 3. Invariants |
| F. Analysis of algorithms | |
| 1. Informal comparisons of running times | |
| 2. Exact calculation of statement execution counts | |
| | 3. Big-Oh notation |
| | 4. Worst case/average case time and space analysis |
| G. Numerical limits Limitations of finite representations (e.g., integer bounds, imprecision of floating-point representations, and round-off error) | |

## IV. Standard Data Structures

Data structures are the means by which the information used by a program is represented within the program. Abstraction is an important theme in the development and application of data structures.

***Computer Science A and AB***

A. Simple data types (e.g., int, char, bool, double, strings)
B. Aggregate data types
   1. Heterogeneous (structs)
   2. Homogeneous (arrays)
C. Classes

***Computer Science AB only***

D. Linked lists
E. Stacks
F. Queues
G. Trees
H. Heaps
I. Priority queues

## V. Standard Algorithms

Standard algorithms can serve as examples of good solutions to standard problems. Programs implementing them can serve as models of good program design. They provide examples for analysis of program efficiency. Many are intertwined with standard data structures.

***Computer Science A and AB***

A. Operations on data structures
   1. Traversals
   2. Insertion
   3. Deletion

***Computer Science AB only***

B. Operations on dynamic data structures
   1. Traversals
   2. Insertion
   3. Deletion
   4. Allocation/deallocation of memory

| ***Computer Science A and AB*** | ***Computer Science AB only*** |
|---|---|
| C. Searching | |
| 1. Sequential (linear) | |
| 2. Binary | |
| | 3. Hashing |
| D. Sorting | |
| 1. Selection | |
| 2. Insertion | |
| 3. Mergesort merge algorithm | |
| 4. Quicksort partition algorithm | |
| | 5. Heapsort |

## VI. Computer Systems

A working knowledge of the major hardware and software components of computer systems is necessary for the study of computer science, as is the importance of considering the ethical and social implications of computing systems. These topics need not be covered in detail, but they should be considered throughout the course.

| ***Computer Science A and AB*** | ***Computer Science AB only*** |
|---|---|
| A. Major hardware components | |
| 1. Primary and secondary memory | |
| 2. Processors | |
| 3. Peripherals | |
| B. System software | |
| 1. Language translators | |
| 2. Separate compilation | |
| 3. Operating systems | |
| C. Types of systems | |
| 1. Single-user systems | |
| 2. Networks | |
| D. Responsible use of computer systems | |
| 1. System reliability | |
| 2. Privacy | |
| 3. Legal issues and intellectual property | |
| 4. Social and ethical ramifications of computer use | |

## Case Studies

A case study is a document that includes the statement of a problem, one or more programs that solve the problem, and a written description of one expert's path from problem statement to solution program(s). The write-up describes the choices made for design and implementation and the justification for the choices that were made.

Case studies provide a vehicle for presenting many of the topics of the AP Computer Science courses. They provide examples of good style, programming language constructs, fundamental data structures, algorithms, and applications. Moreover, case studies provide an economical way to deal with large programs. Large programs give the student practice in the management of complexity and motivate the use of certain programming practices (including thorough procedural decomposition, intermodule communication through parameter passing, and selection of data structures tailored to the needs of the problem) in a much more obvious way than do small programs.

Case studies are most valuable, however, in teaching programming methodology. They allow the teacher to show concretely the design and implementation decisions leading to the solution of a problem and thus to focus more effectively on those aspects of the programming process. This approach gives the student a model of the programming process as well as a model program. The use of case studies also gives the student a context for seeing the importance of good design when a program is to be modified.

The 2003 AP Computer Science Examinations will include questions based on the case study described in the document "AP Marine Biology Case Study." Both the A and AB examinations will contain at least five multiple-choice questions and one free-response question covering material from the case study. Printed excerpts from the case study programs will accompany the examinations.

Questions will deal with activities such as the following:

a. modifying the procedural and data organization of the case study program to correspond to changes in the program specification;
b. extending the case study program by writing new code;
c. evaluating alternatives in procedural and data organization;
d. evaluating alternative incremental development strategies;
e. understanding how the components of the program fit together; and
f. developing test data.

For the AP Marine Biology Case Study appearing on the 2003 examinations, sample questions appear in the teacher's manual. The text and code for the "AP Marine Biology Case Study" are available for downloading from AP Central.

## The Examinations

The AP Examinations for Computer Science A and Computer Science AB are each three hours long and seek to determine how well students have mastered the concepts and techniques contained in the respective course outlines. Before the examination date, students must decide which of the two examinations they will take. In most cases students will prepare during the year for one examination or the other. Some students enrolled in the AB course, however, may not feel comfortable with some of its more advanced topics. Such students might prefer to take the Computer Science A examination.

Each examination consists of two sections: a multiple-choice section (40 questions in 1 hour and 15 minutes), which tests proficiency in a wide variety of topics, and a free-response section (4 questions in 1 hour and 45 minutes), which requires the student to demonstrate the ability to solve problems involving more extended reasoning.

The multiple-choice and the free-response section of both AP Computer Science Examinations require students to demonstrate their ability to design, write, analyze, and document programs and subprograms. As noted in the AP C++ subset, the AB examination may include free-response questions that ask about design as well as implementation of classes. A design question would provide students with a description of the type of information and operations on that information that an object should encapsulate. Students would then be required to provide part or all of a class declaration to define such objects. An example of this type of question appears in the sample free-response questions for Computer Science AB (see question 2 on pages 58-59).

A design question may require a student to develop a solution that includes the following:

- declaration of constructor(s) and member functions with
  - meaningful names
  - appropriate parameters
  - appropriate return types
  - use of `const` qualifier for functions and parameters when appropriate
- appropriate data representation
- appropriate placement of data and member functions into `public` and `private` sections
  - all data should be `private`
  - all client accessible operations must be specified as `public` member functions

Minor points of syntax are not tested on the examinations. All code given in the exams is consistent with the AP C++ subset and AP C++ classes. All student responses involving code must be written in C++. Students are expected to be familiar with and able to use the standard AP C++ classes. For both the multiple-choice and the free-response sections of the examinations, a quick reference to both the case study and the AP C++ classes will be provided.

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

## Computer Science A: Sample Multiple-Choice Questions

Following is a representative set of questions. Questions marked with an asterisk are also representative of AB examination questions. (See the answer key for the Computer Science A multiple-choice questions on page 32.) In this section of the examination, as a correction for haphazard guessing, one-fourth of the number of questions answered incorrectly will be subtracted from the number of questions answered correctly. The 2003 examination will include at least five multiple-choice questions based on the "AP Marine Biology Case Study." (See the teacher's manual for the "AP Marine Biology Case Study" for examples.)

*Directions:* Determine the answer to each of the following questions or incomplete statements, using the available space for any necessary scratchwork. Then decide which is the best of the choices given and fill in the corresponding oval on the answer sheet. No credit will be given for anything written in the examination booklet. Do not spend too much time on any one problem.

Note: Assume that the standard libraries (e.g., `iostream.h`, `fstream.h`, `math.h`, etc.) and the AP C++ classes are included in any programs that use the code segments provided in individual questions. A Quick Reference to the AP C++ classes is provided as part of the exam.

1. Consider the following functions.

```
void Myst(int a, int & b)
{
  a *= b;
  b = 2 + a;
}

void Test()
{
  int u = 2;
  int v = 3;
  Myst(u, v);
  cout << u << " " << v << endl;
}
```

What is printed as a result of the call `Test()` ?

(A) `2 3`
(B) `2 8`
(C) `6 3`
(D) `6 8`
(E) `8 3`

2. Assume that the following definitions have been made.

```
int num = 10;
int sum;
```

Which of the following code segments correctly computes the sum of the first 10 multiples of 5 (i.e., 5, 10, . . ., 50)?

(A)
```
sum = 5;
while (num > 1)
{
  sum += sum;
  num--;
}
```

(B)
```
sum = 5;
while (num > 0)
{
  sum += sum;
  num--;
}
```

(C)
```
sum = 0;
while (num > 0)
{
  num--;
  sum += 5 * num;
}
```

(D)
```
sum = 0;
while (num >= 0)
{
  num--;
  sum += 5 * num;
}
```

(E)
```
sum = 0;
while (num > 0)
{
  sum += 5 * num;
  num--;
}
```

3. Consider the following functions.

```
int F1(int len)
{
  int tot = 0;
  int k = 0;

  while (k < len)
  {
    tot += k;
    k++;
  }
  return tot;
}
```

```
int F2(int len)
{
  int tot = 0;
  int k;

  for (k = 0; k < len; k++)
  {
    tot += k;
  }

  return tot;
}
```

For which values of `len` do `F1(len)` and `F2(len)` return the same result?

I. Any value less than zero
II. The value zero
III. Any value greater than zero

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) I, II, and III

4. Consider the following function.

```
void Mystery(int n)
{
  if (n >= 2)
  {
    Mystery(n / 3);
  }
  cout << n << " ";
}
```

Which of the following is output as a result of the call `Mystery(27)` ?

(A) `0 27`
(B) `1 3 9 27`
(C) `3 9 27`
(D) `27 0`
(E) `27 9 3`

5. Consider the following function.

```
int Fun(const apstring & str)
{
  int n;
  int k;
  n = str.length();
  for (k = 0; k < str.length(); k++)
  {
    if (str[k] == 'x')
    {
      n--;
    }
  }
  return n;
}
```

Assume that `s` is an initialized `apstring`. Which of the following best characterizes the value returned by the call `Fun(s)` ?

(A) The number of occurrences of the character `'x'` in the string
(B) The number of occurrences of characters other than `'x'` in the string
(C) The position of the first occurrence of the character `'x'` in the string
(D) The position of the last occurrence of the character `'x'` in the string
(E) The position of the first occurrence of a character other than `'x'` in the string

6. Consider the following functions.

```
apstring Laugh(const apstring & word, int num)
{
  int k;
  apstring result = "";
  for (k = 0; k < num; k++)
  {
    result += word;
    result += " ";
  }
  return result;
}

void Test()
{
  apstring str("ha");
  cout << Laugh(Laugh(str, 2), 3) << endl;
}
```

Which of the following is output as a result of the call `Test()` ?

(A) `ha`
(B) `ha ha`
(C) `ha ha ha`
(D) `ha ha  ha ha  ha ha`
(E) Nothing will be output because a runtime error will occur.

*7. Consider the following class declaration.

```
class TheaterTicket
{
  public:
    TheaterTicket();
    void SetInfo(const apstring & date,
                 const apstring & location,
                 int seat);
    // other public member functions not shown

  private:
    // not shown
};
```

Consider modifying the `TheaterTicket` class so that it is possible to initialize variables of type `TheaterTicket` with ticket information when they are declared, as well as to set their values later using member function `SetInfo`. For example, the following code should define and initialize two `TheaterTicket` variables.

```
TheaterTicket t1;
t1.SetInfo("May 5, 1999", "AA", 22);

TheaterTicket t2("April 29, 1999", "B", 10);
```

Which of the following best describes the additional member function that should be provided?

(A) An overloaded version of `SetInfo` with no arguments
(B) An overloaded version of `SetInfo` with three arguments
(C) A default constructor
(D) A constructor with three arguments
(E) A constructor with five arguments

Questions 8-9 refer to the following class declaration.

```
class SalesPerson
{
  public:
    SalesPerson(); // constructor

    void SetSales(int month, double sales);
    // sets values for one month’s sales figures

    void PrintAnnualSales() const;
    // prints values for the annual sales
    // other public member functions not shown

  private:
    apvector<double> mySales;
    double ComputeAnnualSales();
    // Totals the sales for the last 12 months
};
```

8. The function `ComputeAnnualSales` is NOT a public member function because

   (A) functions that return a double value cannot be public member functions
   (B) `ComputeAnnualSales` is <u>not</u> intended to be used by clients of the class
   (C) `ComputeAnnualSales` <u>is</u> intended to be used by clients of the class
   (D) public member functions are used for input and output only
   (E) `ComputeAnnualSales` uses the private data member `mySales`, therefore, it must be a private member function

9. Consider the following definition.

   ```
   SalesPerson clerk;
   ```

   Of the following statements, which sets the fifth month’s sales for salesperson `clerk` to `3705.87` ?

   (A) `SalesPerson clerk(5, 3705.87);`
   (B) `clerk = SetSales(5, 3705.87);`
   (C) `SetSales(5, 3705.87);`
   (D) `clerk.SetSales(3705.87, 5);`
   (E) `clerk.SetSales(5, 3705.87);`

*10. A *sequential search* algorithm is used to determine whether a given integer is stored in an array of 1,000 integers. In each of the following cases, what is the maximum number of array items that might be examined during a sequential search?

