RELEASED EXAMS

1999

AP® Computer Science A and Computer Science AB

CONTAINS:

- **Multiple-Choice Questions and Answer Key**
- **Free-Response Questions, Scoring Guidelines, and Sample Student Responses and Commentary**
- **Statistical Information About Student Performance on the 1999 Exams**

Advanced Placement Program®

The 1999 AP® Examinations in Computer Science A and Computer Science AB

Contains:

- Multiple-Choice Questions and Answer Keys
- Free-Response Questions, Scoring Guidelines, and Sample Student Responses and Commentary
- Statistical Information about Student Performance on the 1999 Exams

These test materials are intended for use by AP® teachers for course and exam preparation in the classroom. Teachers may reproduce them, in whole or in part, for limited use with their students, but may not mass distribute the materials, electronically or otherwise. These materials and any copies made of them may not be resold, and the copyright notices must be retained as they appear here. This permission does not apply to any third-party copyrights contained in the materials.

The College Board is a national nonprofit membership association dedicated to preparing, inspiring, and connecting students to college and opportunity. Founded in 1900, the association is composed of more than 3,800 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three million students and their parents, 22,000 high schools, and 5,000 colleges, through major programs and services in college admission, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT®, the Advanced Placement Program® (AP®) and, Pacesetter®. The College Board is committed to the principles of equity and excellence, and that commitment is embodied in all of its programs, services, activities, and concerns.

Copyright © 2000 by College Entrance Examination Board and Educational Testing Service. All rights reserved.

College Board, Advanced Placement Program, AP, College Board Online, College Explorer, and the acorn logo are registered trademarks of the College Entrance Examination Board. APCD, EssayPrep, and Pre-AP are trademarks of the College Entrance Examination Board.

THE COLLEGE BOARD: EDUCATIONAL EXCELLENCE FOR ALL STUDENTS

Chapter I The AP Process

- Who Develops the AP Computer Science Exams?
- How Are the Exams Developed?
  - Section I
  - Section II
- Question Types
  - Multiple Choice
  - Free Response
- Scoring the Exams
  - Who Scores the AP Computer Science Exams?
  - Ensuring Accuracy
  - How the Scoring Guidelines Are Created
  - Training Faculty Consultants to Apply the Scoring Guidelines
  - Maintaining the Scoring Guidelines
- Preparing Students for the Exams
- Teacher Support

This chapter will give you a brief overview of what goes on behind the scenes during the development and grading of the AP Computer Science A and AB Exams. You can find more detailed information in the "Technical Corner" of the AP website (www.collegeboard.org/ap).

Who Develops the AP Computer Science Exams?

The AP Computer Science Development Committee, working with content experts at Educational Testing Service (ETS), is responsible for creating the exams. This committee is made up of six teachers from secondary schools, colleges, and universities in different parts of the United States. The members provide different perspectives: AP high school teachers offer valuable advice regarding realistic expectations when matters of content coverage, skills required, and clarity of phrasing are addressed. On the other hand, college and university faculty members ensure that the questions are at the appropriate level of difficulty for an introductory college course in computer science. Each member typically serves for three to four years.

Another person who aids in the development process is the Chief Faculty Consultant (CFC). He or she attends every committee meeting to ensure that the free-response questions selected for the exam can be scored reliably. You can find out more about the role of the CFC, and the scoring process in general, on pages 2-4.

How Are the Exams Developed?

It takes at least two years to develop each AP Computer Science Exam. The development process is different for multiple-choice and free-response sections:

Section I

1. Each committee member independently writes a selection of multiple-choice questions based on the course content outline.
2. The committee convenes to review these draft questions, and eliminates any language, symbols, or content that may be offensive to major subgroups of the test-taking population. In addition, statistical procedures help the committee identify possibly unfair items.
3. Most of the multiple-choice questions are pretested in college classes to obtain some estimate of each question's level of difficulty.
4. The questions that make it through these screening processes are assembled according to test specifications developed by the committee and, after further editing and checking, comprise Section I of the AP Computer Science A and Computer Science AB Exams.

The committee controls the level of difficulty of the multiple-choice section by including a variety of questions at different levels of difficulty.

Section II

1. Individual committee members write a selection of free-response questions based on the course content outline.
2. The committee reviews and refines draft questions, and determines which ones will work well for the AP Exams. They consider, for example, whether the questions will offer an appropriate level of difficulty and whether they will elicit answers that allow faculty consultants to discriminate among the responses along a particular scoring scale. An ideal question enables the stronger students to demonstrate their accomplishments while revealing the limitations of less advanced students.

In the last stage of development, committee members give approval to a final draft of all multiple-choice and free-response questions. This review takes place several months before the administration of the exams.

Question Types

Each of the 1999 AP Examinations in Computer Science contains a 75-minute multiple-choice section and a 105-minute free-response section. The two sections are designed to complement each other and to meet the overall course objectives and exam specifications.

Multiple-choice questions are useful for measuring the breadth of content in the curriculum. In addition, they have three other strengths:

1. They are highly reliable. Reliability, or the likelihood that candidates of similar ability levels taking a different form of the exam will receive the same scores, is controlled more effectively with multiple-choice questions than with free-response questions.
2. They allow the Development Committee to include a selection of questions at various levels of difficulty, thereby ensuring that the measurement of differences in students' achievement is optimized. For AP Exams, the most important distinctions are between students earning the grades of 2 and 3, and 3 and 4. These distinctions are usually best accomplished by using many questions of middle difficulty.
3. They allow the CFC to compare the ability level of the current candidates with those from another year. A number of questions from an earlier exam are included in the current one, thereby allowing comparisons to be made between the scores of the earlier group of candidates and the current group. This information, along with other data, is used by the CFC to establish AP grades that reflect the competence demanded by the Advanced Placement Program, and that compare with earlier grades.

Free-response questions on the AP Computer Science Exams require students to use their analytical and organizational skills to reason about and write program fragments that fit the specifications given. They allow students to demonstrate their understanding of program structure and their ability to translate that understanding into a concrete solution. The free-response format allows for the presentation of uncommon yet correct responses and permits students to demonstrate their mastery of computer science by a show of creativity.

Free-response and multiple-choice questions are analyzed both individually and collectively after each administration, and the conclusions are used to improve the following year's exams.

Scoring the Exams

Who Scores the AP Computer Science Exams?

The people who score the free-response section of the AP Computer Science Exams are known as "faculty consultants." These faculty consultants are experienced computer science instructors who either teach one of the AP courses in a high school, or the equivalent courses at a college or university. Great care is taken to get a broad and balanced group of teachers. Among the factors considered before appointing someone to the role are school locale and setting (urban, rural, etc.), gender, ethnicity, and years of teaching experience. If you are interested in applying to be a faculty consultant at a future AP Reading, you can complete and submit an online application in the "Teachers" section of the AP website (www.collegeboard.org/ap), or request a printed application by calling (609) 406-5384.

In mid-June 1999, more than 100 teachers of computer science, about half from colleges and half from high schools, gathered at Clemson University, South Carolina, to score the free-response portions of the AP Computer Science Examinations. Leadership for the AP Computer Science Reading consisted of two exam leaders (one for Computer Science A and one for Computer Science AB), twelve question leaders (two per question), and an additional two table leaders for the overlap questions that appeared on both the A and the AB exams. The faculty consultants were divided into six teams of varying sizes, with each team responsible for grading one question. Under the guidance of the Chief Faculty Consultant and the Chief Faculty Consultant-Designate, the question leaders had responsibility for organizing the details of the Reading and conveying information to the faculty consultants in the respective teams.

Ensuring Accuracy

The primary goal of the scoring process is to have each faculty consultant score his or her set of papers fairly, uniformly, and to the same standard as the other faculty consultants. This is achieved through the creation of detailed scoring guidelines, the thorough training of all faculty consultants, and various "checks and balances" applied throughout the AP Reading.

How the Scoring Guidelines Are Created

1. Before the AP Reading, the CFC prepares a draft of the scoring guidelines for each free-response question. In the case of Computer Science, a 10-point scale (0-9) is used. A score of 0 means the student received no credit for the problem.
2. The CFC, question leaders, exam leaders, and ETS content experts meet at the Reading site a few days before the Reading begins. They discuss, review, and revise the draft scoring guidelines, and test them by pregrading randomly selected student papers. If problems or ambiguities become apparent, the scoring guidelines are revised and refined until a final consensus is reached.
3. The CFC, question leaders, and table leaders conduct training sessions for each free-response question, which are attended by all the faculty consultants who are scoring that question.

Training Faculty Consultants to Apply the Scoring Guidelines

Since the training of the faculty consultants is so vital in ensuring that students receive a grade that accurately reflects their performance, the process is thorough:

1. The faculty consultants read sample papers that have been pregraded (see above). These samples reflect all levels of ability.
2. Each group of faculty consultants then compares and discusses the scores for the samples, based on the scoring guidelines.
3. Once the faculty consultants as a group can apply the standards consistently and without disagreement, they begin reading in teams of two. Each team member scores half of a packet of 25 papers and then exchanges the examinations with his or her partner for a second reading. Scores and differences in judgment are discussed until agreement is reached, with the question leaders, the table leaders, or the CFC acting as arbitrators when needed.
4. After a team shows consistent agreement on its scores, its members proceed to score individually. Faculty consultants are encouraged to seek advice from each other, the question leaders and table leaders, or the CFC when in doubt about a score. A student response that is problematic receives multiple readings and evaluations.

Maintaining the Scoring Guidelines

A potential problem is that a faculty consultant could give an answer a higher or lower score than it deserves because the same student has performed well or poorly on other questions. The following steps are taken to prevent this so-called "halo effect:"

- Each question is read by a different faculty consultant;
- All scores given by other faculty consultants are completely masked; and
- The candidate's identification information is covered. Using these practices permits each faculty consultant to evaluate free-response answers without being prejudiced by knowledge about individual candidates.

Here are some other methods that help ensure that everyone is adhering closely to the scoring guidelines:

- The entire group discusses pregraded papers each morning, and as necessary during the day.
- Faculty consultants are paired, so that everyone has a partner to check consistency and to discuss problem cases with; question leaders are also paired up to help each other on questionable calls.
- Question leaders re-read (back read) a portion of the student papers from each member of his or her team. This approach allows each leader to guide the faculty consultants toward appropriate and consistent interpretations of the rubrics.
- The CFC and the question leaders monitor use of the full range of the scoring scale for the group and for each faculty consultant by checking daily graphs of score distributions.

Preparing Students for the Exams

The AP Computer Science courses (Computer Science A and Computer Science AB) are designed to be comparable to typical introductory computer science courses taught in a college or university department of computer science. 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 topic outline in the AP Computer Science Course Description.

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.

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.

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

- Students should be able to design and implement computer-based solutions to problems in several application areas.
- Students should learn well-known algorithms and data structures.
- Students should be able to develop and select appropriate algorithms and data structures to solve problems.
- 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.
- 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 case studies.)
- 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.
- Students should be able to recognize the ethical and social implications of computer use.

As teachers focus on the above goals, they will not only be preparing their students for the AP Computer Science Examinations; they will be preparing them for future study and applications of computer science.

Teacher Support

There are a number of resources available to help teachers prepare their students — and themselves — for the AP courses and exams.

AP workshops and summer institutes. New and experienced teachers are invited to attend workshops and seminars to learn the rudiments of teaching an AP course as well as the latest in each course's

expectations. Sessions of one day to several weeks in length are held year-round. Dates, locations, topics, and fees are available from the College Board's Regional Offices (see the inside front cover of this booklet), in the publication *Graduate Summer Courses and Institutes,* or in the "Teachers" section of our website (see below).

AP's corner of College Board Online®. You can supplement your AP course and preparation for the exam with plentiful advice and resources from our AP web pages (www.collegeboard.org/ap).

Online discussion groups. The AP Program has developed an interactive online mailing list for each AP subject. Many AP teachers find this free resource to be an invaluable tool for sharing ideas with colleagues on syllabi, course texts, teaching techniques, and so on, and for discussing other AP issues and topics as they arise. To find out how to subscribe, go to the "Teachers" section of our website.

AP publications and videos. See the Appendix for descriptions of a variety of useful materials for teachers. Of particular interest is the publication that complements this *Released Exam* – the *Packet of 10.* Teachers can use these multiple copies of the 1999 AP Computer Science Exam, which come with blank answer sheets, to simulate a national administration in their classroom.

AP videoconferences. Several videoconferences are held each year so that AP teachers can converse electronically with the high school and college teachers who develop AP courses and exams. Schools that participate in the AP Program are notified of the time, date, and subject of the videoconference in advance. Or, you can contact your Regional Office for more information. Videotapes of each conference are available shortly after the event; see the back of this book for ordering information.

Chapter II The 1999 AP Computer Science Examinations

- Exam Content and Format
- Purpose of the Exams
- Giving a Practice Exam
- Blank Answer Sheet
- Instructions for Administering the Exams
- The Exams

Exam Content and Format

The AP Computer Science A and Computer Science AB Examinations seek to assess how well a student has mastered the concepts and techniques of the subject matter of the corresponding college-level computer science courses. Each examination consists of a 40-question multiple-choice section and a four-question free-response section; the sections each contribute 50% to the composite score. Knowledge of the *AP Computer Science Large Integer Case Study* was required for the 1999 AP Computer Science A and Computer Science AB Examinations. Five of the 40 multiple-choice questions and one of the four free-response questions were based on the case study material. Two of the free-response questions were common to both exams.

The free-response questions covered a variety of programming techniques. A brief description of the problems is given here; more details are available in Chapter III.

- *Question A1.* A "standard" one-dimensional array question, this question measures the student's facility with one of the most basic data structures, an array (apvector) of records (structs).
- *Question A2.* This question, a harder one-dimensional array question, asked the students to reason about an ordered array of strings. Students needed to both understand the string abstraction and develop correct algorithms for manipulating the ordered array.
- *Question A3/AB2.* This question involved reasoning about the *AP Computer Science Large Integer Case Study.* In particular, students were required to add a member function to the BigInt class as well as implement division.
- *Question A4/AB1.* This question was substantively different from what would have been asked in Pascal. Students were provided a context for a class, given the declaration of the class, and asked to implement a constructor and two member functions for the class.
- *Question AB3.* This question examined the student's facility with a basic dynamically allocated data structure, the linked list, and was very similar to, and somewhat easier than, linked list questions asked in the past.
- *Question AB4.* This question examined the student's facility with binary trees, the other common dynamically allocated data structure.

The scoring guidelines for the free-response questions, and sample student responses, can be found in Chapter III.

Purpose of the Exams

Both the AP Computer Science A and Computer Science AB courses and examinations represent college-level computer science for which many colleges and universities grant advanced placement, college credit, or both. Appropriate credit and placement are granted by each institution in accordance with local policies. Many colleges provide statements regarding their AP policies in their catalogs. The immediate benefits available to students taking the AP Examination can range from tuition savings to more flexibility in fulfilling requirements for a degree. Students who have completed an introductory college-level computer science course in secondary school are also able to pursue more advanced courses sooner in their college career and, ultimately, to take more courses in computer science than they otherwise might. AP Computer Science is a good option for any student, but particularly for a student who plans further study in a quantitative field.

Giving a Practice Exam

The following pages contain the instructions, as printed in the 1999 *Coordinator's Manual,* for administering the AP Computer Science Exams. Following these instructions are copies of the 1999 answer sheet and the entire AP Computer Science A and AP Computer Science AB Exams. If you plan to use this released exam material to test your students, you may wish to use the instructions to create an exam situation that closely resembles a national administration. If so, read only the directions in the boxes to the students; all other instructions are for the person administering the test and need not be read aloud. Some instructions, such as those referring to the date, the time, and page numbers, are no longer relevant; please ignore them. There are also page number references in the exams themselves that refer to page numbers in the actual exam booklets. These do not match the page numbers in this book.

Another publication that you might find useful is the so-called *Packet of 10.* It is just that: packets of ten of the 1999 AP Computer Science A and Computer Science AB Exams, each with a blank answer sheet. Information on ordering AP publications is available at the back of this book.

Important

For regular administrations, read ALL of the boxed instructions below ***except*** for the box marked for administrations using an alternate form of the exam.

For administrations using an alternate form of the exam, read ALL of the boxed instructions below ***except*** for those marked specifically for the May 11th administration. If these instructions are being used for a late administration, all days, dates, and times to be read aloud should be adjusted as necessary.

The Computer Science A Examination and the Computer Science AB Examination should be administered simultaneously.

The administration of this exam includes survey questions. The time allowed for these survey questions is in addition to the actual test-taking time.