Case 1 — The elements are unordered.
Case 2 — The elements are sorted in ascending order.
Case 3 — The elements are sorted in descending order.

| | Case 1 | Case 2 | Case 3 |
|---|---|---|---|
| (A) | 1,000 | 1,000 | 1,000 |
| (B) | 1,000 | 11 | 11 |
| (C) | 1,000 | 11 | 1,000 |
| (D) | 1,000 | 500 | 500 |
| (E) | 500 | 500 | 500 |

*11. The expression `!((x > y) && (y <= 3))` is equivalent to which of the following?

(A) `(x > y) && (y <= 3)`
(B) `(x > y) || (y <= 3)`
(C) `(x < y) || (y >= 3)`
(D) `(x <= y) || (y > 3)`
(E) `(x <= y) && (y > 3)`

*12. Consider designing a data structure to record information about post office boxes using the following structure.

```
struct BoxInfo
{
  int BoxNum;
  int NumLetters;  // number of letters currently
                   // in this box
};
```

Information about post office boxes that currently contain at least one letter will be stored in an array. Each element is of type `BoxInfo`. Two possible implementations are being considered.

| | |
|---|---|
| Method A: | Store the array entries in arbitrary order. |
| Method B: | Store the array entries in sorted order by box number. |

Consider the following operations.

| | |
|---|---|
| Operation 1: | Increment the number of letters in a specified box in the array. |
| Operation 2: | Add a new box to the list of boxes, with a given number of letters. |

Which of the following is true?

(A) Both Operation 1 and Operation 2 can be implemented more efficiently using Method A than using Method B.
(B) Both Operation 1 and Operation 2 can be implemented more efficiently using Method B than using Method A.
(C) Operation 1 can be implemented more efficiently using Method A; Operation 2 can be implemented more efficiently using Method B.
(D) Operation 1 can be implemented more efficiently using Method B; Operation 2 can be implemented more efficiently using Method A.
(E) Operations 1 and 2 can be implemented equally efficiently using either method.

13. The following incomplete function is intended to return the index of the first occurrence of the minimum value in the array A.

```
int MinLoc(const apvector<int> & A)
// precondition: A.length() ≥ 1
{
  int k;
  int minIndex = 0;

  for (k = 1; k < A.length(); k++)
  {
    if ( <condition> )
    {
      <statement>
    }
  }
  return minIndex;
}
```

Which of the following could be used to replace `<condition>` and `<statement>` so that function `MinLoc` works as intended?

| | *<condition>* | *<statement>* |
|---|---|---|
| (A) | `A[k] < minIndex` | `minIndex = A[k];` |
| (B) | `A[k] < A[minIndex]` | `minIndex = A[minIndex];` |
| (C) | `A[k] < A[minIndex]` | `minIndex = k;` |
| (D) | `A[k] > minIndex` | `minIndex = A[k];` |
| (E) | `A[k] > A[minIndex]` | `minIndex = k;` |

*Questions 14-15 rely on the following information.

Consider the following class declaration.

```
class Person
{
  public:
    Person(); // constructor

    int NumChildren() const;
    // postcondition: returns this person's number
    //                of children

    const Person & KthChild(int k) const;
    // postcondition: If this person has at least k
    //                children, then returns this
    //                person's kth child.
    //                If k is 1, returns the person's
    //                first child.

    // other member functions not shown

  private:
    // not shown
};
```

Also assume that the following definitions have been made.

```
Person P;
int k;
bool oddNum;
```

*14. Which of the following code segments correctly sets `oddNum` to `true` if person `P` has an odd number of children, and to `false` otherwise?

I.
```
oddNum = ((P.NumChildren() % 2) == 1);
```

II.
```
if ((P.NumChildren() % 2) == 0)
  oddNum = false;
else
  oddNum = true;
```

III.
```
oddNum = false;
for (k = 1; k <= P.NumChildren(); k++)
  oddNum = !oddNum;
```

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) I, II, and III

*15. Assume `total` is of type `int`. Which of the following code segments correctly sets `total` to the number of `P`'s grandchildren?

(A)
```
total = P.NumChildren();
```

(B)
```
total = P.NumChildren(P.NumChildren());
```

(C)
```
total = 0;
for (k = 1; k <= P.NumChildren(); k++)
{
  total += P.KthChild(k);
}
```

(D)
```
total = 0;
for (k = 1; k <= P.NumChildren(); k++)
{
  total += P.KthChild(k).NumChildren();
}
```

(E)
```
total = 0;
for (k = 1; k <= P.NumChildren(); k++)
{
  total += P.NumChildren().KthChild(k);
}
```

16. Consider the following code segment.

```
apmatrix<int> M(10, 10, 1);
int k;

for (k = 0; k < 10; k++)
{
  M[k][k] = 0;
}
```

Which of the following best describes what happens when the loop is executed?

(A) All entries of matrix `M` are set to zero.
(B) One entry of matrix `M` is set to zero.
(C) One column of matrix `M` is set to zero.
(D) One row of matrix `M` is set to zero.
(E) One diagonal of matrix `M` is set to zero.

*17. Consider the following definitions.

```
const int MAXITEMS = <some positive integer>;

int NumLessThanValue(const apvector<int> & A,
                     int value)
// precondition: Array A contains MAXITEMS items.
{
  int total = 0;
  int index;

  for (index = 0; index < MAXITEMS; index++)
  {
    if (A[index] < value)
    {
      total++;
    }
  }
  return total;
}
```

What is the greatest possible value that can be returned by function `NumLessThanValue`?

(A) `0`
(B) `1`
(C) `MAXITEMS - 1`
(D) `MAXITEMS`
(E) `MAXITEMS * 2`

*18. Consider the following incomplete function.

```
bool Unique(const apvector<int> & A, int n)
// precondition:  A contains n values, n > 1.
//                Values are stored in positions
//                0 to n - 1 in unsorted order.
// postcondition: returns true if there are no
//                duplicate values in A;
//                otherwise, returns false.

{
  int j, k;

  <body of Unique>
}
```

Which of the following code segments can be used to replace `<body of Unique>` so that function `Unique` satisfies its postcondition?

I.
```
for (j = 0; j < n - 1; j++)
{
  for (k = j + 1; k < n; k++)
    if (A[j] == A[k])
      return false;
}
return true;
```

II.
```
for (j = 1; j < n; j++)
{
  for (k = 0; k < j; k++)
    if (A[k] == A[j])
      return false;
}
return true;
```

III.
```
for (j = 0; j < n; j++)
{
  for (k = 1; k < n; k++)
    if (A[k] == A[j])
      return false;
}
return true;
```

(A) I only
(B) II only
(C) I and II only
(D) I and III only
(E) I, II, and III

**Answers to Computer Science A Multiple-Choice Questions**

| | | | | | |
|---|---|---|---|---|---|
| 1 – B | 4 – B | 7 – D | 10 – A | 13 – C | 16 – E |
| 2 – E | 5 – B | 8 – B | 11 – D | 14 – E | 17 – D |
| 3 – E | 6 – D | 9 – E | 12 – D | 15 – D | 18 – C |

## Sample Free-Response Questions

Following is a representative set of questions. Questions marked with an asterisk are also representative of AB examination questions. Additional sample questions can be found in the AP section of the College Board Web site. The 2003 examination will include one free-response question based on the “AP Marine Biology Case Study.” (See the teacher’s manual for the “AP Marine Biology Case Study” for examples.)

*Directions:* SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN C++.

*Note:* Assume that the standard libraries (e.g., `iostream.h`, `fstream.h`, `math.h`, etc.) and the AP C++ classes are included in any program that uses a program segment you write. If other classes are to be included, that information will be specified in individual questions. Unless otherwise noted, assume that all functions are called only when their preconditions are satisfied. A Quick Reference to the AP C++ classes is included as part of the exam.

1. (a) Write function `CommaPosition`, as started below. If there is no comma in `name`, then `CommaPosition` returns –1. Otherwise, `CommaPosition` returns the index, or position of the comma in `name`. Assume that there is at most one comma in `name`.

For example:

| `name` | `CommaPosition(name)` |
|---|---|
| `W,Jeff` | `1` |
| `Smith,` | `5` |
| `Jones` | `-1` |
| `,` | `0` |
| `Doe,Roxanne` | `3` |
| `,Susan` | `0` |

Complete function `CommaPosition` below.

```
int CommaPosition(const apstring & name)
// precondition:  name contains at most
//                one comma.
// postcondition: returns -1 if there is no
//                comma in name or returns
//                the position/index of the
//                comma in name.
```

(b) Write function `PrintFirstLast`, as started below. `PrintFirstLast` should print the name(s) stored in parameter `name` as follows: If there is no comma in `name` then nothing is printed; otherwise, `PrintFirstLast` should print all characters after the comma, if any, followed by a space, followed by all characters before the comma, if any, followed by a new line.

For example:

```
name                Output of the call PrintFirstLast(name)


W,Jeff                    Jeff W
Smith,                     Smith
Jones
,
Doe,Roxanne               Roxanne Doe
,Susan                    Susan
```

In writing `PrintFirstLast`, you may call function `CommaPosition` specified in part (a). Assume that `CommaPosition` works as specified, regardless of what you wrote in part (a).

Complete function `PrintFirstLast` below.

```
void PrintFirstLast(const apstring & name)
// precondition: name contains at most one comma.
```

* 2. (a) Write function `SumCross`, as started below. `SumCross` returns the sum of the values in the row with index `row` and the column with index `col` in the matrix `M`. The value in position `M[row][col]` is included in the sum only once. For example, consider the following matrix `M`, where `M.numrows() == 3` and `M.numcols() == 4`.

Matrix `M`

| | | | |
|---|---|---|---|
| 7 | 22 | 1 | 33 |
| 5 | 3 | 10 | 4 |
| 66 | 44 | 2 | 55 |

The call `SumCross(M, 1, 2)` returns the value `25`, which is the sum of the values in the row with index `1` and the column with index `2`, including the value `10` only once.

Complete function `SumCross` below.

```
int SumCross(const apmatrix<int> & M,
             int row, int col)
// precondition: 0 ≤ row < M.numrows();
//               0 ≤ col < M.numcols()
```

(b) Write function `RemoveCross`, as started below. `RemoveCross` removes the row with index `row` and the column with index `col` from matrix `M`. For example, consider the following matrix `M`, where `M.numrows() == 5` and `M.numcols() == 6`. After the call `RemoveCross(M, 2, 3)`, the original row with index `2` and column with index `3` have been removed and `M.numrows() == 4` and `M.numcols() == 5`.

Matrix `M` Prior to call

| | | | | | |
|---|---|---|---|---|---|
| 11 | 22 | 33 | 5 | 44 | 55 |
| 22 | 33 | 44 | 6 | 55 | 66 |
| 4 | 5 | 6 | 7 | 8 | 9 |
| 33 | 44 | 55 | 8 | 66 | 77 |
| 44 | 55 | 66 | 9 | 77 | 88 |

Matrix `M` After the Call `RemoveCross(M, 2, 3)`

| | | | | |
|---|---|---|---|---|
| 11 | 22 | 33 | 44 | 55 |
| 22 | 33 | 44 | 55 | 66 |
| 33 | 44 | 55 | 66 | 77 |
| 44 | 55 | 66 | 77 | 88 |

Recall that the `apmatrix` member function `resize` works as follows:

For an `apmatrix M, if row < M.numrows()` and `col < M.numcols()`, then after the call `M.resize(row, col)`, `M.numrows() == row, M.numcols() == col`, and for any `j, 0 ≤ j < row`, and `k, 0 ≤ k < col, M[j][k]` has the same value that it did before the call.

Complete function `RemoveCross` below.

```
void RemoveCross(apmatrix<int> & M,
                 int row, int col)
// precondition:  0 <= row < M.numrows();
//                0 <= col < M.numcols()
// postcondition: row with index row and
//                column with index col
//                have been removed from M
//                and M has been resized
```

* 3. This question involves reasoning about two implementations of a data structure called a priority list. A priority list is a collection of items, each of which contains a data field and a unique integer priority. The “maximal” item in a priority list is the element whose integer priority has the greatest value. An element’s data and integer priority are provided when inserting that element into a priority list. Only the maximal element, the item with the highest integer priority, is accessible in a priority list.

The declaration for class `Plist` is given below.

```
struct itemType
{
  apstring myInfo;
  int myPri;
};

class Plist
{
  public:
    Plist(); // constructs and initializes an
             // empty priority list
    bool IsEmpty() const;
    // postcondition: returns true if the
    //                priority list is empty;
    //                otherwise, returns false

    void Insert(const apstring & name, int pri);
    // postcondition: inserts item with data
    //                name and priority

    void FindMax(apstring & name, int & pri) const;
    // postcondition: sets name and pri to the
    //                data and priority of the
    //                maximal item

    void DeleteMax();
    // postcondition: deletes the maximal
    //                element

    // other public member functions not shown

  private:
    apvector<itemType> myList;
    int myQuantity; //quantity of data in myList
};
```

Consider the following two methods for implementing priority lists.

Method 1: A priority list is an unsorted array of items. The `Insert` operation inserts an item at the end of the array.

Method 2: A priority list is an array of items sorted by priority data member from smallest to largest (so that the last item in the array is always the maximal item of the priority list). The `Insert` operation inserts an item so that the array remains sorted by priority.

In this question you will write functions `Insert` and `DeleteMax` for <u>one</u> of the methods for implementing priority lists described at the beginning of this question. You **must** use the same method in writing both `Insert` and `DeleteMax`. It may be the case that it is easier to write code for one of the methods than for the other. Think carefully before choosing a method. Circle below the method you will use to implement both `Insert` and `DeleteMax`.

Method 1 Method 2

(a) Complete function `Insert` below.

```
void Plist::Insert(const apstring & name, int pri)
// precondition:  no item in priority list has
//                priority pri
// postcondition: A new item has been inserted
//                whose data member myInfo has
//                value name and whose data
//                member myPri has value pri
```

(b) Complete function `DeleteMax` below.

```
void Plist::DeleteMax()
// precondition:  IsEmpty() == false,
//                (priority list is not empty)
// postcondition: the maximal item has been
//                deleted
```

## Suggested Solutions to Free-Response Questions

**Note:** There are many correct variations of these solutions.

*Question 1*

a)
```
int CommaPosition(const apstring & name)
{
  int k;

  for (k = 0; k < name.length(); k++)
  {
    if (name[k] == ',')
    {
      return k;
    }
  }
  return -1;
}
```

b)
```
void PrintFirstLast(const apstring & name)
{
  int k;
  int comma = CommaPosition(name);

  if (comma != -1)
  {
    for (k = comma + 1; k < name.length(); k++)
    {
      cout << name[k];
    }
    cout << ' ';
    for (k = 0; k < comma; k++)
    {
      cout << name[k];
    }
    cout << endl;
  }
}
```

Alternate solution to Question 1

a)
```
int CommaPosition(const apstring & name)
{
  int pos = name.find(',');

  if (pos != npos)
  {
    return pos;
  }
  else
  {
    return -1;
  }
}
```

b)
```
void PrintFirstLast(const apstring & name)
{
  int comma = CommaPosition(name);

  if (comma >= 0)
  {
    if (comma + 1 < name.length())
    {
      cout << name.substr(comma + 1,
                          name.length() - comma - 1);
    }
    cout << " ";
    cout << name.substr(0, comma) << endl;
  }
}
```

*Question 2*

a)
```
int SumCross(const apmatrix<int> & M,
             int row, int col)
{
  int sum = 0;
  int index;

  for (index = 0; index < M.numcols(); index++)
  {
    sum += M[row][index];
  }
  for (index = 0; index < M.numrows(); index++)
  {
    sum += M[index][col];
  }
  return (sum - M[row][col]);
}
```

b)
```
void RemoveCross(apmatrix<int> & M, int row, int col)
{
  int r, c;

  for (r = 0; r < M.numrows(); r++)
  {
    for (c = col; c < M.numcols() - 1; c++)
    {
      M[r][c] = M[r][c + 1];
    }
  }
  for (c = 0; c < M.numcols(); c++)
  {
    for (r = row; r < M.numrows() - 1; r++)
    {
      M[r][c] = M[r + 1][c];
    }
  }
  M.resize(M.numrows() - 1, M.numcols() - 1);
}
```

*Question 3*

```
Method 1  (unsorted vector)

a) void Plist::Insert(const apstring & name, int pri)
   {
     if (myQuantity == myList.length())
     {
       myList.resize(2 * myList.length() + 1);
     }
     myList[myQuantity].myInfo = name;
     myList[myQuantity].myPri = pri;
     myQuantity++;
   }

b) void Plist::DeleteMax()
   {
     int k, maxIndex = 0;

     for (k = 1; k < myQuantity; k++)
     {
       if (myList[k].myPri > myList[maxIndex].myPri)
       {
         maxIndex = k;
       }
     }
     myList[maxIndex] = myList[myQuantity - 1];
     myQuantity--;
   }
```

```
Method 2  (sorted vector)

a) void Plist::Insert(const apstring & name, int pri)
   {
     int pos;
     int temp = 0;

     if (myQuantity == myList.length())
     {
       myList.resize(2 * myList.length() + 1);
     }
     pos = myQuantity;
     myQuantity++;

     // uses an insertion sort type algorithm
     while ((pos != 0) && (myList[pos -1].myPri >= pri))
     {
       myList[pos] = myList[pos - 1];
       pos--;
     }
     myList[pos].myInfo = name;
     myList[pos].myPri = pri;
   }

b) void Plist::DeleteMax()
   {
     myQuantity--;
   }
```

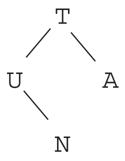
## Computer Science AB: Sample Multiple-Choice Questions

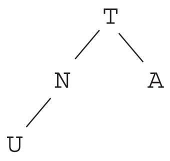
Following is a representative set of questions. Questions marked with an asterisk in the Computer Science A questions are also representative of Computer Science AB questions. (See the answer key for Computer Science AB multiple-choice questions on page 54.) In this section of the examination, as a correction for haphazard guessing, one-fourth of the number of questions answered incorrectly will be subtracted from the number of questions answered correctly. The 2003 examination will include at least five multiple-choice questions based on the "AP Marine Biology Case Study." (See the teacher's manual for the "AP Marine Biology Case Study" for examples.)

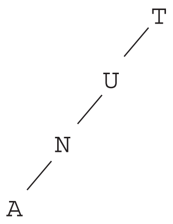
*Directions:* Determine the answer to each of the following questions or incomplete statements, using the available space for any necessary scratchwork. Then decide which is the best of the choices given and fill in the corresponding oval on the answer sheet. No credit will be given for anything written in the examination booklet. Do not spend too much time on any one problem.

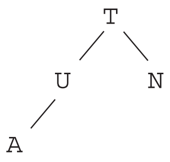
*Note:* Assume that the standard libraries (e.g., `iostream.h`, `fstream.h`, `math.h`, etc.) and the AP C++ classes are included in any programs that use the code segments provided in individual questions. A Quick Reference to the AP C++ classes is provided as part of the exam.

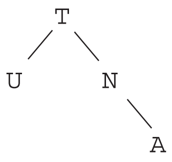
1. For which of the following binary trees does a preorder traversal produce the sequence “T U N A” while a postorder traversal produces the sequence “N U A T”?

(A)


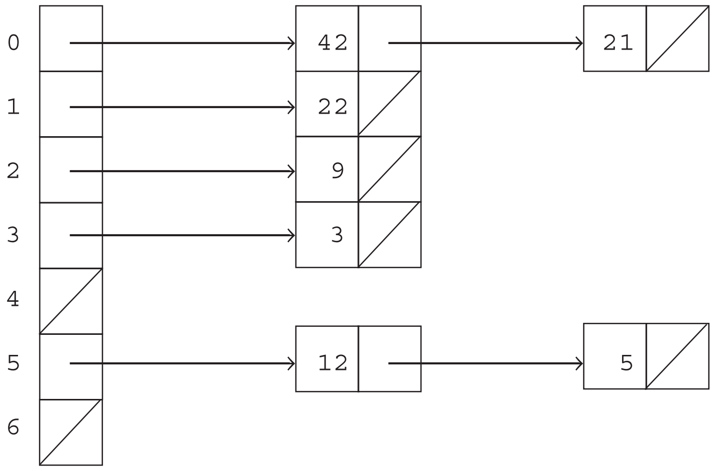
(B)


(C)


(D)


(E)


2. The hash table below results from inserting the values `3`, `5`, `21`, `12`, `9`, `42`, and `22` in that order. The hash table is an array indexed from `0` to `6` in which collisions are resolved by chaining. In the diagram below, / indicates a null pointer.



Which of the following could be the hashing function?

(A) `Value % 7`
(B) `Value / 7`
(C) `Value`
(D) `(Value - 1) / 7`
(E) `(Value - 1) % 7`

3. Assume that linked lists are implemented using the following declaration.

```
struct ListNode
{
  int data;
  ListNode * next;
  ListNode(int D, ListNode * N);
};
```

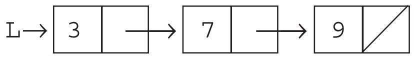
```
   //struct constructor
ListNode::ListNode(int D, ListNode * N)

   : data(D),
     next(N)
{ } // all fields initialized in initializer list
```

Consider function `InsertDuplicate` outlined below.

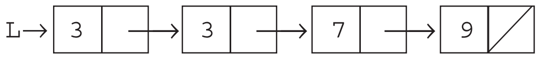
```
void InsertDuplicate(ListNode * L)
// precondition:  L is not NULL
// postcondition: The first node in list L has
//                been duplicated and inserted
//                after the first node.
{
    <more code>
}
```

For example,

Before calling `InsertDuplicate(L)`



After calling `InsertDuplicate(L)`



Which of the following code segments could be used to replace *<more code>* so that `InsertDuplicate` works as intended?

(A) `L = new ListNode(L->data, NULL);`
(B) `L = new ListNode(L->data, L);`
(C) `L = new ListNode(L->data, L->next);`
(D) `L->next = new ListNode(L->data, L);`
(E) `L->next = new ListNode(L->data, L->next);`

4. Assume that `S1` and `S2` are of type `apstack<int>` and contain the following values.

```
            S1                          S2

    4   3   5   2   6               8   9
                    ↑                   ↑
                   top                 top
```

Consider the following code segment.

```
apstack<int> S3;
int value;

while (!S1.isEmpty())
{
  S1.pop(value);
  S3.push(value);
  if (!S2.isEmpty())
  {
    S2.pop(value);
    S3.push(value);
  }
}
```

Which of the following best represents `S3` after the code segment executes?

```
                              top
                               ↓
(A)  4   8   3   9   5   2   6
(B)  8   4   9   3   5   2   6
(C)  6   9   2   8   5   3   4
(D)  9   6   8   2   5   3   4
(E)  4   3   5   2   6   8   9
```

5. Consider the following characters to be entered into an empty binary search tree.

```
'A'  'B'  'C'  'D'  'E'  'F'  'G'
```

Which of the following sequences represent(s) an order of insertion that will result in a binary search tree where each node in the tree has the same number of nodes in its left and right subtrees?

```
  I. 'D'   'B'   'F'   'A'   'E'   'C'   'G'
 II. 'D'   'E'   'F'   'G'   'C'   'B'   'A'
III. 'D'   'F'   'G'   'E'   'B'   'A'   'C'
```

(A) I only
(B) II only
(C) III only
(D) I and III only
(E) I, II, and III

Questions 6-7 refer to the following information.

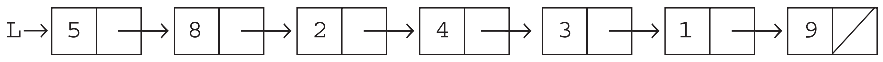
Assume that linked lists are implemented using the following declaration.

```
struct ListNode
{
  int data;
  ListNode * next;
};
```

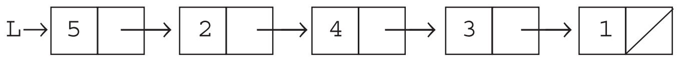
Consider the following function.

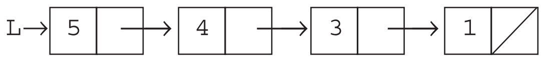
```
void Strip(ListNode * p)
// precondition: p points to the first node of a
//               nonempty list
{
  while (p != NULL)
  {
    while ((p->next != NULL) &&
           (p->next->data >= p->data))
    {
      p->next = p->next->next;
    }
    p = p->next;
  }
}
```

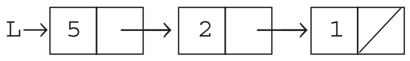
6. Suppose that `L` represents the following list.

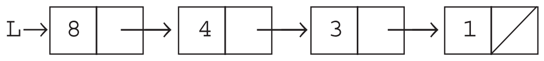


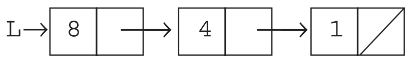
Which of the following does `L` represent after the call `Strip(L)` ?

(A) 

(B) 

(C) 

(D) 

(E) 

7. Which of the following best characterizes the worst-case runtime of function `Strip` when its parameter `p` points to a list with $N$ nodes?

(A) $O(1)$
(B) $O(\log N)$
(C) $O(N)$
(D) $O(N \log N)$
(E) $O(N^2)$

Questions 8-9 refer to the following information.