Complete the general instructions beginning on page 34. Then say:

It is Tuesday afternoon, May 11, and you will be taking one of the AP Computer Science Examinations. Check to make sure you have the correct examination: Computer Science A OR Computer Science AB. If you do not, please raise your hand. Print your name, last name first, on the front cover of the unsealed Section I booklet and read the directions on the back of the booklet. When you have finished, look up. . . .

Work only on Section I until time is called. Do not open the Section II package until you are told to do so. Remember, when you come to the end of the multiple-choice questions, there will be answer ovals left on your answer sheet. Scratch paper is not allowed, but you may use the designated space in the Section I booklet. Only No. 2 pencils may be used to mark your answers in Section I. Are there any questions?

Answer all questions regarding procedure.

When you are ready to begin the exam, note the time here __________. Then say:

The Large Integer Case Study is needed for questions 16 through 20. It begins on page 45 for Computer Science A and page 51 for Computer Science AB.

Open your Section I booklet and begin. You have 1 hour and 15 minutes for this section of the exam.

Allow 1 hour and 15 minutes. Note the time you will stop here __________. While candidates are working on Section I, you and your proctors should make sure they are marking answers on their answer sheets in pencil and are not looking at their Section II booklets.

After 1 hour and 15 minutes, say:

AT THE MAY 11TH ADMINISTRATION ONLY, SAY:

Stop working. Turn to page 40 of the Section I booklet for Computer Science A, and page 43 for Computer Science AB, and answer questions 41 through 44. These are survey questions and will not affect your examination grade. You may not go back at this time to work on any of the previous questions.

AT AN ADMINISTRATION USING AN ALTERNATE FORM OF THE EXAM ONLY, SAY:

Stop working. Turn to page 39 of the Section I booklet for Computer Science A, and page 41 for Computer Science AB, and answer questions 41 through 44. These are survey questions and will not affect your examination grade. You may not go back at this time to work on any of the previous questions.

Give students approximately 2 minutes to answer the survey questions. Then say:

Close your exam booklet and keep it closed on your desk. Do not insert your answer sheet in the booklet. . . . I will now collect the answer sheets.

After you have collected an answer sheet from every candidate, say:

Seal the Section I booklet with the three seals provided. Peel each seal from the backing sheet and press it on the front cover so it just covers the area marked "PLACE SEAL HERE." Fold it over the open edge and press it to the back cover. Use one seal for each open edge. Be careful not to let the seals touch anything except the marked areas. . . .

Collect the sealed Section I exam booklets. Be sure you receive one from every candidate; then give your break instructions. A 5- to 10-minute break is permitted. Students may talk, move about, or leave the room together to get a drink of water or go to the rest room (see "Breaks During the Examination").

Give your break instructions. Then say:

Testing will resume at ________.

After the break, say:

Open the package containing your Section II booklet. Turn to the back cover of the booklet, and read the instructions at the upper left. . . . Print your identification information in the boxes. . . . Detach the perforation at the top. . . . Fold the flap down, and moisten and press the glue strip firmly along the lower edge. . . . Your identification information should now be covered and will not be known by those scoring your answers.

Read the instructions at the upper right of the back cover. . . .

AT THE MAY 11TH ADMINISTRATION ONLY, SAY:

Take one AP number label from your Candidate Pack and place the label in the AP number box at the top of the page. If you do not have number labels left, copy your number from the front cover of your Candidate Pack into the box.

AT AN ADMINISTRATION USING AN ALTERNATE FORM OF THE EXAM ONLY, SAY:

Print your initials in the three boxes provided. . . . Next, take two AP number labels from the center of your Candidate Pack and place them in the two boxed areas, one below the instructions and one to the left. If you don't have number labels left, copy your number from the front cover of your Candidate Pack into the boxed areas.

Item 5 [Item 6 for late administrations] provides you with the option of giving permission to Educational Testing Service to use your free-response materials for educational research and instructional purposes. Your name would not be used in connection with the free-response materials. Read the statement and answer either "yes" or "no." . . . Are there any questions?

Answer all questions regarding procedure. Then say:

If you will be taking another AP Exam, I will collect your Candidate Pack. You may keep your Candidate Pack if this is your last or only AP Examination.

Collect the Candidate Packs. Then say:

Read the directions for Section II on the back of your booklet. Look up when you have finished. . . . Are there any questions?

Answer all questions regarding procedure. Then say:

You may proceed freely from one question to the next. You are responsible for pacing yourself.

AT THE MAY 11TH ADMINISTRATION ONLY, SAY:

You may use the blank areas in your green insert for scratch paper, but write your actual answers in the Section II booklet.

The blue booklet contains the Large Integer Case Study needed for question 3 on the A exam and question 2 on the AB exam.

AT AN ADMINISTRATION USING AN ALTERNATE FORM OF THE EXAM ONLY, SAY:

The peach booklet contains the Large Integer Case Study needed for question 2 on the A exam, and question 3 on the AB exam. If you need more paper, raise your hand. Are there any questions?

Answer all questions regarding procedure. Then say:

Open the Section II booklet.

AT THE MAY 11TH ADMINISTRATION ONLY, SAY:

Tear out the green insert in the center of the booklet. . . . Print your name, teacher, and school in the upper left-hand corner of the insert. I will be collecting this insert at the end of the administration. It will be returned to you at a later date by your teacher.

When you are ready to begin the exam, note the time here __________. Then say:

Begin work on Section II. You have 1 hour and 45 minutes for this section of the exam.

Allow 1 hour and 45 minutes. Note the time you will stop here __________. You and your proctors should check to be sure all candidates are writing their answers in the Section II booklets.

After 1 hour and 45 minutes, say:

Stop working. Close your Section II booklet and keep it closed on your desk. I will now collect your booklets. Remain in your seats, without talking, while the exam materials are being collected. You will receive your grade reports by mid-July and grades will be available by phone beginning July 1st.

Collect the Section II booklets, the green inserts, and the blue [peach] Large Integer Case Study booklets. Be sure you have one of each from every candidate. Check the back of each Section II booklet to make sure the candidate's AP number appears in the box [two boxes for alternate administrations]. The green inserts and the blue booklets must be stored securely for no less than 48 hours (2 school days) after they are collected. After the 48-hour holding time, the inserts and booklets may be given to the appropriate AP teacher(s) for return to the students. If this is a late administration, the peach booklets must be returned to ETS.

When all examination materials have been collected, dismiss the candidates.

Separate the Computer Science A Exam materials from those for Computer Science AB. Fill in the necessary information for the Computer Science Examinations on the S&R Form. Alternate exams should be recorded on their respective line on the S&R Form. Put the exam materials in locked storage until they are returned to ETS in one shipment after your school's last administration. See "Activities After the Exam."

AREA 1 – COMPLETE THIS AREA AT EVERY EXAMINATION.

PLACE AP® NUMBER LABEL HERE.

A. SIGNATURE

To maintain the security of the exam and the validity of my AP grade, I will allow no one other than myself to see the multiple-choice questions and will seal the appropriate section when asked to do so. In addition, I am aware of and agree to the Program's policies and procedures as outlined in the 1999 *AP Bulletin for Students and Parents.*

Sign your name as it will appear on your college applications.

Answer Sheet for May 1999, Form 3VBP

Advanced Placement Program®
THE COLLEGE BOARD

PAGE 1

B. YOUR AP NUMBER

C. NAME — Omit spaces, hyphens, apostrophes, Jr. or II.

Last Name - first 15 letters | First Name - first 12 letters | MI

D. ADMIN. DAY IN MAY

E. TIME OF DAY

- 6 AM
- 7 AM
- 8 AM
- 9 AM
- 10 AM
- 11 AM
- 12 PM
- 1 PM
- 2 PM
- 3 PM
- 4 PM
- 5 PM

F. AP EXAMINATION TO BE TAKEN USING THIS ANSWER SHEET

Print examination name:

Fill in the appropriate oval below for examination name and number.

No.	Examination	No.	Examination
07	U.S. History	55	German Language
13	Art History	57	Gov. & Pol.: **U.S.**
14	Art: Studio Drawing	58	Gov. & Pol.: **Comp.**
15	Art: Studio General	59	International English Language
20	Biology	60	Latin: Vergil
25	Chemistry	61	Latin: Literature
31	Computer Science A	66	Calculus AB
33	Computer Science AB	68	Calculus BC
34	Economics: **Micro**	75	Music Theory
35	Economics: **Macro**	78	Physics B
36	Eng. Language & Comp.	80	Physics C: **Mech.**
37	Eng. Literature & Comp.	80	Physics C: **E & M**
40	Environmental Science	85	Psychology
43	European History	87	Spanish Language
48	French Language	89	Spanish Literature
51	French Literature	90	Statistics

G. What language do you know best?

- English
- English and another language about the same
- Another language

H. DO NOT COMPLETE THIS SECTION UNLESS INSTRUCTED TO DO SO.

Essay Choices
Fill in the ovals under the numbers of the essay questions you answered in this examination.

1 2 3 4 5 6 7 8 9 10 11 12

SCHOOL USE ONLY

Fee Reduction Granted | Report to Teachers Section Designation: 1 2 3 4 5 6 7 8 9

AREA 2 – COMPLETE THIS AREA ONLY ONCE.

I. SEX

1. Male
2. Female

J. PRESENT GRADE LEVEL

1. 9th grade
2. 10th grade
3. 11th grade
4. 12th grade
5. College
6. Other

K. DATE OF BIRTH

Month | Day | Year

01 Jan. · 02 Feb. · 03 Mar. · 04 Apr. · 05 May · 06 June · 07 July · 08 Aug. · 09 Sept. · 10 Oct. · 11 Nov. · 12 Dec.

L. SOCIAL SECURITY NUMBER

M. ETHNIC GROUP

1. American Indian or Alaskan native
2. Black or African American
3. Mexican American or Chicano
4. Asian, Asian American, or Pacific Islander
5. Puerto Rican
6. South American, Latin American, Central American, or other Hispanic
7. White
8. Other

N. EXPECTED DATE OF COLLEGE ENTRANCE

F	Fall	8	1999
W	Winter/Spring	9	2000
S	Summer	0	2001
U	Undecided	1	2002

O. Will you be applying for Sophomore Standing at college?

Yes No

ETS USE ONLY

Exam 1 2 | Exam 1 2

P. STUDENT SEARCH SERVICE OF THE COLLEGE BOARD
(Complete ONLY if you are a SOPHOMORE or a JUNIOR.)

- Yes, I want the College Board to send information about me to colleges, universities, and governmental scholarship programs interested in students like me.
- No, I do not want the College Board to send information about me to colleges, universities, and governmental scholarship programs through the Student Search Service.

Copyright © 1999 by College Entrance Examination Board and Educational Testing Service. All rights reserved.
College Board, Advanced Placement Program, AP, and the acorn logo are registered trademarks of the College Entrance Examination Board.

18002 • 00657 • TF128M1400 Q2856-06 MH98295 I.N. 240070 1 2 3 4

Advanced Placement Program®
THE COLLEGE BOARD

Q. THIS SECTION IS FOR THE SURVEY QUESTIONS IN THE CANDIDATE PACK. (DO NOT PUT RESPONSES TO EXAM QUESTIONS IN THIS SECTION.) BE SURE EACH MARK IS DARK AND COMPLETELY FILLS THE OVAL.

1 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
2 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
3 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
4 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
5 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

DO NOT COMPLETE THIS SECTION UNLESS INSTRUCTED TO DO SO.

R. If this answer sheet is for the French Language, French Literature, German Language, Spanish Language, or Spanish Literature Examination, please answer the following questions. (Your responses will not affect your grade.)

1. Have you lived or studied for one month or more in a country where the language of the exam you are now taking is spoken? ◯ Yes ◯ No
2. Do you regularly speak or hear the language at home? ◯ Yes ◯ No

INDICATE YOUR ANSWERS TO THE EXAM QUESTIONS IN THIS SECTION. IF A QUESTION HAS ONLY FOUR ANSWER OPTIONS, DO NOT MARK OPTION (E). YOUR ANSWER SHEET WILL BE SCORED BY MACHINE. USE ONLY NO. 2 PENCILS TO MARK YOUR ANSWERS ON PAGES 2 AND 3 (ONE RESPONSE PER QUESTION). AFTER YOU HAVE DETERMINED YOUR RESPONSE, BE SURE TO COMPLETELY FILL IN THE OVAL CORRESPONDING TO THE NUMBER OF THE QUESTION YOU ARE ANSWERING. STRAY MARKS AND SMUDGES COULD BE READ AS ANSWERS, SO ERASE CAREFULLY AND COMPLETELY. ANY IMPROPER GRIDDING MAY AFFECT YOUR GRADE.

1 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
2 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
3 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
4 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
5 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
6 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
7 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
8 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
9 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
10 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
11 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
12 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
13 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
14 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
15 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
16 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
17 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
18 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
19 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
20 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
21 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
22 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
23 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
24 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
25 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
26 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
27 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
28 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
29 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
30 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
31 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
32 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
33 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
34 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
35 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
36 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
37 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
38 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
39 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
40 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
41 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
42 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
43 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
44 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
45 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
46 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
47 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
48 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
49 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
50 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
51 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
52 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
53 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
54 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
55 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
56 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
57 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
58 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
59 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
60 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
61 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
62 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
63 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
64 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
65 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
66 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
67 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
68 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
69 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
70 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
71 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
72 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
73 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
74 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
75 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

FOR QUESTIONS 76-151, SEE PAGE 3.

DO NOT WRITE IN THIS AREA.

MULTIPLE-CHOICE BOOKLET SERIAL NUMBER

BE SURE EACH MARK IS DARK AND COMPLETELY FILLS THE OVAL. IF A QUESTION HAS ONLY FOUR ANSWER OPTIONS, DO NOT MARK OPTION E.

76 (A) (B) (C) (D) (E)
77 (A) (B) (C) (D) (E)
78 (A) (B) (C) (D) (E)
79 (A) (B) (C) (D) (E)
80 (A) (B) (C) (D) (E)
81 (A) (B) (C) (D) (E)
82 (A) (B) (C) (D) (E)
83 (A) (B) (C) (D) (E)
84 (A) (B) (C) (D) (E)
85 (A) (B) (C) (D) (E)
86 (A) (B) (C) (D) (E)
87 (A) (B) (C) (D) (E)
88 (A) (B) (C) (D) (E)
89 (A) (B) (C) (D) (E)
90 (A) (B) (C) (D) (E)
91 (A) (B) (C) (D) (E)
92 (A) (B) (C) (D) (E)
93 (A) (B) (C) (D) (E)
94 (A) (B) (C) (D) (E)
95 (A) (B) (C) (D) (E)
96 (A) (B) (C) (D) (E)
97 (A) (B) (C) (D) (E)
98 (A) (B) (C) (D) (E)
99 (A) (B) (C) (D) (E)
100 (A) (B) (C) (D) (E)
101 (A) (B) (C) (D) (E)
102 (A) (B) (C) (D) (E)
103 (A) (B) (C) (D) (E)
104 (A) (B) (C) (D) (E)
105 (A) (B) (C) (D) (E)
106 (A) (B) (C) (D) (E)
107 (A) (B) (C) (D) (E)
108 (A) (B) (C) (D) (E)
109 (A) (B) (C) (D) (E)
110 (A) (B) (C) (D) (E)
111 (A) (B) (C) (D) (E)
112 (A) (B) (C) (D) (E)
113 (A) (B) (C) (D) (E)
114 (A) (B) (C) (D) (E)
115 (A) (B) (C) (D) (E)
116 (A) (B) (C) (D) (E)
117 (A) (B) (C) (D) (E)
118 (A) (B) (C) (D) (E)
119 (A) (B) (C) (D) (E)
120 (A) (B) (C) (D) (E)
121 (A) (B) (C) (D) (E)
122 (A) (B) (C) (D) (E)
123 (A) (B) (C) (D) (E)
124 (A) (B) (C) (D) (E)
125 (A) (B) (C) (D) (E)
126 (A) (B) (C) (D) (E)
127 (A) (B) (C) (D) (E)
128 (A) (B) (C) (D) (E)
129 (A) (B) (C) (D) (E)
130 (A) (B) (C) (D) (E)
131 (A) (B) (C) (D) (E)
132 (A) (B) (C) (D) (E)
133 (A) (B) (C) (D) (E)
134 (A) (B) (C) (D) (E)
135 (A) (B) (C) (D) (E)
136 (A) (B) (C) (D) (E)
137 (A) (B) (C) (D) (E)
138 (A) (B) (C) (D) (E)
139 (A) (B) (C) (D) (E)
140 (A) (B) (C) (D) (E)
141 (A) (B) (C) (D) (E)
142 (A) (B) (C) (D) (E)
143 (A) (B) (C) (D) (E)
144 (A) (B) (C) (D) (E)
145 (A) (B) (C) (D) (E)
146 (A) (B) (C) (D) (E)
147 (A) (B) (C) (D) (E)
148 (A) (B) (C) (D) (E)
149 (A) (B) (C) (D) (E)
150 (A) (B) (C) (D) (E)
151 (A) (B) (C) (D) (E)

ETS USE ONLY

	R	W	FS
PT1			
PT2			
PT3			
PT4			
TOT			
EQ			
TA1			
TA2			

DO NOT WRITE IN THIS AREA.