Assume that binary trees are implemented using the following declaration.

```
struct TreeNode
{
  int data;
  TreeNode * left;
  TreeNode * right;
};
```

Consider the following function.

```
int Mystery(TreeNode * p)
{
  if (p == NULL)
  {
    return 0;
  }
  else if ((p->left == NULL) && (p->right == NULL))
  {
    return 0;
  }
  else if ((p->left != NULL) && (p->right != NULL))
  {
    return (Mystery(p->left) + Mystery(p->right));
  }
  else
  {
    return (Mystery(p->left) + 1 + Mystery(p->right));
  }
}
```

8. Suppose `p` points to the root of a binary tree. Which of the following best characterizes the value returned by the call `Mystery(p)` ?

   (A) The number of nodes in the tree.
   (B) The number of leaves in the tree.
   (C) The number of nodes in the tree with exactly one child.
   (D) The number of nodes in the tree with exactly two children.
   (E) The number of nodes in the tree with either one or two children.

9. Which of the following best characterizes the worst-case runtime of function `Mystery` when it is called with a pointer to the root of a tree with $N$ nodes?

   (A) $O(1)$
   (B) $O(\log N)$
   (C) $O(N)$
   (D) $O(N \log N)$
   (E) $O(N^2)$

**Answers to Computer Science AB Multiple-Choice Questions**

| | | |
|---|---|---|
| 1 – A | 4 – C | 7 – C |
| 2 – A | 5 – D | 8 – C |
| 3 – E | 6 – C | 9 – C |

## Sample Free-Response Questions

Following is a representative set of questions. Additional sample questions can be found at AP Central. The 2003 examination will include one free-response question based on the “AP Marine Biology Case Study.” (See the teacher’s manual for the “AP Marine Biology Case Study” for examples.)

*Directions:* SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN C++.

*Note:* Assume that the standard libraries (e.g., `iostream.h`, `fstream.h`, `math.h`, etc.) and the AP C++ classes are included in any program that uses a program segment you write. If other classes are to be included, that information will be specified in individual questions. Unless otherwise noted, assume that all functions are called only when their preconditions are satisfied. A Quick Reference to the AP C++ classes is included as part of the exam.

1. Assume that binary trees of integers are implemented using the following declaration.

```
struct TreeNode
{
  int info;
  TreeNode * left;
  TreeNode * right;
};
```

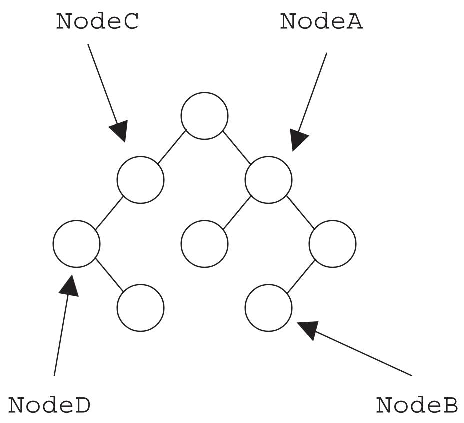
(a) Write function `IsChild`, as started below. `IsChild` returns `true` if the node pointed to by parameter `second` is a child (either left or right) of the node pointed to by parameter `first`, and returns `false` otherwise.

Complete function `IsChild` below.

```
bool IsChild(TreeNode * first, TreeNode * second)
//precondition:  second != NULL
//postcondition: returns true if second is a child
//               of first; otherwise, returns false.
```

(b) Write function `IsDescendant`, as started below. `IsDescendant` returns `true` if there is a path from the node pointed to by parameter `first` to the node pointed to by parameter `second`, where a path is a sequence of `left` or `right` children.

For example, consider the tree diagrammed below.



| Function call | Value returned |
|---|---|
| `IsDescendant(NodeC, NodeA)` | `false` |
| `IsDescendant(NodeA, NodeB)` | `true` |
| `IsDescendant(NodeB, NodeA)` | `false` |
| `IsDescendant(NodeC, NodeD)` | `true` |
| `IsDescendant(NodeA, NodeA)` | `false` |

In writing `IsDescendant`, you may call function `IsChild` specified in part (a). Assume that `IsChild` works as specified, regardless of what you wrote in part (a).

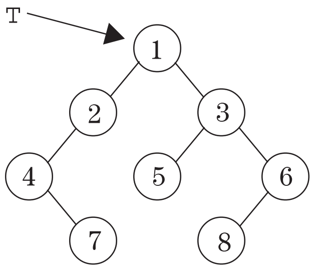
Complete function `IsDescendant` below.

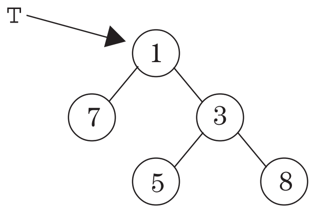
```
bool IsDescendant(TreeNode * first,
                  TreeNode * second)
// precondition:  second != NULL
```

(c) Write function `ChangeTree`, as started below. `ChangeTree` removes from `T` all nodes that have exactly one child, replacing the removed node with the child. (All removed nodes should be returned to available storage.) If `A` and `B` are nodes in `T` that do NOT have exactly one child, and if `IsDescendant(A, B)` is true before the call `ChangeTree(T)`, then `IsDescendant(A, B)` must be true AFTER the call `ChangeTree(T)`.

For example:

after the call `ChangeTree(T)`





In writing function `ChangeTree`, you may call function `HasOneChild`, defined below.

```
bool HasOneChild(TreeNode * t)
// postcondition: returns true if t has exactly
//                one child;
//                otherwise, returns false.
{
  return ((t != NULL) &&
          ((t->left == NULL && t->right != NULL) ||
           (t->left != NULL && t->right == NULL)));
}
```

Complete function `ChangeTree` below.

```
void ChangeTree(TreeNode * T)
```

2. Consider the problem of modeling savings accounts for a bank. Each account includes the following:
    - an identity number that is a positive integer;
    - a current balance in dollars, rounded to whole cents (two decimal places);
    - a monthly interest rate.

   When a new account is created, it must be assigned an identity number and interest rate. The new account may be created with an initial balance that must be non-negative; if no new balance is indicated when the account is created, it is given a balance of zero. Operations on a savings account include the following:
    - retrieve the identity number for the account;
    - retrieve the current balance in the account;
    - deposit some amount into the account, increasing the balance by that amount;
    - withdraw some amount that is less than or equal to the current balance, decreasing the balance by that amount (if an attempt is made to withdraw more than the balance, an error message is printed and the balance is unchanged);
    - calculate the monthly interest due (without changing the current balance).

   The class described above will be called `SavingsAccount`. A separate class representing the bank will store a list of `SavingsAccount` objects in an `apvector`.

   (a) Write the class declaration for `SavingsAccount` as it would appear in a `SavingsAccount.h` file. In writing the interface, you must:
    - choose appropriate function names;
    - provide the functionality specified above;
    - provide a data representation consistent with the specification above;
    - make design decisions that are consistent with information-hiding principles.

   Comments are not required, but may be used as desired to clarify what you write.

   **YOU SHOULD NOT WRITE THE IMPLEMENTATIONS OF THE MEMBER FUNCTIONS OR THE CONSTRUCTOR(S) OF THE `SavingsAccount` CLASS.**

(b) Consider the class `SavingsBank` partially specified below.

```
class SavingsBank
{
  public:

    void PostMonthlyInterest();
    // postcondition: for each account in this
    //                bank, the monthly interest
    //                due has been deposited into
    //                that account

    // constructors and other member functions
    // not shown

  private:

    apvector<SavingsAccount> myAccounts;
             // all accounts in this bank

};
```

Write the `SavingsBank` member function `PostMonthlyInterest`, which is described as follows. For each account in this bank, `PostMonthlyInterest` should calculate the monthly interest and deposit that amount into the account.

In writing `PostMonthlyInterest`, you may use any of the member functions of class `SavingsAccount` that you specified in part (a). Assume that these member functions work as specified.

Complete function `PostMonthlyInterest` below.

```
void SavingsBank::PostMonthlyInterest()
// postcondition: for each account in this
//                bank, the monthly interest
//                due has been deposited into
//                that account
```

## Suggested Solutions to Free-Response Questions

**Note:** There are many correct variations of these solutions.

*Question 1*

a)
```
bool IsChild(TreeNode * first, TreeNode * second)
{
  return  ((first != NULL) &&
           ((first->left == second)||
           (first->right == second)));
}
```

b)
```
bool IsDescendant(TreeNode * first, TreeNode * second)
{
  if (first != NULL)
  {
    if (IsChild(first, second) ||
        IsDescendant(first->left, second) ||
        IsDescendant(first->right, second))
    {
      return true;
    }
  }
  return false;
}
```

c)
```
void ChangeTree(TreeNode * & T)
{
  if (T != NULL)
  {
    ChangeTree(T->left);
    ChangeTree(T->right);
    if (HasOneChild(T))
    {
      TreeNode * temp = T;
          // will be deleting this node
      if (T->left != NULL)
          // move to appropriate only child
      {
        T = T->left;
      }
      else
      {
        T = T->right;
      }
      delete temp;
    }
  }
}
```

*Question 2*

```
class SavingsAccount
{
  public:
    SavingsAccount();
    // default constructor (needed for apvector)

    SavingsAccount(int IDNum, double rate);
    // postcondition: a SavingsAccount with identity
    //                number IDNum, interest rate of
    //                rate and current balance of zero
    //                has been created

    SavingsAccount(int IDNum, double rate,
                   double startBal);
    // precondition:  startBal >= 0.0
    // postcondition: a SavingsAccount with identity
    //                number IDNum, interest rate of
    //                rate and current balance of
    //                startBal has been created

    int IDNumber() const;
    // postcondition: returns the identity number for
    //                this account

    double CurrentBalance() const;
    // postcondition: returns the current balance for
    //                this account

    double MonthlyInterest() const;
    // postcondition: returns the monthly interest due
    //                for this account

    void Deposit(double amount);
    // precondition:  amount >= 0.0
    // postcondition: the current balance of this
    //                account has been increased by
    //                amount; if amount < 0.0, then
    //                current balance is unchanged and
    //                an error message is printed
    void Withdraw(double amount);
    // precondition:  amount <= the current balance of
    //                this account
    // postcondition: the current balance of this
    //                account has been
    //                decreased by amount;
    //                if amount > current balance, then
    //                current balance is unchanged and
    //                an error message is printed.

  private:

    int myIDNum;         // identity number for this account
    double myIntRate;    // interest rate for this account
    double myBalance;    // current balance for this account
};
```

Solution to part (b)

```
void SavingsBank::PostMonthlyInterest()
// postcondition: for each account in this bank, the
//                monthly interest due has been
//                deposited into that account
{
  double interest;
  int k;
  for (k = 0; k < myAccounts.length(); k++)
  {
    interest = myAccount[k].MonthlyInterest();
    myAccount[k].Deposit(interest);
  }
}
```

## References

A list of reference materials for C++ can be found in the *Teacher's Guide — AP Computer Science.* For further information on resources for use in an AP Computer Science class, subscribe to the discussion group that is available for computer science teachers.

AP Computer Science information about topics such as the Marine Biology Case Study, the switch to Java and Java resource materials, upcoming AP Computer Science workshops and materials from previous AP Workshops, along with links to the Chief Reader's Web page and many other useful sites can be found under Computer Science at AP Central.

# Appendix A

The AP Computer Science Development Committee recognizes that no single programming language satisfies all needs. However, C++ provides a standard mechanism, available on almost all computing platforms, for emphasizing and implementing modularity and abstraction, which are key concepts in introductory computer science. Moreover, students completing an AP course using C++ are likely to receive credit or placement at a wide range of colleges.

## Defining the AP Computer Science C++ Subset: A Philosophy

Since C++ is a large and complex language, the AP Computer Science course will not cover the entire language; a restricted subset of C++ has been chosen for use in the AP Computer Science course and examination. The criteria for defining the subset were as follows:

- It should support the activities, techniques, and constructs listed in the AP Computer  Science topic outline (the official documentation for an AP Computer Science course, see pages 9-13).
- It should include classes and constructs that minimize or allow easy detection of common novice errors.
- It should be as small as possible, while still being extensive enough for the development of programs of sufficient size and complexity for the AP Computer Science course.

The subset provides several auxiliary classes, including vector and string classes with facilities similar to those available in languages like Ada, Java, and (extended) Pascal.

An important feature of the subset is its inclusion of classes and related features that facilitate the design of programs based on modern concepts of data abstraction, encapsulation, and information hiding. Object-oriented programming in its full glory is not covered — in particular, features of C++ that allow classes to be related by inheritance are not part of the subset. The subset does, however, support the use of what is often called an "object-based" approach to programming, emphasizing data abstraction and encapsulation.

C++ counterparts to Pascal features that allow avoidance or easier detection of common programming errors are retained in the subset. Examples are type-safe input and output and range-checked vector and string classes.