AREA 3 – COMPLETE THIS AREA ONLY ONCE.

1. YOUR MAILING ADDRESS

- YOUR GRADE REPORT WILL BE MAILED TO THIS ADDRESS IN JULY.
- USING THE ABBREVIATIONS GIVEN IN YOUR CANDIDATE PACK, FILL ADDRESS INTO BOXES PROVIDED. IF YOUR ADDRESS DOES NOT FIT, SEE ITEM 2 BELOW.
- INDICATE A SPACE IN YOUR ADDRESS BY LEAVING A BLANK BOX AND FILLING IN THE CORRESPONDING DIAMOND (◇) BELOW THE BOX.

Street

City

STATE

1	AL	28	NE
2	AK	29	NV
3	AZ	30	NH
4	AR	31	NJ
5	CA	32	NM
6	CO	33	NY
7	CT	34	NC
8	DE	35	ND
9	DC	36	OH
10	FL	37	OK
11	GA	38	OR
12	HI	39	PA
13	ID	40	RI
14	IL	41	SC
15	IN	42	SD
16	IA	43	TN
17	KS	44	TX
18	KY	45	UT
19	LA	46	VT
20	ME	47	VA
21	MD	48	WA
22	MA	49	WV
23	MI	50	WI
24	MN	51	WY
25	MS	52	Puerto Rico
26	MO		
27	MT	53	Other

ZIP OR POSTAL CODE

1a. COUNTRY CODE

3. TELEPHONE NUMBER

Area Code

2. If the address gridded above is not complete enough for delivery of your grade report, please fill in this oval → ◯ and print your complete address below.

Address

Address

City | State or Province

Country | Zip or Postal Code

4. SCHOOL YOU ATTEND

School Code

School Name, City, and State

← Make sure you have correctly entered your School Code and filled in the appropriate ovals.

School Name

City

State

5. COLLEGE TO RECEIVE YOUR AP GRADES

College Code

College Name and Address

Using the College Code list in the AP Candidate Pack, indicate the one college that has *accepted you* and that *you plan to attend.*

College Name

City

State

COMPUTER SCIENCE A

CALCULATORS, REFERENCE MATERIALS, OR OTHER AIDS ARE NOT TO BE USED DURING THE EXAMINATION.

Three hours are allotted for this examination: 1 hour and 15 minutes for Section I, which consists of 40 multiple-choice questions, and 1 hour and 45 minutes for Section II, which consists of 4 free-response questions. In determining your grade, the two sections are given equal weight. Section I is printed in this examination booklet. Section II is printed in a separate booklet. In addition, following Section I there are 4 survey questions to be answered in 2 minutes.

SECTION I

Time — 1 hour and 15 minutes

Number of questions — 40

Percent of total grade — 50

Section I of this examination contains 40 multiple-choice questions. Therefore, please be careful to fill in only the ovals that are preceded by numbers 1 through 40 on your answer sheet when answering the examination questions. Also, please be careful to fill in the ovals preceded by the numbers 41-44 when answering the survey questions.

General Instructions

DO NOT OPEN THIS BOOKLET UNTIL YOU ARE INSTRUCTED TO DO SO.

INDICATE ALL YOUR ANSWERS TO QUESTIONS IN SECTION I ON THE SEPARATE ANSWER SHEET. No credit will be given for anything written in this examination booklet, but you may use the booklet for notes or scratchwork. After you have decided which of the suggested answers is best, COMPLETELY fill in the corresponding oval on the answer sheet. Give only one answer to each question. If you change an answer, be sure that the previous mark is erased completely.

Example:

Chicago is a

(A) state
(B) city
(C) country
(D) continent
(E) village

Sample Answer

Many candidates wonder whether or not to guess the answer to questions about which they are not certain. In this section of the examination, as a correction for haphazard guessing, one-fourth of the number of questions you answer incorrectly will be subtracted from the number of questions you answer correctly. It is improbable, therefore, that mere guessing will improve your score significantly; it may even lower your score, and it does take time. If, however, you are not sure of the correct answer but have some knowledge of the question and are able to eliminate one or more of the answer choices as wrong, your chance of getting the right answer is improved, and it may be to your advantage to answer such a question.

Use your time effectively, working as rapidly as you can without losing accuracy. Do not spend too much time on questions that are too difficult. Go on to other questions and come back to the difficult ones later if you have time. It is not expected that everyone will be able to answer all the multiple-choice questions.

Copyright © 1999 College Entrance Examination Board and Educational Testing Service. All rights reserved.
Certain test materials are copyrighted solely in the name of ETS.

COMPUTER SCIENCE A

SECTION I

Time—1 hour and 15 minutes

Number of questions—40

Percent of total grade—50

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 on pages 41-43.

USE THIS SPACE FOR SCRATCHWORK.

1. Consider the following program.

```
int main()
{
  int k, sum;

  do
  {
    cout << "Enter a nonnegative number to sum, ";
    cout << "negative number to quit:";
    cin >> k;
    if (k > 0)
    {
      sum += k;
    }
  } while (k >= 0);
  cout << sum << endl;
  return 0;
}
```

The program contains the incorrect use of an uninitialized variable. Which of the following is the first line in which this occurs?

(A) `k` in `"cin >> k"`
(B) `k` in `"(k > 0)"`
(C) `sum` in `"sum += k"`
(D) `k` in `"sum += k"`
(E) `sum` in `"cout << sum << endl"`

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

Questions 2-3 refer to the following functions.

```
int Add(int x, int y);
// postcondition: returns x + y

int Multiply(int x, int y);
// postcondition: returns x * y
```

2. What is the value of the expression `Multiply(3, Add(4, 5))` ?

(A) 12
(B) 17
(C) 23
(D) 27
(E) 60

3. Consider the following expression.

```
Multiply(2, Add(Multiply(a, b), Add(Multiply(a, c), Multiply(b, c))))
```

Which of the following corresponds to this expression?

(A) `2 * a * b + a * c + b * c`
(B) `a * b + a * c + b * c + 2`
(C) `2 * (a * b + (a * c + b * c))`
(D) `a * b + (a * c + b * c)`
(E) `a * b + (a * c + b * c) * 2`

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

4. Consider the following function.

```
int Something(int a, int b)
{
  if (b <= 1)
  {
    return a;
  }
  else
  {
    return Something(a, b - 1);
  }
}
```

What value is returned by the call `Something(4, 6)` ?

(A) 4
(B) 6
(C) 24
(D) 1296
(E) 4096

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

5. Consider the following incomplete function.

```
int Total(const apvector<int> & scores)
// precondition: The sentinel -999 occurs somewhere in scores
{
  int k = 0;
  int sum = 0;

  while (scores[k] != -999)
  {
    <program statements>
  }
  return sum;
}
```

Function `Total` is intended to return the sum of the integers in parameter `scores`, beginning with the first integer in `scores` and up to, but not including the sentinel –999 (which occurs somewhere in `scores`). Which of the following code segments could be used to replace *<program statements>* so that `Total` will work as intended?

(A)
```
sum += scores[k];
k++;
```

(B)
```
k++;
sum += scores[k];
```

(C)
```
k++;
sum += k;
```

(D)
```
sum += k;
k++;
```

(E)
```
if (scores[k] != -999)
{
  sum += scores[k];
}
```

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

6. Consider the following code segment.

```
apmatrix<int> M;
Initialize(M);      // resizes M to be a square matrix
                    // and initializes its elements

int sum = 0;
int k;

for (k = 0; k < M.numrows(); k++)
{
   sum += M[k][M.numrows() - k - 1];
}
```

Assume that after the call to `Initialize`, `M` represents the matrix shown below.

	0	**1**	**2**	**3**
0	1	1	1	1
1	1	2	3	4
2	2	2	2	2
3	2	4	6	8

What value will `sum` contain after the code segment is executed?

(A) 4
(B) 8
(C) 13
(D) 20
(E) 42

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

Questions 7-8 are based on the following incomplete function.

```
int Fun(int x, int y)
// precondition: (x * y) ≥ 0
// postcondition: returns a value ≥ 0
{
  <body of Fun>
}
```

Assume that *<body of Fun>* is replaced with code so that `Fun` meets the specification defined by its precondition and its postcondition.

7. What can be assumed about the value returned by the call `Fun(0, 0)` ?

   (A) The value is `0`.
   (B) The value is not equal to `0`.
   (C) The value is less than or equal to `0`.
   (D) The value is greater than or equal to `0`.
   (E) No assumption can be made about the value returned.

8. What can be assumed about the value returned by the call `Fun(-1, 1)` ?

   (A) The value is `-1`.
   (B) The value is `0`.
   (C) The value is `1`.
   (D) The value is greater than or equal to `0`.
   (E) No assumption can be made about the value returned.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

9. Consider the following definitions.

```
apvector<char> A(100);
apvector<char> B(100);
```

Consider the following code segment.

```
int k = 0;
while ((k < A.length()) && (A[k] != B[k]))
{
  k++;
}
```

Which of the following must be true after the while loop terminates?

(A) `k >= A.length()`
(B) `k < A.length()`
(C) `(k < A.length()) && (A[k] != B[k])`
(D) `(k >= A.length()) || (A[k] != B[k])`
(E) `(k >= A.length()) || (A[k] == B[k])`

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

10. Consider the incomplete function `PowerOf` given below. The call `PowerOf(n, x)` should return the quantity n^x.

```
int PowerOf(int base, int power)
// precondition:  power ≥ 1
// postcondition: returns base^power
{
  int result;

  if (<expression1> == 1)
  {
    result = <expression2> ;
  }
  else
  {
    result = <expression3> * PowerOf(base, power - 1);
  }
  return result;
}
```

Which of the following could be used to replace *<expression1>*, *<expression2>*, and *<expression3>* so that `PowerOf` will work as intended?

	<expression1>	<expression2>	<expression3>
(A)	`power`	`base`	`result`
(B)	`power`	`base`	`power`
(C)	`power`	`base`	`base`
(D)	`base`	`power`	`result`
(E)	`base`	`power`	`base`

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

11. Assume `A` is defined as follows.

```
apvector<int> A(5);
```

Consider the following code segment.

```
int k;
for (k = A.length() - 1; k > 0; k--)
{
  A[k] = A[k - 1];
}
```

Assume `A` contains the following values before the code segment is executed.

0	1	2	3	4
10	15	20	25	30

What values will `A` contain after the code segment is executed?

(A)

10	10	10	10	10

(B)

10	10	15	20	25

(C)

10	15	20	25	30

(D)

15	20	25	30	30

(E)

25	25	25	25	25

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

Questions 12-13 concern the following data structure, designed to store information about hiking trails in the United States.

```
struct Trail
{
  apstring name;               // name of trail
  apstring state;              // location of trail
  double distance;             // length of trail in kilometers
  apvector<bool> goodMonths;
  // goodMonths[k] == true if this trail is good for hiking in month k;
  //   otherwise, goodMonths[k] == false

  Trail();
  // constructor initializes goodMonths to length 12
};
```

12. Assume that `T` is an initialized variable of type `Trail` and the integers `j` and `k` represent valid months. Which of the following expressions always evaluates to true if the trail represented by `T` is good for hiking during month `j` or `k` ?

(A) `T.goodMonths[j] && T.goodMonths[k]`
(B) `T.goodMonths[j] || T.goodMonths[k]`
(C) `(T.goodMonths == j) || (T.goodMonths == k)`
(D) `(T.goodMonths == j) && (T.goodMonths == k)`
(E) `T.goodMonths[j] == T.goodMonths[k]`

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

13. Consider the incomplete function `PrintTrails` given below. `PrintTrails` should print the names of the trails in its parameter `trailArray` that are good for hiking in the month specified by parameter `month`.

```
void PrintTrails(const apvector<Trail> & trailArray, int month)
{
  int k;
  for (k = 0; k < trailArray.length(); k++)
  {
    <loop body>
  }
}
```

Which of the following could be used to replace *<loop body>* so that `PrintTrails` works as intended?

(A)
```
if (trailArray[k].goodMonths[month])
{
    cout << trailArray[k].name << endl;
}
```

(B)
```
if (trailArray[month].goodMonths[k])
{
    cout << trailArray[month].name << endl;
}
```

(C)
```
if (trailArray[k] == month)
{
    cout << trailArray[k].name << endl;
}
```

(D)
```
if (trailArray.goodMonths == month)
{
    cout << trailArray.goodMonths.name << endl;
}
```

(E)
```
if (trailArray.goodMonths[k] == month)
{
    cout << trailArray.goodMonths[k].name << endl;
}
```

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

14. Consider the following incomplete function.

```
int Mystery(int k)
{
  if (k <= 0)
  {
    return 0;
  }
  else
  {
    return (<missing code>);
  }
}
```

Which of the following could be used to replace `<missing code>` so that the value of `Mystery(5)` is `15` ?

(A) `k + Mystery(k - 1)`
(B) `k * Mystery(k - 1)`
(C) `Mystery(k - 1)`
(D) `Mystery(k + 1)`
(E) `Mystery(k - 1) * Mystery(k + 1)`

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

15. Assume that an array contains 100 integers sorted in increasing order. Two alternatives to search the array for a particular integer are a sequential and a binary search. When searching for a value that is in the array, which of the following best characterizes the greatest number of items in the array that will be examined during each kind of search?

	Sequential	Binary
(A)	100	1
(B)	100	7
(C)	100	50
(D)	50	7
(E)	50	25

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

Questions 16-20 refer to the code from the Large Integer case study. A copy of the code is provided as part of this exam.

16. The `BigInt` functions assume that `BigInt` values are represented with no leading zeroes. The function `Normalize` is used to remove leading zeroes when necessary. Suppose that all calls to `Normalize` were removed. Which of the following functions would no longer work correctly?

    (A) `operator+`
    (B) `operator*`
    (C) `ToDouble`
    (D) `ToInt`
    (E) `LessThan`

17. The following statement is often used in `BigInt` function definitions.

```
BigInt result(lhs);
```

    Assuming `lhs` is of type `BigInt`, what is the purpose of this statement?

    (A) To make a copy of `lhs` and store the copy in `result`
    (B) To make a copy of `result` and store the copy in `lhs`
    (C) To initialize `result` to `0`
    (D) To initialize `lhs` to `0`
    (E) To declare function `result` with `lhs` as a parameter

18. The following code segment from a client program is intended to find the absolute value of a `BigInt`. However, the code segment does not work as intended.

```
BigInt big;

cout << " Enter a large number ";
cin >> big;

if (big.IsNegative())
{
  big *= -1;
}
```

    Which of the following describes the error in the code segment?

    (A) An integer cannot be multiplied by a `BigInt`.
    (B) `cin` cannot be used with a `BigInt`.
    (C) `IsNegative()` is not a member function of the class `BigInt`.
    (D) `IsNegative()` is a private member function of the class `BigInt`, therefore it cannot be used by the client program.
    (E) The condition `big.IsNegative()` is incorrect and must be changed to `IsNegative()`.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

19. Suppose the `BigInt` class is redesigned so that the private data member `mySign` is changed to be of type `int`. The expression `mySign == +1` indicates that the `BigInt` is positive, and `mySign == -1` indicates that the `BigInt` is negative. Which of the following member functions would also need to be changed to implement this new design?

    I. `IsNegative`
    II. `LessThan`
    III. `operator<`

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

20. Function `Power` takes two parameters, `base` and `exponent`. `Power` is intended to return a `BigInt` that represents the value of `base` to the `exponent` power. Assume that `base` and `exponent` will only take values within the range of an `int` variable for your machine and that `exponent` is always non-negative. Which of the following is the best declaration for the function `Power`?

    (A) `void Power(int base, int exponent, BigInt result);`
    (B) `void Power(BigInt base, BigInt exponent, BigInt result);`
    (C) `BigInt Power(int base, int exponent);`
    (D) `BigInt Power(BigInt base, int exponent);`
    (E) `BigInt Power(BigInt base, BigInt exponent);`

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

21. The program for a video game will use several graphics routines. The program design team plans to place these routines in a graphics class, which can be compiled separately from the video game program. Which of the following would be an advantage of this plan?

    I. The graphics routines can be tested independently of the video game program, thus making it easier to locate errors in both the graphics routines and the video game program.
    II. The programmers assigned to write the video game program can focus on the issues of that program without spending time considering how the graphics routines will be implemented.
    III. The graphics class will be available for use in other programs.