C++ is not C. The AP Computer Science subset includes stream input/output, reference parameters, and definition of constants using the `const` keyword, eliminating the need for their C counterparts. The vector and string classes provided in the subset are significant improvements over C's pointer-based realizations of arrays and strings.

The subset omits a number of C++ features that are not necessary to design and implement large programs as part of an AP Computer Science course. Some features, such as inheritance, are not included because they are not part of the AP Computer Science course. Other language features either are redundant, are easily replaced, or do not provide sufficient benefit to be included in the subset. What remains is sufficiently extensive to permit large and complex programs to be developed using generally accepted C++ idioms. Additional details of the AP Computer Science subset of C++ are summarized in the following pages.

## AP Computer Science: The C++ Subset

The subset of C++ used in AP Computer Science courses is documented here. This document is only the C++ subset, it is neither a description nor a curriculum for an AP Computer Science course. A detailed topic outline of the AP Computer Science courses is provided in the main portion of this course description on pages 9-13. A more detailed description and rationale for the subset follows the subset.

The most current information about the AP C++ Subset and the AP C++ classes is available at AP Central.

In the three-column format used in the subset description below, any feature or topic listed in the second (AB) column is specifically not part of the A course. The third (comment) column explains and elaborates on the topics listed in the first two columns.

Four levels of understanding are used in the topics outline: read, use, modify, and implement. For example, the outline says that AP Computer Science A students should be able to "read, use, and modify classes." This means that they should be able to read class declarations, use the classes in client programs they write, and modify a class implementation by adding a new member function or modifying an existing member function. AP Computer Science A students are not expected to design and implement classes from scratch. When the word "use" modifies a language feature, as in "use `const` member functions," students are expected to be able to write programs that use the feature.

A special section, "Topics not Covered", includes features of C++ that are not part of the AP Computer Science C++ subset and therefore will not be tested on the AP Computer Science exam. This section is divided into three subsections. The first subsection lists language features that teachers may want to include in their courses, but that will not be tested on the AP Computer Science exams. The second subsection lists language features that we do not view as useful in the AP Computer Science courses. The third subsection lists language features that we view as dangerous — these features should be avoided.

Teachers and students are free to use language features in these "not covered" sections, but the features will not be tested using questions that are on the AP Computer Science examinations.

| **A and AB course** | **AB course only** | **Comment** |
|---|---|---|
| | ***Classes*** | |
| Read, use, and modify classes. | Design and implement classes. | *APCS A students read class declarations, write client programs, add/modify member functions.* |
| Read, use, and implement constructors, including initializer lists. | | *Constructors should use initializer lists as opposed to assignments to data because sometimes initializers are required, so use them for uniformity.* |
| Differentiate between public and private. | | *No public data are used in classes. Since inheritance is not part of APCS, there is no reason to use protected functions/data.* |

| A and AB course | AB course only | Comment |
|---|---|---|
| ***Classes (continued)*** | | |
| Read class definitions that include use of `*this`. | Use, modify, and implement class member functions that include `*this`. | |
| Implement overloaded functions; use overloaded operators. | Implement overloaded operators, including `operator<<` (but friend not used). | *Students will NOT be tested on recognizing that functions differing in return type only are overloaded incorrectly.* |
| Use and implement `const` member functions. | Recognize when to make a member function `const`. | |
| Use AP string class. | | *The AP string class is a limited, safe subset of the standard string class.* |
| Use templated AP vector and matrix classes. | Use and reimplement templated AP stack and queue classes. | |
| | Design and implement templated classes and functions. | |
| Differentiate interface and implementation of class (`.h` and `.cpp` file). | | *C++ permits the ideas to be treated separately.* |

| A and AB course | AB course only | Comment |
|---|---|---|
| ***Classes (continued)*** | | |
| Read, use, and implement structs, implementing constructors and initializer lists in structs when appropriate. | Implement linked lists and trees using structs. | *A struct is a class in which all data is public. Constructors facilitate dynamic creation of structs.* |
| ***Properties of Language*** | | |
| Understand short circuit evaluation. | | *Boolean expressions such as* `(a && b)` *are evaluated left-to-right; expression evaluation stops when the value of the entire expression can be determined. This means that constructs such as* `while(k < n && a[k]!= key)` *can be used safely.* |
| Use built-in types: `int, char, double, bool.` | | For compilers that do not use 32 bit integers, `long int` may be used, otherwise no modifiers `short, long, signed, unsigned.` |
| Read and use enum. | | |

| A and AB course | AB course only | Comment |
|---|---|---|
| ***Properties of Language (continued)*** | | |
| Use operators `+, -, *, /, %,` `++,--,` `&&,\|\|,!,` `==,<, >, !=, <=,` `>=, =, +=, -=, *=,` `%=, /=` and insertion extraction for I/O `<<, >>`. | | *Use* `++` *and* `--` *only as shorthand for* `+=1` *and* `-=1;` *do NOT use these operators in expressions like* `a[k++] = 0.` |
| Use `break`. | | *Used in* `switch` *statements, can be used in loops. The* `continue` *statement will not be tested.* |
| Use function syntax for typecasts, e.g., `cout << double(x)/3.` | | *For some types, e.g.,* `long int,` *another form of cast is necessary:* `(long int) x.` *When compilers support the* `static_cast< >` *operator it will be used.* |
| | Use pointers: operators `*, ->,` `new, delete; NULL` preferred to `0`; use `==` and `!=` with pointers. | *The address of operator* `&` *is used to test for aliasing in* `operator=,` *but not used elsewhere. Pointer arithmetic, pointers to functions, and pointer comparison using < and > will NOT be tested.* |

| A and AB course | AB course only | Comment |
|---|---|---|
| ***Properties of Language (continued)*** | | |
| Use `[ ][ ]` in matrix class for indexing. | | *Reinforce vector of vectors and mimic notation for built-in arrays.* |
| Use `#include` and `#ifndef` idiom in header files. | | *No other use of preprocessor;* `const` *variables used instead of* `#define`. |
| Use escape sequences, `\n, \t, \\, \',\"` | | |
| Use value, reference, and constant reference parameters. | | |
| Recognize that values returned by operator `[ ]` for string and vector can be assigned to, e.g., `s[0] = 'a'` is allowed. | Read and use reference return types and understand the implications of returning reference to local variable. | *No testing of reference return types for APCS A students.* |
| Use the functions `fabs, pow` and `sqrt` from `<math.h>`. | | |
| Use constants `INT_MAX, INT_MIN` from `<limits.h>` | | |

| A and AB course | AB course only | Comment |
|---|---|---|
| ***Stream Properties*** | | |
| Use `cin, cout, <<, >>,endl.` | | *Assume input data is "type-safe", e.g., when* `int` *expected* `int` *is read.* `cerr` *not tested, but can be useful.* |
| Use `ifstream, ofstream.` | | *Students will use open, but it will not be tested since parameters to open may be compiler specific.* |
| Use `getline.` | | `getline()` *is a free function in the AP string class, consistent with the standard string class function* `getline.` |
| Use `istream &, ostream &` as formal parameters. | | *Students should realize that different types of streams can be passed as arguments when formal parameter is* `istream &,` *but inheritance hierarchy of streams will not be tested.* |

| **A and AB course** | **AB course only** | **Comment** |
|---|---|---|
| ***NOT part of APCS C++*** | | |
| **Not tested, but potentially useful** | | |
| `assert` | | |
| default parameters | | *Function overloading can be used instead.* |
| `typedef` | | *More useful in C than in C++ when classes are used.* |
| | Implement copy constructors, destructors, and operator= | |
| `iomanip` (but `endl` used) | | *Students and teachers will find manipulators useful in formatting output, but* `iomanip` *will not be tested.* |
| stream member functions `eof()`, `good()`, `bad()`, `fail()`, `ignore()`, etc. | | *Use* `while (cin >> x)` *idiom, or test return value of* `getline` *for end of input.* |
| string streams | | *Input string streams are useful for parsing lines of data; output string streams facilitate conversion to string values.* |

| A and AB course | AB course only | Comment |
|---|---|---|
| ***NOT part of APCS C++ (continued)*** | | |
| **Not tested, but potentially useful** | | |
| `get(char &)` | | *Useful for reading stream input one character at a time.* |
| functions in `<math.h>`, e.g., `sin, cos, tan, floor, ceil` | | *Many programs require these functions, but their use will not be tested.* |
| functions in `<ctype.h>`, e.g., `isalpha, isdigit, islower, ispunct, isspace, isupper, tolower, toupper` | | *Facilitates system independent testing and conversion of characters, but not tested.* |
| constants in `<float.h>`, `DBL_MIN, DBL_MAX` | | *Can be useful, but will not be tested.* |
| command line arguments, `argc, argv[ ]` | | *Can be useful, but will not be tested.* |
| `operators: comma, ?:, bitwise, sizeof` | | *These operators are not essential.* |
| reference variables | | *Reference parameters and reference return types are part of the AP C++ subset, but reference variables are not.* |

| A and AB course | AB course only | Comment |
|---|---|---|
| ***NOT part of APCS C++ (continued)*** | | |
| **Not tested, but potentially useful** | | |
| inline functions | | *Concerns with efficiency at the level of use of inline functions are not part of the AP course. In general, code will not appear in class declarations (defaulting to inline).* |
| friend classes and functions | | *Use public Display/ Print member function instead of overloaded* `operator<<` *when overloaded* `operator<<` *would require use of "friend".* |
| promotion | | *Implicit promotion or conversion, especially in parameter passing, will not be tested.* |
| inheritance | | *While inheritance is a central theme of object-oriented programming, this topic and its implementation in C++ are not part of the current AP C++ subset.* |

| A and AB course | AB course only | Comment |
|---|---|---|
| ***NOT part of APCS C++ (continued)*** | | |
| **Not tested, but potentially useful** | | |
| union | | *Not necessary in programs studied and implemented in AP courses.* |
| **Troublesome and warned against** | | |
| built-in arrays | | *No need for this given use of templated vector class. The built-in array type is fraught with problems, e.g., range checking, pointer similarity. Recognition of limitations makes built-in arrays an interesting topic for study.* |
| C-style `(char *)` strings | | *No need for this given the apstring class.* |
| `goto` | | *Although there are occasions when* `goto` *can be useful, indiscriminate use is dangerous;* `goto` *is NOT part of the AP C++ subset.* |
| `new[ ]/delete[ ]` and `malloc/free` | | *No need for* `[ ]` *(except with built-in arrays) and use* `new/delete` *rather than* `malloc/free.` |

| A and AB course | AB course only | Comment |
|---|---|---|
| ***NOT part of APCS C++ (continued)*** | | |
| **Troublesome and warned against** | | |
| `stdio.h` | | `scanf/printf,` *etc. NEVER used.* |
| `&,` the address of operator | | *Used only in test for aliasing in* `operator=.` |

# Appendix B

## AP Computer Science Java Subset

The AP Java subset is intended to outline the features of Java that may appear on AP Computer Science Examinations. The AP Java subset is NOT intended as an overall prescription for computer science courses — the subset itself will need to be supplemented in order to cover a typical introductory curriculum. For example, input/output is essential to programming and can be done in many different ways. Because of this, specific input/output features are not tested on the AP Computer Science Exam.

This appendix describes the Java subset that students will be expected to understand when they take the AP Computer Science Exam. A number of features are also mentioned that are potentially relevant in a CS1/2 course but are not specifically tested on the AP Computer Science Exam.

Three principles guided the formulation of the subset:

1. Enable the test designers to formulate meaningful questions
2. Help students with test preparation
3. Enable instructors to follow a variety of approaches in their courses

To help students with test preparation, the AP Java subset was intentionally kept small. Language constructs and library features were omitted that did not add significant functionality and that can, for the formulation of exam questions, be expressed by other mechanisms in the subset. For example, inner classes or the StringBuffer class are not essential for the formulation of exam questions — the exam uses alternatives that can be easily understood by students. Of course, these constructs add significant value for programming. Omission of a feature from the AP Java subset does not imply any judgment that the feature is inferior or not worthwhile.

The AP Java subset gives instructors flexibility in how they use Java in their courses. For example, some courses teach how to perform input/output using streams or readers/writers, others teach graphical user interface construction, and yet others rely on a tool or library that handles input/output. For the purpose of the AP Computer Science Exam, these choices are incidental and are not central for the mastery of computer science concepts. The AP Java subset does not address handling of user input at all. That means that the subset is not complete. To create actual programs, instructors need to present additional mechanisms in their classes.

The following section contains the language features that may be tested on the AP Computer Science Exam. A summary table is provided that outlines the features that are tested on the A and AB exams, the AB exam only, and those features that are useful but are not tested on either exam. A list specifying which Standard Java classes and methods will be used on the exam is available at AP Central. There will be no extra AP classes provided as part of the subset.