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

Unauthorized copying or reusing any part of this page is illegal.

22. Consider the following code segment to print a calendar.

```
int month, year;
cout << "Enter year: ";
cin >> year;
for (month = 1; month <= 12; month++)
{
  PrintHeading(month, year);
  PrintDays(month, year);
}
```

Consider the following function.

```
void PrintDays(int month, int year)
{
  int day;
  PrintSpaces(month, year); // indent the first week of the month
  for (day = 1; day <= NumDaysIn(month, year); day++)
  {
    cout << day << " ";
    if (EndOfWeek(day, month, year))
    {
      cout << endl;
    }
  }
}
```

Suppose that when the program is run, every month is printed correctly except for February, for which only a heading and some white space is printed. Of the following functions, which is most likely to contain the error?

(A) `NumDaysIn`
(B) `PrintSpaces`
(C) `EndOfWeek`
(D) `PrintHeading`
(E) `PrintDays`

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

23. Consider the following code segment.

```
int row, col;
int sum = 0;
apmatrix<int> A;
Initialize(A); // resizes A and initializes its elements

for (row = 0; row < A.numrows(); row++)
{
  for (col = 0; col < A.numcols(); col++)
  {
    sum += A[row][col];
  }
}
```

Which of the following best describes the result of executing the code segment?

(A) Each element in the two-dimensional array `A` contains the value `0`.
(B) Each element in the two-dimensional array `A` contains the sum of its row number and its column number.
(C) Each element in the two-dimensional array `A` contains the sum of all preceding elements in two-dimensional array `A`.
(D) The variable `sum` contains the sum of the values in the two-dimensional array `A`.
(E) The variable `sum` contains the value `row * col`.

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

24. Consider the following code.

```
apmatrix<int> M;
Initialize(M);    // resizes M to be a square matrix
                  // and initializes its elements
```

Which of the following code segments correctly sets a diagonal of the two-dimensional array `M` to contain all zeroes?

(A)
```
int row = 0;
int col = 0;
while ((row < M.numrows()) && (col < M.numcols()))
{
   M[row][col] = 0;
   row++;
   row = col;
}
```

(B)
```
int row = 0;
int col = 0;
while (row < M.numrows())
{
   M[row][col] = 0;
   row++;
   col = row;
   col++;
}
```

(C)
```
int row = 0;
int col = 0;
while (row < M.numrows())
{
   M[row][col] = 0;
   row++;
}
```

(D)
```
int row;
for (row = 0; row < M.numrows(); row++)
{
   M[row][row] = 0;
}
```

(E)
```
int row;
for (row = 1; row <= M.numrows(); row++)
{
   M[row][row] = 0;
}
```

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

25. For each hour of the day, a weather station records temperature using integer values, and pressure, wind speed, and wind direction using values of type `double`. Of the following definitions of `dailyRecord`, which would be most suitable for recording these weather readings for one day?

(A)
```
apmatrix<int> dailyRecord(24, 4);
```

(B)
```
apvector<int> dailyRecord(96);
```

(C)
```
struct WeatherInfo
{
   int temperature;
   double pressure;
   double windSpeed;
   double windDir;
};
WeatherInfo dailyRecord;
```

(D)
```
struct WeatherInfo
{
   int temperature;
   double pressure;
   double windSpeed;
   double windDir;
};
apvector<WeatherInfo> dailyRecord(24);
```

(E)
```
struct WeatherInfo
{
   int temperature;
   double pressure;
   double windSpeed;
   double windDir;
};
apvector<WeatherInfo> dailyRecord(366 * 24);
```

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

26. C++ classes can include both public and private data members and member functions. Which of the following statements represents the best design decision regarding the public and private sections of a class?

(A) All data members should be public to make it easier for client programs to use such data.
(B) All member functions should be public to facilitate future changes to parameter lists of member functions.
(C) All data members should be private to minimize the dependency between client programs and the manner in which data is stored in the class.
(D) Some member functions should be private to minimize memory usage.
(E) All data members should be public and all member functions should be private to make it easier to modify the class without requiring changes in the code of the client program.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

27. Consider the following function.

```
bool SomethingDifferent(bool p, bool q)
{
  return ((p || q) && !(p && q));
}
```

What does function `SomethingDifferent` return?

(A) `SomethingDifferent` always returns `false`.
(B) `SomethingDifferent` always returns `true`.
(C) `SomethingDifferent` returns `true` whenever `p` is `false`.
(D) `SomethingDifferent` returns `true` whenever `q` is `false`.
(E) `SomethingDifferent` returns `true` whenever `p` is not equal to `q`.

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

28. Which of the following statements would best characterize a well-designed program?

    I. Functions can be tested independently before integrating them into the final program.
    II. Client programs know about, and take advantage of, implementation details of abstract data types.
    III. The algorithmic details of the abstract data types can be altered without changing client routines.

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

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

29. Consider the following program.

```
void One(int a, int b)
{
  a = b + 1;
  b = a + 2;
}

void Two(int & a, int & b)
{
  a = b + 1;
  b = a + 2;
}

void Three(int & a, int b)
{
  a = b + 1;
  b = a + 2;
}

int main()
{
  int x, y;

  x = 1;
  y = 2;
  One(x, y);
  Two(x, y);
  Three(x, y);
  cout << x << " " << y << endl;
  return 0;
}
```

What is the output of the program?

(A) 1 2
(B) 4 4
(C) 6 5
(D) 6 8
(E) 9 8

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

30. Consider the following definitions and code segment.

```
apvector<int> A(7);

int x;

for (x = 0; x < A.length(); x++)
{
  A[x] = x;
}
for (x = 0; x < A.length(); x++)
{
  A[x / 3] = A[x];
}
```

What values will `A` contain after the code segment is executed?

(A) 0 0 1 1 1 2 2
(B) 0 1 2 3 4 5 6
(C) 1 2 0 1 2 0 1
(D) 2 3 4 5 6 0 1
(E) 2 5 6 3 4 5 6

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

31. Assume that `A` is an array of `N` integers and that variable `k` has a value in the range $0 \leq k < N$. Also assume that the following assertion is true:

```
for all j,  0 ≤ j < k, A[j] < A[j + 1]
```

Which of the following is a valid conclusion?

(A) All elements of `A` are in increasing order.
(B) All elements of `A` are in decreasing order.
(C) Elements `0` through `k` of `A` are in increasing order.
(D) Elements `0` through `k` of `A` are in decreasing order.
(E) The smallest value in `A` is stored in `A[0]`.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

32. Consider the following code segment. Assume that neither *<condition 1>*, *<condition 2>*, nor *<condition 3>* changes the value of `k`.

```
int k = 0;
if (<condition 1>)
{
  k++;
}
if (<condition 2>)
{
  k++;
}
if (<condition 3>)
{
  k++;
}
```

What are the possible final values of `k` after the code segment executes?

(A) 0 only
(B) 1 only
(C) 0 or 1 only
(D) 1, 2, or 3 only
(E) 0, 1, 2, or 3

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

33. Consider designing a data structure to represent the positions of 50 game pieces on a 100 x 100 gameboard. (The position of a game piece is the row and column number of the square that it is on.) Two alternatives are described below.

    Method 1. Use a two-dimensional array of Boolean values indexed by row and column number, where each array element represents one square of the gameboard. If there is a game piece on that square, then the array element is true; otherwise, the array element is false.

    Method 2. Use a one-dimensional array in which each element represents the position of one game piece (i.e., the row and column number of the square that it is on).

    Which of the following is true?

    (A) Method 1 is not suitable if two game pieces can occupy the same square of the gameboard.
    (B) Method 2 is not suitable if two game pieces can occupy the same square of the gameboard.
    (C) Printing the positions of all game pieces can be done more efficiently by using Method 1 than by using Method 2.
    (D) Determining whether there is a game piece on a particular square (given the row and column numbers) can be done more efficiently by using Method 2 than by using Method 1.
    (E) Removing the game piece from a particular square (given its row and column numbers) can be done more efficiently by using Method 2 than by using Method 1.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

34. Consider the following function.

```
void Print(int count)
{
  int k;

  if (count > 0)
  {
    cin >> k;
    Print(count - 1);
    cout << k << endl;
  }
}
```

Of the following, which best describes what is printed as a result of the call `Print(10)` ?

(A) Nothing is printed because a run-time error occurs.
(B) Nothing is printed because the if condition never evaluates to true.
(C) Ten integers are printed in the same order in which they were read.
(D) Ten integers are printed in the reverse order in which they were read.
(E) Only the nonzero values that were read are printed; they are printed in the same order in which they were read.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

35. Consider the following code segment.

```
apvector<int> A;
Initialize(A);    // resizes A and initializes its elements

int k;
for (k = 0; k < A.length(); k++)
{
  Swap(A[k], A[A.length() - k - 1]);
}
```

Assume that function `Swap` interchanges the values of its parameters. Which of the following best characterizes the effect of the for loop?

(A) It sorts the elements of `A`.
(B) It reverses the elements of `A`.
(C) It reverses the order of the first half of `A` and leaves the second half unchanged.
(D) It reverses the order of the second half of `A` and leaves the first half unchanged.
(E) It leaves all of the elements of `A` in their original order.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

36. The following declaration and incomplete function is intended to sort an array of unique integers in increasing order using the quicksort algorithm.

```
void Partition(apvector<int> & A, int first, int last,
               int & pivotPos);

void QuickSort(apvector<int> & A, int first, int last)
{
  int pivotPos;

  // If the subarray has at least 2 elements, partition
  // and recursively sort the two partitions.

  if (first < last)
  {
    Partition(A, first, last, pivotPos);

    <statements>
  }
}
```

The variables `first` and `last` are the indices of the first and last elements in the subarray of array `A` to be sorted. The function `Partition` performs the task of splitting the array into two subarrays around a pivot point, `pivotPos`, chosen by `Partition`. After the call to `Partition`, the subarray from `A[first]` to `A[pivotPos - 1]` contains integers that are less than `A[pivotPos]`, and the subarray from `A[pivotPos + 1]` to `A[last]` contains integers greater than `A[pivotPos]`. The element `A[pivotPos]` is in its final sorted position.

Which of the following can be used to replace *`<statements>`* so that `QuickSort` will work as intended?

(A)
```
QuickSort(A, first, pivotPos + 1);
QuickSort(A, pivotPos - 1, last);
```

(B)
```
QuickSort(A, last, pivotPos);
QuickSort(A, pivotPos, first);
```

(C)
```
QuickSort(A, first, pivotPos - 1);
QuickSort(A, last, pivotPos + 1);
```

(D)
```
QuickSort(A, first, pivotPos - 1);
QuickSort(A, pivotPos + 1, last);
```

(E)
```
QuickSort(A, pivotPos - 1, first);
QuickSort(A, last, pivotPos + 1);
```

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

Questions 37-38 refer to the following information.

Consider the following class declaration.

```
class Restaurant
{
  public:
      // Accessors
    apstring Name() const;  // returns the restaurant's name
    double Price() const;   // returns the price of a meal
                            // (all meals in a specific
                            // restaurant cost the same)
    int Capacity() const;   // returns the maximum number of
                            // customers the restaurant can
                            // serve at one time
      // Other member functions not shown

  private:
    apstring myName;        // the restaurant's name
      // other data members not shown
};
```

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

37. Assume that a client program declares and initializes `rList` as follows:

```
apvector<Restaurant> rList;
Initialize(rList); //resizes rList and initializes its elements
```

Which of the following code segments correctly prints the names of all the restaurants whose meal price is under $10.00 ?

I.
```
int r;
for (r = rList.length() - 1; r >= 0; r--)
{
  if (rList[r].Price() < 10.00)
  {
    cout << rList[r].Name() << endl;
  }
}
```

II.
```
int r;
for (r = 0; r < rList.length(); r++)
{
  if (rList[r].Price() < 10.00)
  {
    cout << rList[r].Name() << endl;
  }
}
```

III.
```
int r;
for (r = 0; r < rList.length(); r++)
{
  if (rList[r].Price() < 10.00)
  {
    cout << rList[r].myName << endl;
  }
}
```

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

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

38. Consider the following function.

```
void PrintSomeRestaurants(const apvector<Restaurant> & rList)
// precondition: rList.length() > 0
{
  int r;
  int numRests = rList.length();
  double sum = 0.0;
  double average;

  for (r = 0; r < numRests; r++)
  {
    sum += rList[r].Price();
  }
  average = sum / numRests;
  for (r = 0; r < numRests; r++)
  {
    if ((rList[r].Capacity() >= 50) && (rList[r].Price() < average))
    {
      cout << rList[r].Name() << endl;
    }
  }
}
```

Of the following, which best describes the behavior of `PrintSomeRestaurants` ?

(A) It prints the name of the first restaurant whose meal price is below the average of all restaurants.

(B) It prints the name of the first restaurant whose capacity is at least 50 and whose meal price is less than the average for the meal prices for all restaurants.

(C) It prints the names of all restaurants whose capacity is at least 50 and whose meal price is below the average for the meal prices for all restaurants.

(D) It prints the average of the meal prices of all restaurants.

(E) It prints the average of the meal prices of all restaurants whose capacity is at least 50.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

39. Consider the following code segment.

```
x = !y;
y = !x;
```

Assume that `x` and `y` are initialized variables of type `bool`. Which of the following statements is (are) true?

I. The final value of `x` is the same as the initial value of `x`.
II. The final value of `y` is the same as the initial value of `y`.
III. The final value of `x` is the same as the initial value of `y`.

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

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

40. Assume that functions `RowSum` and `ColSum`, declared below, have been implemented correctly.

```
int RowSum(const apmatrix<int> & A, int k);
// precondition:  A is a square matrix, 0 ≤ k < A.numrows()
// postcondition: returns the sum of the elements in
//                row k of array A

int ColSum(const apmatrix<int> & A, int k);
// precondition:  A is a square matrix, 0 ≤ k < A.numcols()
// postcondition: returns the sum of the elements in
//                column k of array A
```

Consider the partially written function `MagicSquare`, shown below. `MagicSquare` should return `true` if and only if every row and every column of its parameter `A` sums to the same value.

```
bool MagicSquare(const apmatrix<int> & A)
// precondition: A is a square matrix
{
  int k;
  int sum = RowSum(A, 0);

  for (k = 0; k < A.numrows(); k++)
  {
    if (<condition>)
    {
      <statement 1>
    }
  }
  <statement 2>
}
```

Which of the following could be used to replace *<condition>*, *<statement 1>*, and *<statement 2>* so that `MagicSquare` will work as intended?

	<condition>	*<statement 1>*	*<statement 2>*
(A)	`((RowSum(A,k) != sum) || (ColSum(A,k) != sum))`	`return(false);`	`return(true);`
(B)	`((RowSum(A,k) != sum) && (ColSum(A,k) != sum))`	`return(false);`	`return(true);`
(C)	`((RowSum(A,k) != sum) && (ColSum(A,k) != sum))`	`return(true);`	`return(false);`
(D)	`((RowSum(A,k) == sum) || (ColSum(A,k) == sum))`	`return(true);`	`return(false);`
(E)	`((RowSum(A,k) == sum) && (ColSum(A,k) == sum))`	`return(true);`	`return(false);`

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

END OF SECTION I

IF YOU FINISH BEFORE TIME IS CALLED, YOU MAY CHECK YOUR WORK ON THIS SECTION.

DO NOT GO ON TO SECTION II UNTIL YOU ARE TOLD TO DO SO.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

41. Approximately how many class periods did you spend using the Large Integer case study?

    (A) 0
    (B) 1-2
    (C) 3-5
    (D) 6-10
    (E) More than 10

42. Approximately how many hours did you spend working on the computer on problems related to the Large Integer case study?

    (A) 0
    (B) 1-2
    (C) 3-5
    (D) 6-10
    (E) More than 10

43. At what time during the school year did you use the Large Integer case study?

    (A) Did not use it.
    (B) Used it right before the AP examination.
    (C) Used it only in the middle of the year.
    (D) Used it only at the beginning of the year.
    (E) Used it at multiple times during the year.