## Language Features

1. The primitive types `int`, `double`, and `boolean` are part of the AP Java subset. The other primitive types `short`, `long`, `byte`, `char`, and `float` are not in the subset. In particular, students need not be aware that strings are composed of `char` values. Introducing `char` does not increase the expressiveness of the subset. Students already need to understand string concatenation, `String.substring`, and `String.equals`. Not introducing `char` avoids complexities with the `char/int` conversions and confusion between `"x"` and `'x'`.
2. Arithmetic operators: `+, -, *, /, %` are part of the AP Java subset.
3. The increment/decrement operators `++` and – are part of the AP Java subset. These operators are used only for their side effect, not for their value. That is, the postfix form (for example, `x++`) is always used, and the operators are not used inside other expressions. For example, `a[x++]` is not used.
4. The assignment operator = is part of the AP Java subset. The combined arithmetic/assignment operators `+=, -=, *=, /=, %=` are part of the AP Java subset although they are used simply as a shorthand and will not be used in the adjustment part of a for loop.
5. Relational operators `==, !=, <, <=, >, >=` are part of the AP Java subset.
6. Logical operations `&&, ||, !` are part of the AP Java subset. Students need to understand the "short circuit" evaluation of the `&&` and `||` operators. The logical `&, |` and `^` and the bit operators `<<, >>, >>>, &, ~, |, ^` are not in the subset.
7. The ternary `?:` operator is not in the subset.

8. The numeric casts `(int)` and `(double)` are part of the AP Java subset. Since the only primitive types in the subset are `int`, `double`, and `boolean`, the only required numeric casts are the cast `(int)` and the cast `(double)`. Students are expected to understand "truncation towards 0" behavior as well as the fact that positive floating-point numbers can be rounded to the nearest integer as `(int)(x + 0.5)`, negative numbers as `(int)(x - 0.5)`.

9. String concatenation + is part of the AP Java subset. Students are expected to know that concatenation converts numbers to strings and invokes `toString` on objects. String concatenation can be less efficient than using the `StringBuffer` class. However, for greater simplicity and conceptual clarity, the `StringBuffer` class is not in the subset.

10. The escape sequences inside strings `\\`, `\"`, `\n` are part of the AP Java subset. The `\t` escape and Unicode `\uxxxx` escapes are not in the subset. The `\'` escape is only necessary inside character literals and is not in the subset.

11. User input is not part of the AP Java subset. There are many possible ways for supplying user input; e.g., by reading from a `BufferedReader` that is wrapped around `System.in`, reading from a stream (such as a file or an URL), or from a dialog box. There are advantages and disadvantages to the various approaches. In particular, reading from `System.in` is both fraught with complexities (two nested readers and the handling of checked exceptions) and considered old fashioned by some instructors. The exam does not prescribe any one approach. Instead, if reading input is necessary, it will be indicated in a way similar to the following:

    ```
    double x = call to a method that reads a floating-
               point number;
    ```

    or

    ```
    double x = IO.readDouble(); // read user input
    ```

    Processing string input (e.g., with `StringTokenizer`) and converting strings to numeric values (e.g., with `Integer.parseInt`) is not in the subset.

12. Testing of output is restricted to `System.out.print` and `System.out.println`. As with user input, there are many possible ways for directing the output of a program, for example to `System.out`, to a file, or to a text area in a graphical user interface. The AP Java subset includes the ability to print output to `System.out`, because it makes it easy to formulate questions. Since most graphical environments allow printing of debug messages to `System.out` (with output being collected in a special window, e.g. the "Java console" in a browser), students are usually familiar with this method of producing output. Formatted output (e.g., with `NumberFormat`) is not in the subset.

13. The `main` method and command-line arguments are not in the subset. Not all students are familiar with the use of the `main` method. Common alternate methods for invoking programs include applets, extending an instructor-provided framework, or using an environment such as "BlueJ." The AP Computer Science Exam does not prescribe any particular approach for program invocation. In free-response questions, students are not expected to invoke programs. In case studies, program invocation with `main` may occur, but the `main` method will be kept very simple.

14. Arrays: one-dimensional arrays and two-dimensional rectangular arrays are part of the AP Java subset. Both arrays of primitive types (e.g., `int[]` and arrays of objects (e.g., `Student[]`)are in the subset. Initialization of named arrays (`int[] a = { 1, 2, 3 };`) is part of the AP Java subset. Arrays with more than two dimensions (e.g., `rubik = new Color[3][3][3]`) are not in the subset. "Ragged" arrays (e.g., `new int[3][]`) are not in the subset. In particular, students do not need to know that an `int[3][3]` really is an "array of arrays" whose rows can be replaced with other `int[]` arrays. However, students are expected to know that `a[0].length` is the number of columns in a rectangular two-dimensional array a. Anonymous arrays (e.g., `new int[] { 1, 2, 3 }`) are not in the subset.

15. The control structures `if, if/else, while, for, return` are part of the AP Java subset. The `do/while, switch`, plain and labeled `break` and `continue` statements are not in the subset.

16. Method overloading (e.g. `MyClass.foo(String s)` and `MyClass.foo(int n)`) is part of the AP Java subset. Students should understand that the signature of a method depends on the number, types, and order of its parameters but does not include the return type of the method.

17. Classes: Students are expected to construct objects with the new operator, to supply construction parameters, and to invoke accessor and modifier methods. For the A exam, students are expected to modify existing classes (by adding or modifying methods and instance variables). For the AB exam, students are expected to design their own classes.

18. Visibility: In the AP Java subset, all classes are public. All instance variables are `private`. Methods, constructors, and constants (static final variables) are either `public` or `private`. The AP Java subset does not use `protected` and `package` (default) visibility.

19. The AP Java subset uses `/* */`, and `//` comments. Javadoc comments are not part of the subset.

20. The `final` keyword is only used for `final` block scope constants and `static final` class scope constants. `final` parameters or instance variables, `final` methods and `final` classes are not in the subset.

21. The concept of `static` methods is a part of the subset. Students are required to understand when the use of `static` methods is appropriate. In the exam, `static` methods are always invoked through a class, never an object (i.e., `ClassName.foo()`, not `obj.foo()`).

22. `static final` variables are part of the subset, other `static` variables are not.

23. The `null` reference is part of the AP Java subset.

24. The use of this is restricted to passing the implicit parameter in its entirety to another method (e.g., `obj.method(this)`) and to descriptions such as "the implicit parameter `this`". Using `this.var` or `this.method(args)` is not in the subset. In particular, students are not required to know the idiom "`this.var = var`", where var is both the name of an instance variable and a parameter variable. Calling other constructors from a constructor with the `this(args)` notation is not in the subset.

25. The use of `super` is restricted to invoking a superclass constructor (`super(args)`). Invoking a superclass method through the `super` keyword (i.e., `super.method(args)`) may be tested on the AB exam.

26. Students need to be able to implement constructors that initialize all instance variables. Class constants are initialized with an initializer:

    ```
    public static final MY_CONSTANT = initialization
    expression;
    ```

    The rules for default initialization (with 0, `false` or `null`) are not in the subset. Initializing instance variables with an initializer is not in the subset. Initialization blocks are not in the subset.

27. Students are expected to be able to extend classes and implement interfaces. On the A exam, students are expected to have a reading knowledge of inheritance that includes understanding the concepts of method overriding and polymorphism as well as the ability to modify existing subclasses. On the AB exam, students are expected to implement their own subclasses.

28. Students are expected to be able to read the definitions of interfaces and abstract classes and understand that the abstract methods need to be redefined. On the AB exam, students are expected to define their own interfaces and abstract classes.

29. Students are expected to understand the difference between object equality (`equals`) and identity (`==`). The implementation of `equals` and `hashCode` methods are not in the subset.

30. Cloning is not in the subset, because of the complexities of implementing the `clone` method correctly and the fact that `clone` is rarely required in Java programs.

31. The `finalize` method is not in the subset.

32. Students are expected to understand that conversion from a subclass reference to a superclass reference is legal and does not require a cast. Class casts (generally from Object to another class) are part of the AP Java subset, to enable the use of generic collections, for example, `Person p = (Person)people.get(i);` The `instanceof` operator is not in the subset. Array type compatibility and casts between array types are not in the subset.

33. Students are expected to have a basic understanding of packages and a reading knowledge of `import` statements of the form

    ```
    import packageName.subpackageName.ClassName;
    ```

    `import` statements with a trailing `*`, packages and methods for locating class files (e.g., through a class path) are not in the subset.

34. Inner classes are not in the subset.

35. Threads are not in the subset.

36. Students are expected to understand the exceptions that occur when their programs contain errors (in particular, `NullPointerException`, `ArrayIndexOutOfBoundsException`, `ArithmeticException`, `ClassCastException`). Students are expected to be able to throw the unchecked `IllegalStateException` and `NoSuchElementException` in their own methods (principally when implementing collection ADTs). Checked exceptions are not in the subset. In particular, the `try/catch/finally` statements and the `throws` modifier are not in the subset.

## Summary Table

| Tested in A, AB exam | Tested in AB exam only | Potentially relevant to CS1/CS2 course but not tested |
|---|---|---|
| `int, double, boolean` | | `short, long, byte, char, float` |
| `+ , -, *, /, %, ++, —` | | Using the values of `++`, – expressions in other expressions |
| `=, +=, -=, *=, /=, %=` | | |
| `==, !=, <, <=, >, >=` | | |
| `&&, \|\|, !` and short-circuit evaluation | | `<<, >>, >>>, &, ~, \|, ^, ?:` |
| `(int)`, `(double)` | | Other numeric casts such as `(char)` or `(float)` |
| String concatenation | | `StringBuffer` |

| Tested in A, AB exam | Tested in AB exam only | Potentially relevant to CS1/CS2 course but not tested |
|---|---|---|
| *(continued)* | | |
| Escape sequences `\" \\ \n` inside strings | | Other escape sequences (`\' \t \unnnn`) |
| `System.out.print`, `System.out.println` | | `System.in`, Stream input/output, GUI input/output, parsing input, formatted output |
| | | `public static void main (String args)` |
| 1-dimensional arrays, 2-dimensional rectangular arrays | | Arrays with 3 or more dimensions, ragged arrays |
| `if, if/else, while, for, return` | | `do/while, switch, break, continue` |
| Modify existing classes | Design classes | |
| public classes, private instance variables, public or private methods or constants | | `protected` or package visibility |
| | | `@param, @return, @precondition, @postcondition` |
| `final` local variables, static final class variables | | `final` parameter variables, instance variables, methods or classes |

| **Tested in A, AB exam** | **Tested in AB exam only** | **Potentially relevant to CS1/CS2 course but not tested** |
|---|---|---|
| *(continued)* | | |
| `static` methods | | `static non-final` variables |
| `null, this, super` | `super.method(args)` | `this.var, this.method(args), this(args)` |
| Constructors and initialization of static variables | | Default initialization of instance variables, initialization blocks |
| Understand inheritance hierarchies. Modify subclass implementations and implementations of interfaces. | Design and implement subclasses | |
| Understand the concepts of abstract classes and interfaces | Design and implement abstract classes and interfaces | |
| Understand equals, `==`, and `!=` comparison of objects | | `clone`, implementation of `equals` |
| `Comparable.compareTo` | | |
| Conversion to supertypes and (`Subtype`) casts | | `instanceof` |
| | | Inner classes |

| Tested in A, AB exam | Tested in AB exam only | Potentially relevant to CS1/CS2 course but not tested |
|---|---|---|
| *(continued)* | | |
| Package concept, `import x.y.Z;` | | `import x.y.*`, defining packages, class path |
| Exception concept, common exceptions, throwing standard unchecked exceptions | | Checked exceptions try/catch/ finally, throws |
| `Comparable, String, Math, Random, Object, ArrayList` | `List, Set, Map, Iterator, ListIterator, LinkedList, HashSet, TreeSet, HashMap, TreeMap` | |
| Wrapper classes (`Integer, Double`) | | |
| | | Sorting methods in `Arrays` and `Collections` |

# Appendix C

## Quick Reference – A Exam

**`class java.lang.Object`**

```
• boolean equals(Object other)
• String toString()
```

**`interface java.lang.Comparable`**

```
• int compareTo(Object other)
      // return value < 0 if this is less than other
      // return value = 0 if this is equal to other
      // return value > 0 if this is greater than other
```

**`class java.lang.Integer implements java.lang.Comparable`**

```
• Integer(int value)          // constructor
• int intValue()
• boolean equals(Object other)
• String toString()
• int compareTo(Object other)
      // specified by java.lang.Comparable
```

**`class java.lang.Double implements java.lang.Comparable`**

```
• Double(double value)        // constructor
• double doubleValue()
• boolean equals(Object other)
• String toString()
• int compareTo(Object other)
      // specified by java.lang.Comparable
```