44. How many students are in your AP Computer Science class?

    (A) Independent study
    (B) 2-5
    (C) 6-10
    (D) 11-20
    (E) More than 20

Quick Reference for apstring

```
extern const int npos;  // used to indicate not a position in the string

// public member functions

  // constructors/destructor
  apstring();                           // construct empty string ""
  apstring(const char * s);             // construct from string literal
  apstring(const apstring & str);       // copy constructor
  ~apstring();                          // destructor

  // assignment
  const apstring & operator= (const apstring & str); // assign str
  const apstring & operator= (const char * s);       // assign s
  const apstring & operator= (char ch);              // assign ch

  // accessors
  int length() const;                         // number of chars
  int find(const apstring & str) const;       // index of first occurrence of str
  int find(char ch) const;                    // index of first occurrence of ch
  apstring substr(int pos, int len) const;    // substring of len chars, starting at pos
  const char * c_str() const;                 // explicit conversion to char *

  // indexing
  char operator[ ](int k) const;  // range-checked indexing
  char & operator[ ](int k);      // range-checked indexing

  // modifiers
  const apstring & operator+= (const apstring & str); // append str
  const apstring & operator+= (char ch);              // append char


// The following free (non-member) functions operate on strings

// I/O functions
ostream & operator<< ( ostream & os, const apstring & str );
istream & operator>> ( istream & is, apstring & str );
istream & getline( istream & is, apstring & str );

// comparison operators
bool operator== ( const apstring & lhs, const apstring & rhs );
bool operator!= ( const apstring & lhs, const apstring & rhs );
bool operator<  ( const apstring & lhs, const apstring & rhs );
bool operator<= ( const apstring & lhs, const apstring & rhs );
bool operator>  ( const apstring & lhs, const apstring & rhs );
bool operator>= ( const apstring & lhs, const apstring & rhs );

// concatenation operator +
apstring operator+ ( const apstring & lhs, const apstring & rhs );
apstring operator+ ( char ch, const apstring & str );
apstring operator+ ( const apstring & str, char ch );
```

Quick Reference for apvector and apmatrix

```
template <class itemType>
class apvector

  // public member functions

    // constructors/destructor
    apvector();                                          // default constructor (size==0)
    apvector(int size);                                  // initial size of vector is size
    apvector(int size, const itemType & fillValue);      // all entries == fillValue
    apvector(const apvector & vec);                      // copy constructor
    ~apvector();                                         // destructor

    // assignment
    const apvector & operator= (const apvector & vec);

    // accessors
    int length() const;                                  // capacity of vector

    // indexing
    itemType & operator[ ](int index);                   // indexing with range checking
    const itemType & operator[ ](int index) const;       // indexing with range checking

    // modifiers
    void resize(int newSize); // change size dynamically; can result in losing values
```

```
template <class itemType>
class apmatrix

  // public member functions

    // constructors/destructor
    apmatrix();                                                     // default size is 0 x 0
    apmatrix(int rows, int cols);                                   // size is rows x cols
    apmatrix(int rows, int cols, const itemType & fillValue);       // all entries == fillValue
    apmatrix(const apmatrix & mat);                                 // copy constructor
    ~apmatrix( );                                                   // destructor

    // assignment
    const apmatrix & operator = (const apmatrix & rhs);

    // accessors
    int numrows() const;                                    // number of rows
    int numcols() const;                                    // number of columns

    // indexing
    const apvector<itemType> & operator[ ](int k) const;    // range-checked indexing
    apvector<itemType> & operator[ ](int k);                // range-checked indexing

    // modifiers
    void resize(int newRows, int newCols);    // resizes matrix to newRows x newCols
                                              // (can result in losing values)
```

Header File for the BigInt class

```
#ifndef _BIGINT_H
#define _BIGINT_H

//
// implements an arbitrary precision integer class
//
// constructors:
//
// BigInt()             -- default constructor, value of integers is 0
// BigInt(int n)        -- initialize to value of n (C++ int)
// BigInt(const apstring & s) -- initialize to value specified by s
//        it is an error if s is an invalid integer, e.g.,
//        "1234abc567".  In this case the bigint value is garbage
//
//
// *****  arithmetic operators:
//
// all arithmetic operators +, -, * are overloaded both
// in form +=, -=, *= and as binary operators
//
// multiplication also overloaded for *= int
// e.g., BigInt a *= 3 (mostly to facilitate implementation)
//
//  ***** logical operators:
//
// bool operator == (const BigInt & lhs, const BigInt & rhs)
// bool operator != (const BigInt & lhs, const BitInt & rhs)
// bool operator <  (const BigInt & lhs, const BigInt & rhs)
// bool operator <= (const BigInt & lhs, const BigInt & rhs)
// bool operator >  (const BigInt & lhs, const BigInt & rhs)
// bool operator >= (const BigInt & lhs, const BigInt & rhs)
//
//  ***** I/O operators:
//
//  void Print()
//        prints value of BigInt (member function)
//  ostream & operator << (ostream & os, const BigInt & b)
//        stream operator to print value
//
// istream & operator >> (istream & in, const BigInt & b)
//        reads whitespace delimited BigInt from input stream in
//
```

```
#include <iostream.h>
#include "apstring.h"                  // for strings
#include "apvector.h"                  // for sequence of digits

class BigInt
{
  public:

    BigInt();                          // default constructor, value = 0
    BigInt(int);                       // assign an integer value
    BigInt(const apstring &);          // assign a string

    // may need these in alternative implementation

    // BigInt(const BigInt &);         // copy constructor
    // ~BigInt();                      // destructor
    // const BigInt & operator = (const BigInt &); // assignment operator

    // operators: arithmetic, relational

    const BigInt & operator += (const BigInt &);
    const BigInt & operator -= (const BigInt &);
    const BigInt & operator *= (const BigInt &);
    const BigInt & operator *= (int num);

    apstring ToString() const;         // convert to string
    int      ToInt()    const;         // convert to int
    double   ToDouble() const;         // convert to double

    // facilitate operators ==, <, << without friends

    bool Equal(const BigInt & rhs)          const;
    bool LessThan(const BigInt & rhs)       const;
    void Print(ostream & os)                const;

  private:

    // other helper functions

    bool IsNegative()  const;          // return true iff number is negative
    bool IsPositive()  const;          // return true iff number is positive
    int NumDigits()    const;          // return # digits in number

    int GetDigit(int k) const;
    void AddSigDigit(int value);
    void ChangeDigit(int k, int value);

    void Normalize();

    // private state/instance variables

    enum Sign{positive,negative};
    Sign mySign;                            // is number positive or negative
    apvector<char> myDigits;           // stores all digits of number
    int myNumDigits;                        // stores # of digits of number
};
```

```
// free functions

ostream & operator <<(ostream &, const BigInt &);
istream & operator >>(istream &, BigInt &);

BigInt operator +(const BigInt & lhs, const BigInt & rhs);
BigInt operator -(const BigInt & lhs, const BigInt & rhs);
BigInt operator *(const BigInt & lhs, const BigInt & rhs);
BigInt operator *(const BigInt & lhs, int num);
BigInt operator *(int num, const BigInt & rhs);

bool operator == (const BigInt & lhs, const BigInt & rhs);
bool operator <  (const BigInt & lhs, const BigInt & rhs);
bool operator != (const BigInt & lhs, const BigInt & rhs);
bool operator >  (const BigInt & lhs, const BigInt & rhs);
bool operator >= (const BigInt & lhs, const BigInt & rhs);
bool operator <= (const BigInt & lhs, const BigInt & rhs);


#endif   // _BIGINT_H not defined
```

Index of functions in the BigInt class

BigInt::BigInt() 53
BigInt::BigInt(int num) 53
BigInt::BigInt(const apstring & s) 55
const BigInt & BigInt::operator -=(const BigInt & rhs) 57
const BigInt & BigInt::operator +=(const BigInt & rhs) 59
BigInt operator +(const BigInt & lhs, const BigInt & rhs) 59
BigInt operator -(const BigInt & lhs, const BigInt & lhs) 59
void Bigint::Print(ostream & os) const 61
apstring BigInt::ToString() const 61
int BigInt::ToInt() const 61
double BigInt::ToDouble() const 61
ostream & operator <<(ostream & out, const BigInt & big) 63
istream & operator >>(istream & in, BigInt & big) 63
bool operator ==(const BigInt & lhs, const BigInt & rhs) 63
bool BigInt::Equal(const BigInt & rhs) const 63
bool operator !=(const BigInt &lhs, const BigInt & rhs) 63
bool operator <(const BigInt & lhs, const BigInt & rhs) 63
bool BigInt:: LessThan(const BigInt &rhs) const 65
bool operator >(const BigInt & lhs, const BigInt & rhs) 65
bool operator <=(const BigInt & lhs, const BigInt & rhs) 65
bool operator >=(const BigInt & lhs, const BigInt & rhs) 65
void BigInt::Normalize() 65
const BigInt & BigInt:: operator *=(int num) 67
BigInt operator *(const BigInt & a, int num) 67
BigInt operator *(int num, const BigInt & a) 67
const BigInt & BigInt::operator *=(const BigInt & rhs) 69
BigInt operator *(const BigInt & lhs, const BigInt & rhs) 69
int BigInt::NumDigits() const 69
int BigInt::GetDigits() const 69
void BigInt::ChangeDigit(int k, int value) 69
void BigInt::AddSigDigit(int value) 71
bool BigInt::IsNegative() const 71
bool BigInt::IsPositive() const 71

Implementation of BigInt

```
#include <iostream.h>
#include <stdlib.h>
#include <ctype.h>
#include <limits.h>
#include "bigint.h"
#include "apvector.h"

const int BASE = 10;

//
// BigInts are implemented using a Vector<char> to store
// the digits of a BigInt
// Currently a number like 5,879 is stored as the vector {9,7,8,5}
// i.e., the least significant digit is the first digit in the vector;
// for example, GetDigit(0) returns 9 and getDigit(3) returns 5.
// All operations on digits should be done using private
// helper functions:
//
// int  GetDigit(k)         -- return k-th digit
// void ChangeDigit(k,val)  -- set k-th digit to val
// void AddSigDigit(val)    -- add new most significant digit val
//
// by performing all ops in terms of these private functions we
// make implementation changes simpler
//
// I/O operations are facilitated by the ToString() member function
// which converts a BigInt to its string (ASCII) representation


BigInt::BigInt()
// postcondition: bigint initialized to 0
   : mySign(positive),
     myDigits(1,'0'),
     myNumDigits(1)
{
   // all fields initialized in initializer list
}

BigInt::BigInt(int num)
// postcondition: bigint initialized to num
   : mySign(positive),
     myDigits(1,'0'),
     myNumDigits(0)
{
    // check if num is negative, change state and num accordingly

    if (num < 0)
    {
        mySign = negative;
        num = -1 * num;
    }


    // handle least-significant digit of num (handles num == 0)

    AddSigDigit(num % BASE);
    num = num / BASE;

    // handle remaining digits of num

    while (num != 0)
    {
        AddSigDigit(num % BASE);
        num = num / BASE;
    }
}
```

```
BigInt::BigInt(const apstring & s)
// precondition: s consists of digits only, optionally preceded by + or -
// postcondition: Bigint initialized to integer represented by s
//                if s is not a well-formed BigInt (e.g., contains non-digit
//                characters) then an error message is printed and abort called
  : mySign(positive),
    myDigits(s.length(),'0'),
    myNumDigits(0)
{
    int k;
    int limit = 0;

    if (s.length() == 0)
    {
        myDigits.resize(1);
        AddSigDigit(0);
        return;
    }
    if (s[0] == '-')
    {
        mySign = negative;
        limit = 1;
    }
    if (s[0] == '+')
    {
        limit = 1;
    }
    // start at least significant digit

    for(k=s.length() - 1; k >= limit; k--)
    {
        if (! isdigit(s[k]))
        {
            cerr << "badly formed BigInt value = " << s << endl;
            abort();
        }
        AddSigDigit(s[k]-'0');
    }
    Normalize();
}
```

```
const BigInt & BigInt::operator -=(const BigInt & rhs)
// postcondition: returns value of bigint - rhs after subtraction
{
    int diff;
    int borrow = 0;

    int k;
    int len = NumDigits();

    if (this == &rhs)          // subtracting self?
    {
        *this = 0;
        return *this;
    }

    // signs opposite? then lhs - (-rhs) = lhs + rhs

    if (IsNegative() != rhs.IsNegative())
    {
        *this +=(-1 * rhs);
        return *this;
    }
    // signs are the same, check which number is larger
    // and switch to get larger number "on top" if necessary
    // since sign can change when subtracting
    // examples: 7 - 3 no sign change,     3 - 7 sign changes
    //          -7 - (-3) no sign change, -3 -(-7) sign changes
    if (IsPositive() && (*this) < rhs ||
        IsNegative() && (*this) > rhs)
    {
        *this = rhs - *this;
        if (IsPositive()) mySign = negative;
        else              mySign = positive;    // toggle sign
        return *this;
    }
    // same sign and larger number on top

    for(k=0; k < len; k++)
    {
        diff = GetDigit(k) - rhs.GetDigit(k) - borrow;
        borrow = 0;
        if (diff < 0)
        {
            diff += 10;
            borrow = 1;
        }
        ChangeDigit(k,diff);
    }
    Normalize();
    return *this;
}
```

```
const BigInt & BigInt::operator +=(const BigInt & rhs)
// postcondition: returns value of bigint + rhs after addition
{

    int sum;
    int carry = 0;

    int k;
    int len = NumDigits();           // length of larger addend

    if (this == &rhs)                // to add self, multiply by 2
    {
      *this *= 2;
      return *this;
    }

    if (rhs == 0)                     // zero is special case
    {
      return *this;
    }

    if (IsPositive() != rhs.IsPositive())    // signs not the same, subtract
    {
        *this -= (-1 * rhs);
        return *this;
    }

    // process both numbers until one is exhausted

    if (len < rhs.NumDigits())
    {
        len = rhs.NumDigits();
    }
    for(k=0; k < len; k++)
    {
        sum = GetDigit(k) + rhs.GetDigit(k) + carry;
        carry = sum / BASE;
        sum = sum % BASE;

        if (k < myNumDigits)
        {
            ChangeDigit(k,sum);
        }
        else
        {
            AddSigDigit(sum);
        }
    }
    if (carry != 0)
    {
        AddSigDigit(carry);
    }
    return *this;
}

BigInt operator +(const BigInt & lhs, const BigInt & rhs)
// postcondition: returns a bigint whose value is lhs + rhs
{
    BigInt result(lhs);
    result += rhs;
    return result;
}

BigInt operator -(const BigInt & lhs, const BigInt & rhs)
// postcondition: returns a bigint whose value is lhs - rhs
{
    BigInt result(lhs);
    result -= rhs;
    return result;
}
```

```
void BigInt::Print(ostream & os) const
// postcondition: BigInt inserted onto stream os
{
    os << ToString();
}

apstring BigInt::ToString() const
// postcondition: returns string equivalent of BigInt
{
    int k;
    int len = NumDigits();
    apstring s = "";

    if (IsNegative())
    {
        s = '-';
    }
    for(k=len-1; k >= 0; k--)
    {
        s += char('0'+GetDigit(k));
    }
    return s;
}

int BigInt::ToInt() const
// precondition: INT_MIN <= self <= INT_MAX
// postcondition: returns int equivalent of self
{
    int result = 0;
    int k;
    if (INT_MAX < *this) return INT_MAX;
    if (*this < INT_MIN) return INT_MIN;

    for(k=NumDigits()-1; k >= 0; k--)
    {
        result = result * 10 + GetDigit(k);
    }
    if (IsNegative())
    {
        result *= -1;
    }
    return result;
}

double BigInt::ToDouble() const
// precondition: DBL_MIN <= self <= DLB_MAX
// postcondition: returns double equivalent of self
{
    double result = 0.0;
    int k;
    for(k=NumDigits()-1; k >= 0; k--)
    {
        result = result * 10 + GetDigit(k);
    }
    if (IsNegative())
    {
        result *= -1;
    }
    return result;
}
```

```
ostream & operator <<(ostream & out, const BigInt & big)
// postcondition: big inserted onto stream out
{
    big.Print(out);
    return out;
}

istream & operator >> (istream & in, BigInt & big)
// postcondition: big extracted from in, must be whitespace delimited
{
    apstring s;
    in >> s;
    big = BigInt(s);
    return in;
}

bool operator == (const BigInt & lhs, const BigInt & rhs)
// postcondition: returns true if lhs == rhs, else returns false
{
    return lhs.Equal(rhs);
}

bool BigInt::Equal(const BigInt & rhs) const
// postcondition: returns true if self == rhs, else returns false
{

    if (NumDigits() != rhs.NumDigits() || IsNegative() != rhs.IsNegative())
    {
        return false;
    }
    // assert: same sign, same number of digits;

    int k;
    int len = NumDigits();
    for(k=0; k < len; k++)
    {
        if (GetDigit(k) != rhs.GetDigit(k)) return false;
    }

    return true;
}

bool operator != (const BigInt & lhs, const BigInt & rhs)
// postcondition: returns true if lhs != rhs, else returns false
{
    return ! (lhs == rhs);
}

bool operator < (const BigInt & lhs, const BigInt & rhs)
// postcondition: return true if lhs < rhs, else returns false
{
    return lhs.LessThan(rhs);
}
```

```
bool BigInt::LessThan(const BigInt & rhs) const
// postcondition: return true if self < rhs, else returns false
{
    // if signs aren't equal, self < rhs only if self is negative

    if (IsNegative() != rhs.IsNegative())
    {
        return IsNegative();
    }

    // if # digits aren't the same must check # digits and sign

    if (NumDigits() != rhs.NumDigits())
    {
        return (NumDigits() < rhs.NumDigits() && IsPositive()) ||
               (NumDigits() > rhs.NumDigits() && IsNegative());
    }

    // assert: # digits same, signs the same

    int k;
    int len = NumDigits();

    for(k=len-1; k >= 0; k--)
    {
        if (GetDigit(k) < rhs.GetDigit(k)) return IsPositive();
        if (GetDigit(k) > rhs.GetDigit(k)) return IsNegative();
    }
    return false;      // self == rhs
}

bool operator > (const BigInt & lhs, const BigInt & rhs)
// postcondition: return true if lhs > rhs, else returns false
{
    return rhs < lhs;
}

bool operator <= (const BigInt & lhs, const BigInt & rhs)
// postcondition: return true if lhs <= rhs, else returns false
{
    return lhs == rhs || lhs < rhs;
}

bool operator >= (const BigInt & lhs, const BigInt & rhs)
// postcondition: return true if lhs >= rhs, else returns false
{
    return lhs == rhs || lhs > rhs;
}

void BigInt::Normalize()
// postcondition: all leading zeros removed
{
    int k;
    int len = NumDigits();
    for(k=len-1; k >= 0; k--)          // find a non-zero digit
    {
        if (GetDigit(k) != 0) break;
        myNumDigits--;                 // "chop" off zeros
    }
    if (k < 0)     // all zeros
    {
        myNumDigits = 1;
        mySign = positive;
    }
}
```

```
const BigInt & BigInt::operator *=(int num)
// postcondition: returns num * value of BigInt after multiplication
{
    int carry = 0;
    int product;                  // product of num and one digit + carry
    int k;
    int len = NumDigits();

    if (0 == num)                 // treat zero as special case and stop
    {
        *this = 0;
        return *this;
    }

    if (BASE < num|| num < 0)           // handle pre-condition failure
    {
        *this *= BigInt(num);
        return *this;
    }

    if (1 == num)                 // treat one as special case, no work
    {
        return *this;
    }

    for(k=0; k < len; k++)        // once for each digit
    {
        product = num * GetDigit(k) + carry;
        carry = product/BASE;
        ChangeDigit(k,product % BASE);
    }

    while (carry != 0)            // carry all digits
    {
        AddSigDigit(carry % BASE);
        carry /= BASE;
    }
    return *this;
}


BigInt operator *(const BigInt & a, int num)
// postcondition: returns a * num
{
    BigInt result(a);
    result *= num;
    return result;
}

BigInt operator *(int num, const BigInt & a)
// postcondition: returns num * a
{
    BigInt result(a);
    result *= num;
    return result;
}
```

```
const BigInt & BigInt::operator *=(const BigInt & rhs)
// postcondition: returns value of bigint * rhs after multiplication
{
    // uses standard "grade school method" for multiplying

   if (IsNegative() != rhs.IsNegative())
   {
      mySign = negative;
   }
   else
   {
      mySign = positive;
   }

   BigInt self(*this);                            // copy of self
   BigInt sum(0);                                 // to accumulate sum
   int k;
   int len = rhs.NumDigits();                     // # digits in multiplier

   for(k=0; k < len; k++)
   {
      sum += self * rhs.GetDigit(k);              // k-th digit * self
      self *= 10;                                 // add a zero
   }
   *this = sum;
   return *this;
}

BigInt operator *(const BigInt & lhs, const BigInt & rhs)
// postcondition: returns a bigint whose value is lhs * rhs
{
    BigInt result(lhs);
    result *= rhs;
    return result;
}

int BigInt::NumDigits() const
// postcondition: returns # digits in BigInt
{
    return myNumDigits;
}

int BigInt::GetDigit(int k) const
// precondition: 0 <= k < NumDigits()
// postcondition: returns k-th digit
//                (0 if precondition is false)
//                Note: 0th digit is least significant digit
{
    if (0 <= k && k < NumDigits())
    {
        return myDigits[k] - '0';
    }
    return 0;
}

void BigInt::ChangeDigit(int k, int value)
// precondition: 0 <= k < NumDigits()
// postcondition: k-th digit changed to value
//                Note: 0th digit is least significant digit
{
    if (0 <= k && k < NumDigits())
    {
        myDigits[k] = char('0' + value);
    }
    else
    {
        cerr << "error changeDigit " << k << " " << myNumDigits << endl;
    }
}
```

```
void BigInt::AddSigDigit(int value)
// postcondition: value added to BigInt as most significant digit
//                Note: 0th digit is least significant digit
{
    if (myNumDigits >= myDigits.length())
    {
        myDigits.resize(myDigits.length() * 2);
    }
    myDigits[myNumDigits] = char('0' + value);
    myNumDigits++;
}

bool BigInt::IsNegative() const
// postcondition: returns true iff BigInt is negative
{
    return mySign == negative;
}

bool BigInt::IsPositive() const
// postcondition: returns true iff BigInt is positive
{
   return mySign == positive;
}
```

COMPUTER SCIENCE A
SECTION II
Time—1 hour and 45 minutes
Number of questions—4
Percent of total grade—50

Some questions in the free-response section require you to write program segments. These are to be written in C++.

The questions are printed in this booklet and on the green insert. You are to use the green insert only to organize your responses and for scratchwork, but you must write all your answers in the pink booklet. Write your answers in pencil only. Be sure to write CLEARLY and LEGIBLY. If you make an error, you may save time by crossing it out rather than trying to erase it. All questions are given equal weight. Credit for partial solutions will be given. Do not spend too much time on any one problem.

When you are told to begin, open your booklet, carefully tear out the green insert, and start to work.

DO NOT OPEN THIS BOOKLET UNTIL YOU ARE TOLD TO DO SO.

Copyright © 1999 College Entrance Examination Board and Educational Testing Service. All rights reserved.

COMPUTER SCIENCE A
SECTION II

Time—1 hour and 45 minutes
Number of questions—4
Percent of total grade—50

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

Note: Assume that the standard libraries (`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. A Quick Reference to the AP C++ classes is included in the case study insert.

GO ON TO THE NEXT PAGE

1. Assume that student records are implemented using the following declaration.

```
struct StudentInfo
{
   apstring name;
   int creditHours;
   double gradePoints;
   double GPA;
};
```

(a) Write function `ComputeGPA`, as started below. `ComputeGPA` should fill in the `GPA` data member for the first `numStudents` records in its `apvector` parameter `roster`. A student's GPA (grade point average) is computed by dividing `gradePoints` by `creditHours`. The GPA for a student with 0 credit hours should be set to 0.

Complete function `ComputeGPA` below. Assume that `ComputeGPA` is called only with parameters that satisfy its precondition.

```
void ComputeGPA(apvector<StudentInfo> & roster, int numStudents)
// precondition:  roster contains numStudents records,
//                0 < numStudents ≤ roster.length(), in which the
//                name, creditHours and gradePoints data members
//                have been initialized.
// postcondition: The GPA data member for the first numStudents records
//                in roster has been calculated.
```

GO ON TO THE NEXT PAGE

(b) Write function `IsSenior`, as started below. `IsSenior` should return `true` if the given student has at least 125 credit hours and has a GPA of at least 2.0; otherwise, `IsSenior` should return `false`.

For example:

student				Result of the call `IsSenior(student)`
name	creditHours	gradePoints	GPA	
King	45	171	3.8	`false` (not enough credit hours)
Norton	128	448	3.5	`true`
Solo	125	350	2.8	`true`
Kramden	150	150	1.0	`false` (GPA too low)

Complete function `IsSenior` below.

```
bool IsSenior(const StudentInfo & student)
// postcondition: returns true if this student's credit hours ≥ 125
//                and GPA ≥ 2.0; otherwise, returns false
```

GO ON TO THE NEXT PAGE

(c) Write function `FillSeniorList`, as started below. `FillSeniorList` determines which students in the array `roster` are seniors and copies those students' records to the array `seniors`. It should also set the value of parameter `numSeniors` to be the number of seniors in the array `seniors`.

In writing `FillSeniorList`, you may call function `IsSenior` specified in part (b). Assume that `IsSenior` works as specified, regardless of what you wrote in part (b).

Complete function `FillSeniorList` below. Assume that `FillSeniorList` is called only with parameters that satisfy its precondition.

```
void FillSeniorList(const apvector<StudentInfo> & roster,
                    int numStudents, apvector<StudentInfo> & seniors,
                    int & numSeniors)
// precondition: roster contains numStudents records,
//               0 < numStudents ≤ roster.length(),
//               and seniors is large enough to hold all of
//               the seniors' records
```

GO ON TO THE NEXT PAGE

2.

(a) Write function `WordIndex`, as started below. The array `wordList` contains `numWords` strings in alphabetical order. If `word` is already in `wordList`, then `WordIndex` should return the index of `word` in `wordList`. Otherwise, `WordIndex` should return the index of the first string in `wordList` that comes after `word` in alphabetical order; it should return `numWords` if `word` comes after all of the strings in `wordList` in alphabetical order.

For example, assume that array `wordList` is as follows:

0	1	2	3
"apple"	"berry"	"pear"	"quince"

Function Call	Value Returned
`WordIndex("air", wordList, 4)`	0
`WordIndex("apple", wordList, 4)`	0
`WordIndex("orange", wordList, 4)`	2
`WordIndex("raspberry", wordList, 4)`	4

Complete function `WordIndex` below. Assume that `WordIndex` is called only with parameters that satisfy its precondition.

```
int WordIndex(const apstring & word,
              const apvector<apstring> & wordList, int numWords)
// precondition: wordList contains numWords strings in alphabetical
//               order, 0 ≤ numWords < wordList.length()
```

GO ON TO THE NEXT PAGE

(b) Write function `InsertInOrder`, as started below. The array `wordList` contains `numWords` strings in alphabetical order. If the string `word` is already in `wordList`, `InsertInOrder` should not change any of its parameters. Otherwise, it should insert `word` into `wordList` in alphabetical order (i.e., all values greater than `word` should be moved one place to the right to make room for `word`), and it should also increment `numWords` by 1. Assume that `wordList.length()` is greater than `numWords`.

In the examples below, `numWords = 3` before the following call is made.

```
InsertInOrder("pear", wordList, numWords)
```

Before the call wordList	After the call wordList	numWords
"apple" "berry" "quince"	"apple" "berry" "pear" "quince"	4
"apple" "berry" "pear"	"apple" "berry" "pear"	3
"apple" "fig" "peach"	"apple" "fig" "peach" "pear"	4
"quince" "raisin" "tart"	"pear" "quince" "raisin" "tart"	4

In writing `InsertInOrder`, you may include calls to function `WordIndex` specified in part (a). Assume that `WordIndex` works as specified, regardless of what you wrote in part (a).

Complete function `InsertInOrder` below. Assume that `InsertInOrder` is called only with parameters that satisfy its precondition.

```
void InsertInOrder(const apstring & word,
                   apvector <apstring> & wordList, int & numWords)
// precondition:  wordList contains numWords strings in alphabetical
//                order, 0 ≤ numWords < wordList.length()
// postcondition: if word was already in wordList, then wordList and
//                numWords are unchanged;
//                otherwise, word has been inserted into wordList in
//                sorted order, and numWords has been incremented by 1
```

GO ON TO THE NEXT PAGE

3. This question involves reasoning about the code from the Large Integer case study. A copy of the code is provided as part of this examination.

(a) Write a new `BigInt` member function `Div2`, as started below. `Div2` should change the value of the `BigInt` to be the original value divided by 2 (integer division). Assume the `BigInt` is greater than or equal to 0. One algorithm for implementing `Div2` is:

```
1. Initialize a variable carryDown to 0.

2. For each digit, d, starting with the most significant digit,

   2.1 replace that digit with (d / 2) + carryDown

   2.2 let carryDown be (d % 2) * 5

3. Normalize the result
```

Complete member function `Div2` below.

```
void BigInt::Div2()
// precondition:  BigInt ≥ 0
```

GO ON TO THE NEXT PAGE

(b) Write function `DivPos`, as started below. `DivPos` returns the quotient of the integer division of `dividend` by `divisor`. Assume that `dividend` and `divisor` are both positive values of type `BigInt`.

For example, assume that `bigNum1` and `bigNum2` are positive values of type `BigInt`:

bigNum1	bigNum2	DivPos(bigNum1, bigNum2)
18	9	2
17	2	8
8714	2178	4
9990	999	10

There are many ways to implement division; however, you must use a binary search algorithm to find the quotient of `dividend` divided by `divisor` in this problem. You will receive no credit on this part if you do not use a binary search algorithm.

One algorithm for implementing division using binary search is as follows:

1. Initialize `low` to `0` and `high` to `dividend`.
2. For each iteration,

   2.1 compute `mid = (low + high + 1)`

   2.2 divide `mid` by `2`

   2.3 if `mid * divisor` is larger than `dividend` (`mid` is too large to be the quotient) then set `high` equal to `mid - 1` else set `low` equal to `mid`.

3. When `low == high` the search terminates, and you should return `low`.

In writing function `DivPos`, you may call function `Div2` specified in part (a). Assume that `Div2` works as specified, regardless of what you wrote in part (a). You will receive no credit on this part if you do not use a binary search algorithm.

Complete function `DivPos` below. Assume that `DivPos` is called only with parameters that satisfy its precondition.

```
BigInt DivPos(const BigInt & dividend, const BigInt & divisor)
// precondition: dividend > 0, divisor > 0
```

GO ON TO THE NEXT PAGE

4. A patchwork quilt can be made by sewing together many blocks, all of the same size. Each individual block is made up of a number of small squares cut from fabric. A block can be represented as a two-dimensional array of nonblank characters, each of which stands for one small square of fabric. The entire quilt can also be represented as a two-dimensional array of completed blocks. The example below shows an array that represents a quilt made of 9 blocks (in 3 rows and 3 columns). Each block contains 20 small squares (of 4 rows by 5 columns). The quilt uses 2 different fabric squares, represented by the characters `'x'` and `'.'`. We consider only quilts where the main block alternates with the same block flipped upside down (i.e., reflected about a horizontal line through the block's center), as in the example below.

```
x...x|..x..|x...x
.x.x.|..x..|.x.x.
..x..|.x.x.|..x..
..x..|x...x|..x..
-----+-----+-----
..x..|x...x|..x..
..x..|.x.x.|..x..
.x.x.|..x..|.x.x.
x...x|..x..|x...x
-----+-----+-----
x...x|..x..|x...x
.x.x.|..x..|.x.x.
..x..|.x.x.|..x..
..x..|x...x|..x..
```

Consider the problem of storing and displaying information about a quilt.

The class `Quilt`, whose declaration is shown below, is used to keep track of the blocks for an entire quilt. Since the pattern is based on one block, we only store that block and the number of rows and columns of blocks. For the example shown above, we would store the upper left 4×5 block, 3 for the number of rows of blocks in the quilt and 3 for the number of columns of blocks in the quilt.

```
class Quilt
{
  public:
    Quilt(istream & inFile, int rowsOfBlocks, int colsOfBlocks);
    // constructor, given number of blocks in each row and column

    apmatrix<char> QuiltToMat();
    // returns a matrix with the entire quilt stored in it

  private:
    apmatrix<char> myBlock; // stores pattern for one block
    int myRowsOfBlocks;     // number of rows of blocks in the quilt
    int myColsOfBlocks;     // number of columns of blocks in the quilt

    void PlaceBlock(int startRow, int startCol,
                    apmatrix<char> & qmat);
    void PlaceFlipped(int startRow, int startCol,
                      apmatrix<char> & qmat);
};
```

GO ON TO THE NEXT PAGE

(a) Write the code for the constructor that initializes a quilt, as started below. The constructor reads the block pattern for the main block from a file represented by the parameter `inFile`. You may assume the file is open and that the file contains the number of rows followed by the number of columns for the block, followed by the characters representing the pattern. For example, the file `pattern`, which contains the pattern for the first block in the quilt shown above, would look like this:

```
4 5
x...x
.x.x.
..x..
..x..
```

The constructor also sets the number of rows and columns of blocks which make up the entire quilt in the initializer list.