**`class java.lang.String implements java.lang.Comparable`**

```
• int compareTo(Object other)
      // specified by java.lang.Comparable
• boolean equals(Object other)
• int length()
• String substring(int from, int to)
      // returns the substring beginning at from and ending at to-1
• String substring(int from)
      // returns substring (from, length())
• int indexOf(String s)
      // returns the index of the first occurrence of s;
      // returns -1 if not found
```

## Quick Reference – A Exam (continued)

**`class java.lang.Math`**
- `static int abs(int x)`
- `static double abs(double x)`
- `static double pow(double base, double exponent)`
- `static double sqrt(double x)`

**`class java.util.Random`**
- `int nextInt(int n)`
  `//` returns an integer in the range from 0 to `n-1` inclusive
- `double nextDouble()`

**`class java.util.ArrayList`**
- `int size()`
- `boolean add(Object x)`
- `Object get(int index)`
- `Object set(int index, Object x)`
  `//` replaces the element at `index` with `x`
  `//` returns the element formerly at specified position
- `void add(int index, Object x)`
  `//` inserts `x` at position `index`, sliding elements at
  `//` position `index` and higher to the right (adds 1 to
  `//` their indices) and adjusts size

- `Object remove(int index)`
  `//` removes element from position `index`, sliding elements
  `//` at position `index + 1` and higher to the left (subtracts 1
  `//` from their indices) and adjusts size

# Appendix D

## Quick Reference – AB Exam

**`class java.lang.Object`**
- `boolean equals(Object other)`
- `String toString()`
- `int hashcode()`

**`interface java.lang.Comparable`**
- `int compareTo(Object other)`
  - // return value < 0 if `this` is less than `other`
  - // return value = 0 if `this` is equal to `other`
  - // return value > 0 if `this` is greater than `other`

**`class java.lang.Integer implements java.lang.Comparable`**
- `Integer(int value)` // constructor
- `int intValue()`
- `boolean equals(Object other)`
- `String toString()`
- `int compareTo(Object other)`
  - // specified by `java.lang.Comparable`

**`class java.lang.Double implements java.lang.Comparable`**
- `Double(double value)` // constructor
- `double doubleValue()`
- `boolean equals(Object other)`
- `String toString()`
- `int compareTo(Object other)`
  - // specified by `java.lang.Comparable`

**`class java.lang.String implements java.lang.Comparable`**
- `int compareTo(Object other)`
  - // specified by `java.lang.Comparable`
- `boolean equals(Object other)`
- `int length()`
- `String substring(int from, int to)`
  - // returns the substring beginning at `from` and ending at `to-1`
- `String substring(int from)`
  - // returns `substring(from, length())`
- `int indexOf(String s)`
  - // returns the index of the first occurrence of `s`;
  - // returns `-1` if not found

## Quick Reference – AB Exam (continued)

**`class java.lang.Math`**
- `static int abs(int x)`
- `static double abs(double x)`
- `static double pow(double base, double exponent)`
- `static double sqrt(double x)`

**`class java.util.Random`**
- `int nextInt(int n)`
  - `//` returns an integer in the range from 0 to `n-1` inclusive
- `double nextDouble()`

**`interface java.util.List`**
- `boolean add(Object x)`
- `int size()`
- `Iterator iterator()`
- `ListIterator listIterator()`

**`class java.util.ArrayList implements java.util.List`**
- Methods in addition to the `List` methods:
- `Object get(int index)`
- `Object set(int index, Object x)`
  - `//` replaces the element at `index` with `x`
  - `//` returns the element formerly at specified position
- `void add(int index, Object x)`
  - `//` inserts `x` at position `index`, sliding elements at
  - `//` position `index` and higher to the right (adds 1 to
  - `//` their indices) and adjusts size

- `Object remove(int index)`
  - `//` removes element from position `index`, sliding elements
  - `//` at position `index + 1` and higher to the left (subtracts 1
  - `//` from their indices) and adjusts size

**`class java.util.LinkedList implements java.util.List`**
- Methods in addition to the `List` methods:
- `void addFirst(Object x)`
- `void addLast(Object x)`
- `Object getFirst()`
- `Object getLast()`
- `Object removeFirst()`
- `Object removeLast()`

## Quick Reference – AB Exam (continued)

**`interface java.util.Set`**
- `boolean add(Object x)`
- `boolean contains(Object x)`
- `boolean remove(Object x)`
- `int size()`
- `Iterator iterator()`

**`class java.util.HashSet implements java.util.Set`**
**`class java.util.TreeSet implements java.util.Set`**

**`interface java.util.Map`**
- `Object put(Object key, Object value)`
- `Object get(Object key)`
- `boolean containsKey(Object key)`
- `int size()`
- `Set keySet()`

**`class java.util.HashMap implements java.util.Map`**
**`class java.util.TreeMap implements java.util.Map`**

**`interface java.util.Iterator`**
- `boolean hasNext()`
- `Object next()`
- `void remove()`

**`interface java.util.ListIterator extends java.util.Iterator`**
- Methods in addition to the `Iterator` methods
- `void add(Object x)`
- `void set(Object x)`

## Quick Reference – AB Exam (continued)

### Implementation classes for linked list and tree nodes

```
public class ListNode
{
  public ListNode(Object initValue, ListNode initNext)
    { value = initValue; next = initNext; }

  public Object getValue() { return value; }
  public ListNode getNext() { return next; }

  public void setValue(Object theNewValue)
    { value = theNewValue; }
  public void setNext(ListNode theNewNext)
    { next = theNewNext; }

  private Object value;
  private ListNode next;
}
```

## Quick Reference – AB Exam (continued)

```
public class TreeNode
{
  public TreeNode(Object initValue)
    { value = initValue; left = null; right = null; }

  public TreeNode(Object initValue, TreeNode initLeft,
                  TreeNode initRight)
    { value = initValue; left = initLeft;
      right = initRight; }

  public Object getValue() { return value; }
  public TreeNode getLeft() { return left; }
  public TreeNode getRight() { return right; }

  public void setValue(Object theNewValue)
    { value = theNewValue; }
  public void setLeft(TreeNode theNewLeft)
    { left = theNewLeft; }
  public void setRight(TreeNode theNewRight)
    { right = theNewRight; }

  private Object value;
  private TreeNode left;
  private TreeNode right;
}
```

## Quick Reference – AB Exam (continued)

**Interface for stacks** (* See note at end of reference)

```
public interface Stack
{
  // postcondition: returns true if stack is empty;
  //                otherwise, returns false
  boolean isEmpty();

  // precondition:  stack is [e1, e2, ..., en]
  //                with n >= 0
  // postcondition: stack is [e1, e2, ..., en, x]
  void push(Object x);

  // precondition:  stack is [e1, e2, ..., en]
  //                with n >= 1
  // postcondition: stack is [e1, e2, ..., e(n-1)];
  //                returns en
  //                throws an unchecked exception if
  //                the stack is empty
  Object pop();

  // precondition:  stack is [e1, e2, ..., en]
  //                with n >= 1
  // postcondition: returns en
  //                throws an unchecked exception if
  //                the stack is empty
  Object peekTop();
}
```

## Quick Reference – AB Exam (continued)

**Interface for queues** (* See note at end of reference)

```
public interface Queue
{
  // postcondition: returns true if queue is empty;
  //                otherwise, returns false
  boolean isEmpty();

  // precondition:  queue is [e1, e2, ..., en]
  //                with n >= 0
  // postcondition: queue is [e1, e2, ..., en, x]
  void enqueue(Object x);

  // precondition:  queue is [e1, e2, ..., en]
  //                with n >= 1
  // postcondition: queue is [e2, ..., en]; returns e1
  //                throws an unchecked exception if
  //                the queue is empty
  Object dequeue();

  // precondition:  queue is [e1, e2, ..., en]
  //                with n >= 1
  // postcondition: returns e1
  //                throws an unchecked exception if
  //                the queue is empty
  Object peekFront();
}
```

## Quick Reference – AB Exam (continued)

**Interface for priority queues** (* See note at end of reference)

```
public interface PriorityQueue
{
  // postcondition: returns true if the number of
  //                elements in the priority queue
  //                is 0; otherwise, returns false
boolean isEmpty();

  // postcondition: x has been added to the priority
  //                queue; the number of elements in
  //                the priority queue is increased
  //                by 1.
  void add(Object x);

  // postcondition: The smallest item in the priority
  //                queue is removed and returned; the
  //                number of elements in the priority
  //                queue is decreased by 1.
  //                throws unchecked exception if
  //                priority queue is empty
  Object removeMin();

  // postcondition: The smallest item in the priority
  //                queue is returned; the priority
  //                queue is unchanged throws
  //                unchecked exception if priority
  //                queue is empty
  Object peekMin();
}
```

*** Note regarding use of stacks, queues, and priority queues**

When a stack, queue, or priority queue object needs to be instantiated, code such as the following is used:

```
Queue q = new ListQueue();
              // ListQueue implements Queue
```

# AP Program Essentials

## The AP Reading

In June, the free-response sections of the exams, as well as the Studio Art portfolios, are scored by college faculty and secondary school AP teachers at the AP Reading. Thousands of these readers participate, under the direction of a chief reader in each field. The experience offers both significant professional development and the opportunity to network with like-minded educators; if you are an AP teacher or a member of a college faculty and would like to serve as a reader, you can apply online at apcentral.collegeboard.com/reader. Alternatively, send an e-mail message to apreader@ets.org, or call Performance Scoring Services at 609 406-5383.

### AP Grades

The readers' scores on the essay and problem-solving questions are combined with the results of the computer-scored multiple-choice questions, and the total raw scores are converted to AP's 5-point scale:

| AP GRADE | QUALIFICATION |
|---|---|
| 5 | Extremely Well Qualified |
| 4 | Well Qualified |
| 3 | Qualified |
| 2 | Possibly Qualified |
| 1 | No Recommendation |

### Grade Distributions

Many teachers want to compare their students' grades with the national percentiles. Grade distribution charts are available at AP Central, as is information on how the cut-off points for each AP grade are calculated.

## AP and College Credit

Advanced standing and/or credit is awarded by the college or university, not the College Board or the AP Program. The best source of specific and up-to-date information about an individual institution's policy is its catalog or Web site.

## Why Colleges Give Credit for AP Grades

Colleges need to know that the AP grades they receive for their incoming students represent a level of achievement equivalent to that of students who take the same course in the colleges' own classrooms. That equivalency is assured through several Advanced Placement Program processes:

- College faculty serve on the committees that develop the course descriptions and examinations in each AP subject.
- College faculty are responsible for standard setting and are involved in the evaluation of student responses at the AP Reading.
- AP courses and exams are updated regularly, based on both the results of curriculum surveys at up to 200 colleges and universities and the interactions of committee members with professional organizations in their discipline.
- College comparability studies are undertaken in which the performance of college students on AP Exams is compared with that of AP students to confirm that the AP grade scale of 1–5 is properly aligned with current college standards.

In addition, the College Board has commissioned studies that use a "bottom-line" approach to validating AP Exam grades by comparing the achievement of AP versus non-AP students in higher-level college courses. For example, in the 1998 Morgan and Ramist "21-College" study, AP students who were exempted from introductory courses and who completed a higher-level course in college are compared, on the basis of their college grades, with students who completed the prerequisite first course in college, then took the second, higher-level course in the subject area. Such studies answer the question of greatest concern to colleges — are their AP students who are exempted from introductory courses as well prepared to continue in a subject area as students who took their first course in college? To see the results of several college validity studies, go to AP Central. (The aforementioned Morgan and Ramist study can be downloaded from the site in its entirety.)

## Guidelines on Granting Credit for AP Grades

If you are an admission administrator and need guidance on setting a policy for your college, you will find the *College and University Guide to the Advanced Placement Program* useful; see the back of this booklet for ordering information. Alternatively, contact your local College Board Regional Office, as noted on the inside back cover of this Course Description.

### Finding Colleges That Accept AP Grades

In addition to contacting colleges directly for their AP policies, students and teachers can use College Search, an online resource maintained by the College Board through its Annual Survey of Colleges. College Search can be accessed via the College Board's Web site (www.collegeboard.com). It is worth remembering, though, that policies are subject to change. Contact the college directly to get the most up-to-date information.

## AP Awards

The AP Program offers a number of awards to recognize high school students who have demonstrated college-level achievement through AP courses and exams. Although there is no monetary award, in addition to an award certificate, student achievement is acknowledged on any grade report sent to colleges following the announcement of the awards.

For detailed information on AP Awards, including qualification criteria, visit AP Central or contact the College Board's National Office. Students' questions are also answered in the *AP Bulletin for Students and Parents;* information about ordering and downloading the *Bulletin* can be found at the back of this Course Description.

## AP Calendar

To get an idea of the various events associated with running an AP program and administering the AP Exams, see the *AP Program Guide*; information about ordering and downloading the *Guide* can be found at the back of this booklet.