Complete the constructor below. Assume that the constructor is called only with parameters that satisfy its precondition.

```
Quilt::Quilt(istream & inFile, int rowsOfBlocks, int colsOfBlocks)
  : myBlock(0, 0), myRowsOfBlocks(rowsOfBlocks),
    myColsOfBlocks(colsOfBlocks)
// precondition:  inFile is open, rowsOfBlocks > 0, colsOfBlocks > 0
// postcondition: myRowsOfBlocks and myColsOfBlocks are initialized to
//                the number of rows and columns of blocks that make up
//                the quilt; myBlock has been resized and
//                initialized to the block pattern from the
//                stream inFile.
```

GO ON TO THE NEXT PAGE

(b) Write the private member function `PlaceFlipped`, as started below. `PlaceFlipped` is intended to place a flipped (upside-down) version of the block into the matrix `qmat` with the flipped block's upper left corner located at the `startRow, startCol` position in `qmat`.

For example, if quilt `Q` contains the block shown in part (a) and if `M` is a matrix large enough to hold the characters in the whole quilt, then the call

```
Q.PlaceFlipped(4, 10, M)
```

would place the flipped version of `Q`'s quilt block into matrix `M` as the third block in the second row of quilt blocks. This is the block whose upper-left corner is at position `M[4][10]`. In the diagram below, the upper-left corner of the flipped block being placed into `M` is circled.

```
x...x | ..x.. | x...x
.x.x. | ..x.. | .x.x.
..x.. | .x.x. | ..x..
..x.. | x...x | ..x..
------+-------+------
..x.. | x...x | ⊙.x..
..x.. | .x.x. | ..x..
.x.x. | ..x.. | .x.x.
x...x | ..x.. | x...x
------+-------+------
x...x | ..x.. | x...x
.x.x. | ..x.. | .x.x.
..x.. | .x.x. | ..x..
..x.. | x...x | ..x..
```

You may adapt the code of the private member function `PlaceBlock`, given below, which places the block (not inverted) into the matrix `qmat` with the block's upper left corner located at the `startRow, startCol` position.

```
void Quilt::PlaceBlock(int startRow, int startCol,
                       apmatrix<char> & qmat)
// precondition:  startRow ≥ 0; startCol ≥ 0;
//                startRow + myBlock.numrows() ≤ qmat.numrows();
//                startCol + myBlock.numcols() ≤ qmat.numcols();
// postcondition: myBlock has been copied into the matrix
//                qmat with its upper-left corner at the position
//                startRow, startCol
{
  int r, c;
  for (r = 0; r < myBlock.numrows(); r++)
  {
    for (c = 0; c < myBlock.numcols(); c++)
    {
      qmat[startRow + r][startCol + c] = myBlock[r][c];
    }
  }
}
```

GO ON TO THE NEXT PAGE

Complete the member function `PlaceFlipped` below. Assume that `PlaceFlipped` is called only with parameters that satisfy its precondition.

```
void Quilt::PlaceFlipped(int startRow, int startCol,
                         apmatrix<char> & qmat)
// precondition:  startRow ≥ 0; startCol ≥ 0;
//                startRow + myBlock.numrows() ≤ qmat.numrows();
//                startCol + myBlock.numcols() ≤ qmat.numcols();
// postcondition: a flipped version of myBlock has been copied into the
//                matrix qmat with its upper-left corner at the position
//                startRow, startCol
{
  int r, c;

  for (r = 0; r < myBlock.numrows(); r++)
  {
    for (c = 0; c < myBlock.numcols(); c++)
    {

    }
  }
}
```

GO ON TO THE NEXT PAGE

(c) Write the member function `QuiltToMat`, as started below. `QuiltToMat` returns a matrix representing the whole quilt in such a way that the main block alternates with the flipped version of the main block, as shown in the original example. If `Q` represents the example quilt, then the call `Q.QuiltToMat()` would return a matrix of characters with the given block placed starting with the upper-left corner at position 0, 0; the flipped block placed with its upper-left corner at position 0, 5; the given block placed with its upper-left corner at position 0, 10; the flipped block placed with its upper-left corner at position 4, 0, and so on.

In writing `QuiltToMat`, you may call functions `PlaceBlock` and `PlaceFlipped` specified in part (b). Assume that `PlaceBlock` and `PlaceFlipped` work as specified, regardless of what you wrote in part (b).

Complete the member function `QuiltToMat` below.

```
apmatrix<char> Quilt::QuiltToMat()
```

END OF EXAMINATION

COMPUTER SCIENCE AB

CALCULATORS, REFERENCE MATERIALS, OR OTHER AIDS ARE NOT TO BE USED DURING THE EXAMINATION.

Three hours are allotted for this examination: 1 hour and 15 minutes for Section I, which consists of 40 multiple-choice questions, and 1 hour and 45 minutes for Section II, which consists of 4 free-response questions. In determining your grade, the two sections are given equal weight. Section I is printed in this examination booklet. Section II is printed in a separate booklet. In addition, following Section I there are 4 survey questions to be answered in 2 minutes.

SECTION I

Time — 1 hour and 15 minutes

Number of questions — 40

Percent of total grade — 50

Section I of this examination contains 40 multiple-choice questions. Therefore, please be careful to fill in only the ovals that are preceded by numbers 1 through 40 on your answer sheet when answering the examination questions. Also, please be careful to fill in the ovals preceded by numbers 41 through 44 when answering the survey questions.

General Instructions

DO NOT OPEN THIS BOOKLET UNTIL YOU ARE INSTRUCTED TO DO SO.

INDICATE ALL YOUR ANSWERS TO QUESTIONS IN SECTION I ON THE SEPARATE ANSWER SHEET. No credit will be given for anything written in this examination booklet, but you may use the booklet for notes or scratchwork. After you have decided which of the suggested answers is best, COMPLETELY fill in the corresponding oval on the answer sheet. Give only one answer to each question. If you change an answer, be sure that the previous mark is erased completely.

Example:

Chicago is a

(A) state
(B) city
(C) country
(D) continent
(E) village

Sample Answer

(A) ● (C) (D) (E)

Many candidates wonder whether or not to guess the answer to questions about which they are not certain. In this section of the examination, as a correction for haphazard guessing, one-fourth of the number of questions you answer incorrectly will be subtracted from the number of questions you answer correctly. It is improbable, therefore, that mere guessing will improve your score significantly; it may even lower your score, and it does take time. If, however, you are not sure of the correct answer but have some knowledge of the question and are able to eliminate one or more of the answer choices as wrong, your chance of getting the right answer is improved, and it may be to your advantage to answer such a question.

Use your time effectively, working as rapidly as you can without losing accuracy. Do not spend too much time on questions that are too difficult. Go on to other questions and come back to the difficult ones later if you have time. It is not expected that everyone will be able to answer all the multiple-choice questions.

Copyright © 1999 College Entrance Examination Board and Educational Testing Service. All rights reserved.
Certain test materials are copyrighted solely in the name of ETS.

COMPUTER SCIENCE AB

SECTION I

Time— 1 hour and 15 minutes

Number of questions—40

Percent of total grade—50

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 on pages 45-49.

USE THIS SPACE FOR SCRATCHWORK.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

1. A "strictly" binary tree is a binary tree in which every node has either 0 or 2 children (i.e., no node has exactly 1 child). How many nodes are in a strictly binary tree that has 8 leaves?

   (A) 7
   (B) 8
   (C) 15
   (D) 16
   (E) The answer cannot be determined from the information given.

GO ON TO THE NEXT PAGE

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

2. Consider the following declarations and code segment.

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

ListNode * L1;
ListNode * L2;

L1 = new ListNode;
L2 = new ListNode;
L1->next = L2;
L1->next->next = L2;
```

Which of the following diagrams best depicts the data structure created when the code segment is executed?

(A) L1, L2

(B) L1, L2

(C) L1, L2

(D) L2, L1

(E) L1, L2

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

3. Consider the following definitions.

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

apmatrix<double> A(N,N), B(N,N), C(N,N);
```

Consider the following code segment.

```
int row, col, k;
for (row = 0; row < N; row++)
{
  for (col = 0; col < N; col++)
  {
    C[row][col] = 0.0;
    for (k = 0; k < N; k++)
    {
      C[row][col] += A[row][k] * B[k][col];
    }
  }
}
```

Of the following, which best describes the running time of the code segment?

(A) $O(1)$
(B) $O(N)$
(C) $O(N^2)$
(D) $O(N^3)$
(E) $O(N^4)$

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

4. A racehorse breeder owns 100 horses, identified by ID numbers from 0 to 99. Each year there are 50 races. Consider designing a data structure to keep track of the number of races won by each horse in the current year. Two different methods are described below.

   Method 1. Use a one-dimensional array of integers: the value of the *k*th element of the array is the number of races won by the breeder's horse whose ID number is *k*.

   Method 2. Use a linked list of integers: each list element corresponds to one race won by one of the breeder's horses and contains that horse's ID number.

   Which of the following is true?

   (A) If Method 1 is used, the size of the array should be 50.
   (B) If the breeder's horses win every race, then Method 1 cannot be used.
   (C) If the breeder's horses win every race, then Method 2 cannot be used.
   (D) If the breeder's horses win <u>no</u> races, then Method 1 is more storage-efficient than Method 2.
   (E) If the breeder's horses win <u>no</u> races, then Method 2 is more storage-efficient than Method 1.

GO ON TO THE NEXT PAGE

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

5. Consider the following declarations.

```
apstack<char> S;
apqueue<char> Q;
```

Assume that `Q` is initially empty and that `S` initially contains

```
W X Y Z
↑
top
```

in that order, with `W` at the top of the stack.

Consider the following code segment.

```
char item;
while (!S.isEmpty())
{
  S.pop(item);
  Q.enqueue(item);
}
while (!Q.isEmpty())
{
  Q.dequeue(item);
  S.push(item);
}
```

Which of the following best describes queue `Q` and stack `S` after the code segment executes?

(A) Queue `Q` is empty and stack `S` contains `W, X, Y, Z` in that order, with `W` at the top of the stack.
(B) Queue `Q` is empty and stack `S` contains `Z, Y, X, W` in that order, with `Z` at the top of the stack.
(C) Queue `Q` contains `W, X, Y, Z,` in that order, with `W` at the front of the queue, and stack `S` is empty.
(D) Queue `Q` contains `W, X, Y, Z,` in that order, with `W` at the front of the queue, and stack `S` contains `Z, Y, X, W` in that order, with `Z` at the top of the stack.
(E) Queue `Q` contains `W, X, Y, Z` in that order, with `W` at the front of the queue, and stack `S` contains `W, X, Y, Z` in that order, with `W` at the top of the stack.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

6. Consider the following function.

```
int Mystery(int n)
{
  if (n <= 1)
  {
    return 0;
  }
  else
  {
    return (n + Mystery(n - 1) + Mystery(n - 2));
  }
}
```

What value is returned by the call `Mystery(4)` ?

(A) 0
(B) 4
(C) 6
(D) 9
(E) 11

GO ON TO THE NEXT PAGE

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

7. Consider the following definition of a stack of integers.

```
class IntStack
{
  public:
    IntStack();         // constructor, initializes to empty
                        // stack
    void Push(int x);   // pushes x onto the stack
    int Pop();          // removes and returns the top element
                        // from the stack
    bool isEmpty() const;  // returns true if the stack is
                           // empty; otherwise, returns false

  <designation>:
    apvector<int> myStack;        // stack items
    int myCount;                  // number of items on the stack
};
```

Which of the following is the best choice for `<designation>` and the best reason for that choice?

(A) `<designation>` should be `private`. Otherwise, the client program will not be able to modify the stack using member functions `Push` and `Pop`.

(B) `<designation>` should be `private` so that the stack can be modified only by using member functions such as `Push` and `Pop`, thereby preserving the principle of information hiding.

(C) `<designation>` should be `public` because the programmer of a client program should know how the `IntStack` class has been implemented.

(D) `<designation>` should be `public` so that a client program can include code such as `if (myCount == 0)` to determine whether the stack is empty, thereby preserving the principle of maximum flexibility.

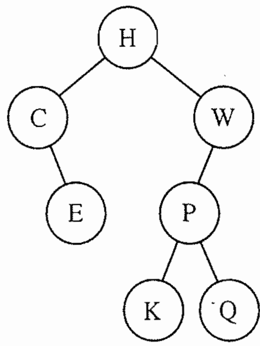
(E) `<designation>` should be `public`. Otherwise, a client program will not be able to modify the contents of the stack in any way.

GO ON TO THE NEXT PAGE

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

8. Consider the binary search tree shown below.



This tree could be the result of inserting letters into an empty binary search tree in any of the following orders EXCEPT

(A) H C E W K P Q
(B) H C E W P K Q
(C) H W P C E Q K
(D) H W P K C E Q
(E) H W P Q C K E

GO ON TO THE NEXT PAGE

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

Questions 9-10 refer to the following information.

An office building is 50 stories high and each story contains ten offices. Some of the offices have no computers, others have one or more computers. Consider the problem of representing the locations of the computers. Three alternative data structures to represent this information are defined below.

Method 1. Use a two-dimensional array `A` of integer values indexed by story number and office number. The number of computers on story `j` in office `k` is stored in location `A[j][k]`.

Method 2. Use a linked list of structures. Each structure has three data members: the story number and the office number of an office and the number of computers in the office. Offices without a computer would not appear in this list.

Method 3. Use a linked list of structures. Each structure has two data members: the story number and the office number of an office that has a computer. If an office has multiple computers, that office will appear as many times in the list as it has computers. Offices without a computer would not appear in this list.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

9. Which of the following statements about the relative space requirements of the three methods is true?

(A) When every office in the building has five computers, Method 2 requires more space than does Method 3, which requires more space than does Method 1.
(B) When every office in the building has five computers, Method 3 requires more space than does Method 2, which requires more space than does Method 1.
(C) When no office in the building has a computer, Method 1 requires more space than does Method 3, which requires more space than does Method 2.
(D) When no office in the building has a computer, Method 3 requires more space than does Method 2, which requires more space than does Method 1.
(E) All three methods require the same amount of space regardless of how many computers are currently in the building.

10. Which of the following operations can be performed more efficiently using Method 1 than using either of the other two methods?

I. Determine whether a particular office currently has a computer (given the story number and the office number of the office).
II. Move a computer from one office to another (given the story number and the office number of each of the two offices).
III. Determine the total number of computers currently in the building.

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

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

11. Assume that binary trees are implemented using the following declaration and that `Height` is a function that operates on binary trees.

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

int Height(TreeNode * T);
// postcondition: returns the length of the longest path
//                from the root to a leaf, or 0 if the tree
//                is empty
```

A "full complete binary tree" is one in which every level of the tree is completely filled, i.e., every nonleaf node has two children and both children are full complete binary trees of equal size. (An empty tree is considered a full complete tree.) For example, T1 below is a full complete binary tree but T2 below is not.

T1 T2

The height of an empty tree is 0; the height of a nonempty tree is the number of nodes on the longest path from the root to a leaf.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

Consider the following incomplete function.

```
bool IsFullComplete(TreeNode * T)
// postcondition: returns true if tree T is a full complete
//                binary tree; otherwise, returns false
{
  if (T == NULL)
  {
    return true;
  }
  else
  {
    <statement>
  }
}
```

Assume that function `Height` works as specified. Which of the following can be used to replace `<statement>` in function `IsFullComplete` so that `IsFullComplete` will satisfy its postcondition?

(A) `return (IsFullComplete(T->left) && IsFullComplete(T->right));`

(B) `return (IsFullComplete(T->left) || IsFullComplete(T->right));`

(C)
```
return ((IsFullComplete(T->left) && IsFullComplete(T->right)) &&
        (Height(T->left) && Height(T->right)));
```

(D)
```
return ((IsFullComplete(T->left) && IsFullComplete(T->right)) &&
        (Height(T->left) == Height(T->right)));
```

(E)
```
return ((IsFullComplete(T->left) && IsFullComplete(T->right)) ||
        (Height(T->left) == Height(T->right)));
```

GO ON TO THE NEXT PAGE

Unauthorized copying or reusing
any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

12. A sparse array is one that contains mostly zeroes. If a sparse array is very large, there may not be enough computer memory to store the entire array; instead, a special `SparseArray` data structure which stores only the nonzero values can be used. Consider the following two designs for the `SparseArray` data structure.

    Design 1. A `SparseArray` is represented by using a linked list in which each node contains the index and the value of one nonzero array entry. The linked list is maintained in sorted order by indices.

    Design 2. A `SparseArray` is represented by using a binary search tree in which each node contains the index and the value of one nonzero array entry. The binary search tree is organized according to the indices stored in the tree nodes; that is, the index at node `p` is greater than all of the indices in `p`'s left subtree and is less than all of the indices in `p`'s right subtree.

    Assume that the nonzero values of the sparse array are inserted in increasing order of their index into the `SparseArray` data structure.

    Which of the following correctly describes an advantage of Design 2 over Design 1 ?

    (A) In general, Design 2 will take less time than Design 1 to initialize an array that contains all zeroes.
    (B) In general, Design 2 will take less time than Design 1 to access a particular element with index `k`.
    (C) Design 2 will require less space than Design 1 to represent an array that contains all zeroes.
    (D) Design 2 will require less space than Design 1 to represent an array that contains some nonzero entries.
    (E) Design 2 has no advantage over Design 1.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

13. Which of the following properties of a C++ program CANNOT always be checked at compile time?

    (A) No `NULL` pointer is ever dereferenced.
    (B) No function is called with the wrong number of arguments.
    (C) Every `"{"` has a matching `"}"`.
    (D) Every nonvoid function includes a return statement.
    (E) Private members of a class are accessed only by member functions of that class.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

14. Consider using quicksort to sort an array of integers into ascending order. Recall that quicksort first partitions the array about an element and then recursively sorts the two partitions. One way of choosing the partition value is to always use the left-most (first) element in the subarray being partitioned.

    Consider what happens when the version of quicksort defined above is applied to each of the following arrays:

    1. A list whose values are in ascending order
    2. A list whose values are in descending order
    3. A list whose values are in random order

    Of the following, which best characterizes the expected runtime of quicksort when applied to the three lists defined above?

    (A) The expected runtime of quicksort will be the same for all three lists.
    (B) The expected runtime of quicksort for List 1 and List 2 will be the same and will be slower than the runtime of quicksort for List 3.
    (C) The expected runtime of quicksort for List 1 and List 2 will be the same and will be faster than the runtime of quicksort for List 3.
    (D) The expected runtime of quicksort for List 2 and List 3 will be the same and will be slower than the runtime of quicksort for List 1.
    (E) The expected runtime of quicksort for List 2 and List 3 will be the same and will be faster than the runtime of quicksort for List 1.

GO ON TO THE NEXT PAGE

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

15. The following integers are read in the order indicated and inserted into a binary search tree with smaller elements inserted in the left subtree and larger elements inserted in the right subtree.