## Test Security

The entire AP Exam must be kept secure at all times. Forty-eight hours after the exam has been administered, the green and blue inserts containing the free-response questions (Section II) can be made available for teacher and student review.* **However, the multiple-choice section (Section I) MUST remain secure both before and after the exam administration.** No one other than students taking the exam can ever have access to or see the questions contained in Section I — this includes AP Coordinators and all teachers. The multiple-choice section must never be shared or copied in any manner.

*The alternate form of the free-response section (used for late testing) is NOT released.

Selected multiple-choice questions are reused from year to year to provide an essential method of establishing high exam reliability, controlled levels of difficulty, and comparability with earlier exams. These goals can be attained only when the multiple-choice questions remain secure. This is why teachers cannot view the questions and students cannot share information about these questions with anyone following the exam administration.

To ensure that all students have an equal chance to demonstrate their abilities on the exam, AP Exams must be administered in a uniform manner. **It is extremely important to follow the administration schedule and all procedures outlined in detail in the most recent *AP Coordinator's Manual.*** The manual also includes directions on how to deal with misconduct and other security problems. Any breach of security should be reported to ETS Test Security (call 800 353-8570, fax 609 406-9709, or e-mail tsreturns@ets.org).

## Teacher Support

Look for these enhanced Web resources at AP Central:

- Teachers' Resources and Institutions and Workshops.
- The most up-to-date and comprehensive information on AP courses, exams, and other Program resources.
- The opportunity to exchange teaching methods and materials with the international AP community.
- An electronic library of AP publications, including released exam questions, the *AP Coordinator's Manual*, *Course Descriptions*, and sample syllabi.
- Opportunities for professional involvement in the AP Program.
- Information about state and federal support for the AP Program.
- AP Program data, research, and statistics.
- FAQs about the AP Program.
- Current news and information in education.

To supplement these online resources, there are a number of AP publications, CD-ROMs, and videos that can assist AP teachers. Please see the following pages for an overview and ordering information.

## Pre-AP™

### Preparing Students for Challenging Courses; Preparing Teachers for Student Success

Many students reach high school without learning the skills and concepts necessary to succeed in demanding courses. To address this issue, the College Board has developed and implemented Pre-AP Initiatives, which help middle school and high school teachers bring out their students' potential and contribute to their future success. Pre-AP Initiatives provide strategies for introducing essential skills and concepts before students' junior and senior years of high school.

Pre-AP is not a course or prescribed curriculum. Instead, it consists of two teacher professional development workshops: *Building Success* and *Setting the Cornerstones: Building the Foundation of AP Vertical Teams*™. The workshops help teachers build a demanding curriculum and support the creation of teams of middle school and high school teachers that work together to prepare students for AP and other courses. In 2001, more than 12,700 teachers attended Pre-AP conferences and workshops, a 55 percent increase from the previous year.

### Building Success

*Building Success* is a two-day workshop designed to assist English and history teachers in grades seven and above, providing these teachers with a series of techniques and methods for teaching the reading, writing, and communication skills that are necessary for advanced work. Participants learn the SOAPS (Speaker, Occasion, Audience, Purpose, Subject) technique for critical reading and analytical writing. Additionally, they develop strategies that encourage students to ask questions, draw inferences, and construct good verbal and written arguments.

### Setting the Cornerstones Workshops: Building the Foundation of AP Vertical Teams™

An AP Vertical Team is made up of teachers from different grade levels who work together to develop and implement a curriculum that gradually introduces key concepts and skills in a particular discipline, starting in middle school. The team's goal is to help students acquire the skills necessary for success in AP and other rigorous courses.

*Setting the Cornerstones: Building the Foundation of AP Vertical Teams* is a two-day workshop for district and campus administrators, curriculum coordinators, counselors, department leaders, and groups of teachers interested in forming teams to improve student performance and participation in the AP Program. For more information, contact your College Board Regional Office.

## AP Publications and Other Resources

A number of AP publications, CD-ROMs, and videos are available to help students, parents, AP Coordinators, and high school and college faculty learn more about the AP Program and its courses and exams. To identify resources that may be of particular use to you, refer to the following key.

| | |
|---|---|
| **AP Coordinators and Administrators** | **A** |
| **College Faculty** | **C** |
| **Students and Parents** | **SP** |
| **Teachers** | **T** |

### Ordering Information

You have several options for ordering publications:

- **Online.** Visit the College Board Store (http://store.collegeboard.com) to see descriptions and pictures of AP publications and to place your order.
- **By mail.** Send a completed order form with your payment or credit card information to: Advanced Placement Program, Dept. E-06, P.O. Box 6670, Princeton, NJ 08541-6670. If you need a copy of the order form, you can download one from AP Central.
- **By fax.** Credit card orders can be faxed to AP Order Services at 609 771-7385.
- **By phone.** Call AP Order Services at 609 771-7243, Monday through Friday 8:00 a.m. to 9:00 p.m. ET. Have your American Express, Discover, JCB, MasterCard, or VISA information ready. This phone number is for credit card orders only.

Payment must accompany all orders not on an institutional purchase order or credit card, and checks should be made payable to the College Board. The College Board pays UPS ground rate postage (or its equivalent) on all prepaid orders; delivery generally takes two to three weeks. Please do not use P.O. Box numbers. Postage will be charged on all orders requiring billing and/or requesting a faster method of delivery.

Publications may be returned for a full refund if they are returned within 30 days of invoice. Software and videos may be exchanged within 30 days if they are opened, or returned for a full refund if they are unopened. No collect or C.O.D. shipments are accepted. Unless otherwise specified, **orders will be filled with the currently available edition;** prices and discounts are subject to change without notice.

In compliance with Canadian law, all AP publications delivered to Canada incur the 7 percent GST. The GST registration number is 13141 4468 RT. Some Canadian schools are exempt from paying the GST. Appropriate proof of exemption must be provided when AP publications are ordered so that tax is not applied to the billing statement.

## Print

Items marked with a computer mouse icon can be downloaded for free from AP Central.

**AP Bulletin for Students and Parents: Free** **SP**

This bulletin provides a general description of the AP Program, including policies and procedures for preparing to take the exams, and registering for the AP courses. It describes each AP Exam, lists the advantages of taking the exams, describes the grade reporting and award options available to students, and includes the upcoming exam schedule. Free copies of this bulletin for all AP students are mailed to a school after it registers to participate in the fall.

**AP Program Guide: Free** **A**

This guide takes the AP Coordinator step-by-step through the school year — from organizing an AP program, through ordering and administering the AP Exams, payment, and grade reporting. It also includes information on teacher professional development, AP resources, and exam schedules. The *AP Program Guide* is sent automatically to all schools that register to participate in AP.

**College and University Guide to the AP Program: $10** **C, A**

This guide is intended to help college and university faculty and administrators understand the benefits of having a coherent, equitable AP policy. Topics included are validity of AP grades; developing and maintaining scoring standards; ensuring equivalent achievement; state legislation supporting AP; and quantitative profiles of AP students by each AP subject.

**Course Descriptions: $15 or a free download from AP Central** **SP, T, A, C**

Course Descriptions provide an outline of the AP course content, explain the kinds of skills students are expected to demonstrate in the corresponding introductory college-level course, and describe the AP Exam. They also provide sample multiple-choice questions with an answer key, as well as sample free-response questions. A complete set of Course Descriptions is available for $125. Note: The Course Description for AP Computer Science is available in electronic format only.

**Pre-AP: Achieving Equity, Emphasizing Excellence: Free** **A, T**

An informational brochure describing the Pre-AP concept and outlining the characteristics of a successful Pre-AP program.

**Released Exams: $25 ($35 for "double" subjects: Calculus, Computer Science, Latin, Physics)** **T**

About every four years, on a rotating schedule, the AP Program releases a complete copy of each exam. In addition to providing the multiple-choice questions and answers, the publication describes the process of scoring the free-response questions and includes examples of students' actual responses, the scoring standards, and commentary that explains why the responses received the scores they did.

*Packets of 10: $35.* For each subject with a released exam, you can purchase a packet of 10 copies of that year's exam for use in your classroom (e.g., to simulate an AP Exam administration).

**Teacher's Guides: $17** **T**

For those about to teach an AP course for the first time, or for experienced AP teachers who would like to get some fresh ideas for the classroom, the Teacher's Guide is an excellent resource. Each Teacher's Guide contains syllabi developed by high school teachers currently teaching the AP course and college faculty who teach the equivalent course at colleges and universities. Along with detailed course outlines and innovative teaching tips, you'll also find extensive lists of recommended teaching resources.

**AP Vertical Team Guides** **T, A**

An AP Vertical Team (APVT) is made up of teachers from different grade levels who work together to develop and implement a sequential curriculum in a given discipline. The team's goal is to help students acquire the skills necessary for success in AP. To help teachers and administrators who are interested in establishing an APVT at their school, the College Board has published four guides: *AP Vertical Teams in Science*, *Social Studies*, *Foreign Language*, *Studio Art*, *and Music Theory: An Introduction* ($10)*;* *A Guide for Advanced Placement English Vertical Teams* ($10); *Advanced Placement Program Mathematics Vertical Teams Toolkit* ($35); and *the AP Vertical Teams Guide for Social Studies* ($25). A discussion of the English Vertical Teams guide, and the APVT concept, is also available on a 15-minute VHS videotape ($10).

## Multimedia

**APCD®: $49 (home version),**
**$450 (multi-network site license)** **SP, T**

These CD-ROMs are available for Calculus AB, English Language, English Literature, European History, Spanish Language, and U.S. History. They each include actual AP Exams, interactive tutorials, and other features including exam descriptions, answers to frequently asked questions, study-skill suggestions, and test-taking strategies. There is also a listing of resources for further study and a planner to help students schedule and organize their study time.

The teacher version of each CD, which can be licensed for up to 50 workstations, enables you to monitor student progress and provide individual feedback. Included is a Teacher's Manual that gives full explanations along with suggestions for utilizing the APCD® in the classroom.

**Videoconference Tapes: $15** **T, C**

AP has conducted live, interactive videoconferences for various subjects, enabling AP teachers and students to talk directly with the Development Committees that design and develop the AP courses and exams. Tapes of these events are available in VHS format and are approximately 90 minutes long.

## College Board Regional Offices

**National Office**
45 Columbus Avenue, New York, NY 10023-6992
212 713-8066
E-mail: ap@collegeboard.org

**Middle States**
Serving Delaware, District of Columbia, Maryland, New Jersey, New York, Pennsylvania, Puerto Rico, and U.S. Virgin Islands
2 Bala Plaza, Suite 900, Bala Cynwyd, PA 19004-1501
610 667-4400
E-mail: msro@collegeboard.org

**Midwestern**
Serving Illinois, Indiana, Iowa, Kansas, Michigan, Minnesota, Missouri, Nebraska, North Dakota, Ohio, South Dakota, West Virginia, and Wisconsin
1560 Sherman Avenue, Suite 1001, Evanston, IL 60201-4805
847 866-1700
E-mail: mro@collegeboard.org

**New England**
Serving Connecticut, Maine, Massachusetts, New Hampshire, Rhode Island, and Vermont
470 Totten Pond Road, Waltham, MA 02451-1982
781 890-9150
E-mail: nero@collegeboard.org

**Southern**
Serving Alabama, Florida, Georgia, Kentucky, Louisiana, Mississippi, North Carolina, South Carolina, Tennessee, and Virginia
100 Crescent Centre Parkway, Suite 340, Tucker, GA 30084-7039
770 908-9737
E-mail: sro@collegeboard.org

**Southwestern**
Serving Arkansas, New Mexico, Oklahoma, and Texas
4330 South MoPac Expressway, Suite 200, Austin, TX 78735-6734
512 891-8400
E-mail: swro@collegeboard.org

**Dallas/Fort Worth Metroplex AP Office**
P.O. Box 19666, 600 South West Street, Suite 108, Arlington, TX 76019
817 272-7200
E-mail: kwilson@collegeboard.org

**Western**
Serving Alaska, Arizona, California, Colorado, Hawaii, Idaho, Montana, Nevada, Oregon, Utah, Washington, and Wyoming
2099 Gateway Place, Suite 480, San Jose, CA 95110-1048
408 452-1400
E-mail: wro@collegeboard.org

**Canada**
1708 Dolphin Avenue, Suite 406, Kelowna, BC, Canada V1Y 9S4
250 861-9050; 800 667-4548 in Canada only
E-mail: gewonus@ap.ca

**International**
Serving all countries outside the United States and Canada
45 Columbus Avenue, New York, NY 10023-6992
212 713-8091
E-mail: apintl@collegeboard.org

*2003 Exam Date: Wednesday, May 7, morning session*

## 2001-02 Development Committee and Chief Reader

**apcentral.collegeboard.com**

**I.N. 994586**