```
15  12  27  3  10  14  33  18
```

What would be output if this tree was printed using a postorder traversal?

(A) 3 10 12 14 15 18 27 33
(B) 10 3 14 12 18 33 27 15
(C) 10 3 14 18 33 12 27 15
(D) 15 12 3 10 14 27 18 33
(E) 33 27 18 15 14 12 10 3

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

Questions 16-20 refer to the code from the Large Integer case study. A copy of the code is provided as part of this exam.

16. In the `BigInt` class, functions `Equal` and `LessThan` are the only comparison functions implemented as public member functions. Which of the following best describes the reason for this implementation decision?

    (A) The author intended for the client to implement other comparison functions.
    (B) It is impossible to implement the other comparison functions as public member functions.
    (C) It is impossible to implement relational operators as public member functions.
    (D) All the relational operators can be defined as free functions in terms of `Equal` and `LessThan`.
    (E) The other comparison functions are not necessary because a `BigInt` is an array of characters.

17. If a linked list were used to store the digits of a `BigInt`, which of the following member functions would need to be changed?

    (A) `IsNegative`
    (B) `LessThan`
    (C) `Equal`
    (D) `ChangeDigit`
    (E) `ToString`

GO ON TO THE NEXT PAGE

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

18. Suppose we want to add an additional constructor to create a `BigInt` consisting of a positive number in which every digit is the same. This constructor would have two parameters: `digit`, a nonzero digit and `occur`, the number of times the digit occurs. For example, `BigInt A('9', 4)` would result in `A` having the value `9999`. The precondition of this constructor specifies that `digit` is a character between `'1'` and `'9'` inclusive and `occur` is an integer greater than `0`.

Consider the following constructor definitions.

```
I.  BigInt::BigInt(char digit, int occur)
      : mySign(positive), myDigits(occur, digit), myNumDigits(occur)
    {
    }

II. BigInt::BigInt(char digit, int occur)
      : mySign(positive), myDigits(1, digit), myNumDigits(1)
    {
      while (occur > 1)
      {
        AddSigDigit(GetDigit(0));
        occur--;
      }
    }

III. BigInt::BigInt(char digit, int occur)
      : mySign(positive), myDigits (1, '0'), myNumDigits(1)
    {
      while (occur > 1)
      {
        AddSigDigit(GetDigit(0));
        occur--;
      }
    }
```

Which of these constructors will work correctly?

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

GO ON TO THE NEXT PAGE

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

19. Consider adding to the `BigInt` class an additional `operator=` member function that sets a `BigInt` value equal to the value represented by the `apstring rhs`. An incomplete function definition is given below.

```
const BigInt & BigInt::operator= (const apstring & rhs)
// precondition:  rhs consists of digits only,
//                optionally preceded by + or -
// postcondition: BigInt is set to the integer
//                represented by rhs.
{
    <additional code>
}
```

Which of the following code segments can be used to replace *<additional code>* so that `operator=` will work as intended?

(A)
```
BigInt temp(rhs);
*this = temp;
return *this;
```

(B)
```
BigInt temp;
temp = rhs;
return temp;
```

(C)
```
*this = rhs;
return *this;
```

(D)
```
BigInt temp(rhs);
*this = temp;
```

(E)
```
BigInt temp(rhs);
return temp;
```

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

20. The author of the case study implemented `operator+` as a free function which in turn calls the `BigInt` member function `operator+=` to compute the addition. Suppose the author had implemented `operator+` as a member function of the `BigInt` class.

```
BigInt BigInt::operator+ (const BigInt & rhs) const;
// postcondition: returns the sum of self + rhs
```

Assume a client program had declared two `BigInt` values, `A` and `B`, in a program. Which of the following code segments would NOT compile if the author had coded `operator+` as a member function?

(A) `A = A + A;`
(B) `A = A + B;`
(C) `A = B + A;`
(D) `A = B + 3;`
(E) `A = 3 + B;`

GO ON TO THE NEXT PAGE

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

Questions 21-22 refer to the following code.

Assume that linked lists are implemented using the following declaration.

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

Consider the following function.

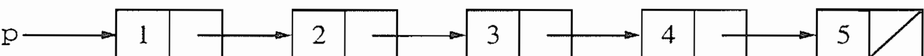
```
void WhatsThis(ListNode * & p)
{
  ListNode * q;
  ListNode * r;

  int sum = 0;
  q = p;
  while (q != NULL)
  {
    sum += q->info;
    q = q->next;
  }
  r = new ListNode;
  r->info = sum;
  r->next = p;
  p = r;
}
```

GO ON TO THE NEXT PAGE

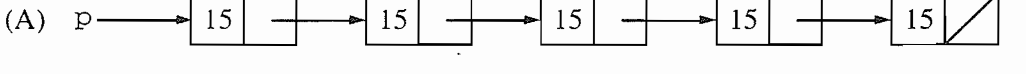
Unauthorized copying or reusing any part of this page is illegal.

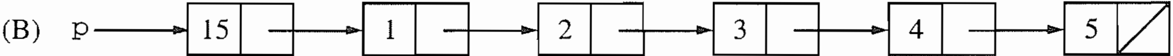
USE THIS SPACE FOR SCRATCHWORK.

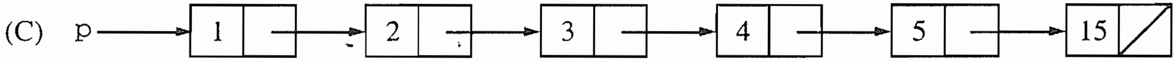
21. Assume that `p` is of type `ListNode *` and points to the first node of the following list.



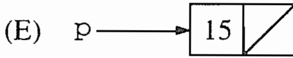
Which of the following best represents `p` after the call `WhatsThis(p)` ?

(A) 

(B) 

(C) 

(D)

(E) 

22. Assume that `t` is of type `ListNode *` and has the value `NULL`. What would be the result of the call `WhatsThis(t)` ?

(A) The function would execute without error and `t` would be left pointing to a list with a single node. The node's `info` data member would be zero, and its `next` data member would be `NULL`.
(B) The function would execute without error and `t` would be left pointing to a list with a single node. The node's `info` data member would be zero, and its `next` data member would be uninitialized.
(C) The function would execute without error and `t` would be left pointing to a list with a single node. The node's `info` data member and `next` data member would both be uninitialized.
(D) The function would execute without error and `t` would remain `NULL`.
(E) A run-time error would occur due to an attempt to dereference a `NULL` pointer.

GO ON TO THE NEXT PAGE

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

Questions 23-24 are based on the following incomplete function definition.

```
int ListMin(const apvector<int> & A, int numItems)
// precondition:  A contains numItems elements
// postcondition: returns the value of the
//                smallest element in A
{
  int min;
  int k = 1;

  min = A[0];
  while (<condition>)
  {
    <body>
  }
  return min;
}
```

The placeholders `<condition>` and `<body>` are to be replaced with code so that the function `ListMin` returns the value of the smallest element in `A`, which contains `numItems` integers. In addition, the following is to be maintained as an invariant of the while loop, true each time the loop begins execution.

```
for all j, 0 ≤ j < k, min ≤ A[j]
```

GO ON TO THE NEXT PAGE

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

23. Of the following choices for *<body>*, which maintains the given loop invariant?

(A)
```
k++;
if (A[k] < min)
{
  min = A[k];
}
```

(B)
```
if (A[k] < min)
{
  min = A[k];
}
k++;
```

(C)
```
k++;
if (A[k] >  min)
{
  min = A[k];
}
```

(D)
```
if (A[k] >  min)
{
  min = A[k];
}
k++;
```

(E)
```
k++;
if (A[k] >=  min)
{
  min = A[k];
}
```

24. Assume that *<body>* has been replaced with code that maintains the given loop invariant. Which of the following choices for *<condition>*, ensures that when the loop terminates, the function `ListMin` returns the value of the smallest element in `A` ?

(A) `k  < min`
(B) `k  < numItems - 1`
(C) `k  < numItems`
(D) `k <= numItems`
(E) `k >= numItems`

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

25. Consider choosing between storing data in a binary search tree and storing data in a hash table. Choosing the binary search tree rather than the hash table is best supported by which of the following reasons?

I. The Big-Oh average time required to insert an item into a binary search tree is less than the average time required to insert an item into a hash table.

II. The Big-Oh average time required to determine whether a given value is in a binary search tree is less than the average time required to determine whether a given value is in a hash table.

III. The Big-Oh average time required to print all items stored in a binary search tree in sorted order is less than the time required to print all items stored in a hash table in sorted order.

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

GO ON TO THE NEXT PAGE

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

26. 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 * T)
{
  if (T == NULL)
  {
    return 0;
  }
  else
  {
    return (Mystery(T->left) + Mystery(T->right));
  }
}
```

Of the following, which best describes what function `Mystery` does?

(A) `Mystery` returns the number of nodes in the tree `T`.
(B) `Mystery` returns the number of leaf nodes in the tree `T`.
(C) `Mystery` returns the number of nonleaf nodes in the tree `T`.
(D) `Mystery` returns the sum of all the integer values stored in the tree `T`.
(E) `Mystery` returns `0`.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

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

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

Consider the following function.

```
void Mystery(ListNode * p)
{
  if (p != NULL)
  {
    Mystery(p->next);
    if ((p->info % 2) != 0)
    {
      cout << p->info << endl;
    }
  }
}
```

Assume that `q` is of type `ListNode *` and that list `q` is a nonempty list whose `info` data members are in increasing order. Of the following, which best describes the output produced by the call `Mystery(q)` ?

(A) All `info` data members with nonzero values will be printed.
(B) All `info` data members with odd values will be printed in increasing order.
(C) All `info` data members with odd values will be printed in decreasing order.
(D) Only the first node's `info` data member will be printed if its value is odd.
(E) Only the last node's `info` data member will be printed if its value is odd.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

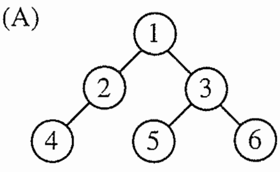
28. Assume that binary trees are implemented using the following declaration.

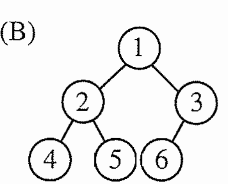
```
struct TreeNode
{
  int data;
  TreeNode * leftChild;
  TreeNode * rightChild;
};
```

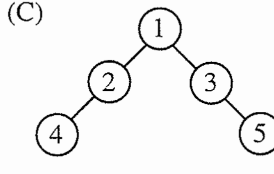
The following function may modify the tree `T`.

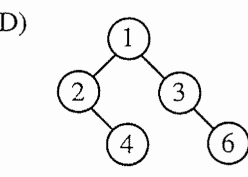
```
void ChangeTree(TreeNode * T)
{
  if (T == NULL)
  {
    return;
  }
  if ((T->rightChild == NULL) && (T->leftChild == NULL))
  {
    return;
  }
  if ((T->rightChild != NULL) && (T->leftChild != NULL))
  {
    ChangeTree(T->rightChild);
    ChangeTree(T->leftChild);
  }
  else if (T->rightChild != NULL)
  {
    T->rightChild = NULL;
  }
  else
  {
    T->leftChild = NULL;
  }
}
```

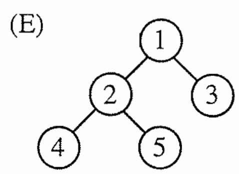
Which of the following trees would NOT be modified by `ChangeTree` ?

(A)


(B)


(C)


(D)


(E)


GO ON TO THE NEXT PAGE

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

29. Consider the design of a hash function to be applied to names (sequences of 2 or more letters). The names will be stored in a hash table (an array) of size 101, and chaining will be used for collision resolution. Which of the following is true?

    (A) The hash function should return a linked list containing the letters of the given name.
    (B) The hash function should return the same value for names with the same number of letters.
    (C) If $name_1$ comes before $name_2$ in alphabetical order, then the value returned by the hash function for $name_1$ should be smaller than the value returned by the hash function for $name_2$.
    (D) The hash function should return integer values in the range 0 to 100.
    (E) The hash function should print an error message if it is called twice with the same name.

30. Consider the following function.

```
void DoSomething(int n)
{
  int j;

  for (j = 1; j < n; j *= 2)
  {
    Proc(j);
  }
}
```

Assume that `Proc` runs in $O(1)$ time. Of the following, which best characterizes the running time of the call `DoSomething(n)` ?

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

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

31. Consider the following code segment.

```
apvector<int> A;
Initialize(A); // resizes A and initializes its elements
int k;

k = 0;
while ((k < A.length()) && (A[k] > 0))
{
  k++;
}
cout << k << endl;
```

Which of the following must be true of the value that is output when the code segment is executed?

(A) The value is less than `A.length()` and it is the index of the first nonpositive element of array `A`.
(B) The value is less than `A.length()` or it is the index of the first nonpositive element of array `A`.
(C) The value is greater than `A.length()` or it is the index of the first nonpositive element of array `A`.
(D) The value is greater than or equal to `A.length()` and it is the index of the first nonpositive element of array `A`.
(E) The value is greater than or equal to `A.length()` or it is the index of the first nonpositive element of array `A`.

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

32. The information stored in a binary search tree (BST) is written to a file as follows:

    Traverse the tree in some order;
    when a node is visited, write its information to a file.

    A new BST is constructed by reading the information from the file and inserting that information into the tree.

    Which of the following traversals will write the information to the file so that the new BST created by reading the file is the same as the original BST ?

    I. Inorder traversal
    II. Preorder traversal
    III. Postorder traversal

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

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

33. Consider the following function.

```
int Strange(int n)
{
  if (n <= 1)
  {
    return 1;
  }
  else
  {
    return (Strange(n - 2) + 1);
  }
}
```

Which of the following replacements for the line containing the recursive call returns the same value as the original version of function `Strange` ?

(A) `return 1;`
(B) `return (n + 1);`
(C) `return (n / 2);`
(D) `return (n / 2 - 1);`
(E) `return (n / 2 + 1);`

GO ON TO THE NEXT PAGE

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

Questions 34-35 assume that binary trees are implemented using the following declaration.

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

34. Consider the following function.

```
int TotalOne(TreeNode * T)
{
  if (T == NULL)
  {
    return 0;
  }
  else if ((T->left == NULL) && (T->right == NULL))
  {
    return T->info;
  }
  else
  {
    return (TotalOne(T->left) + TotalOne(T->right));
  }
}
```

Of the following, which best describes the value returned by the call `TotalOne(T1)` ?

(A) The number of leaf nodes in the tree `T1`
(B) The number of nonleaf nodes in the tree `T1`
(C) The sum of all of the integers stored in the nodes in the tree `T1`
(D) The sum of all of the integers stored in the nonleaf nodes in the tree `T1`
(E) The sum of all of the integers stored in the leaf nodes in the tree `T1`

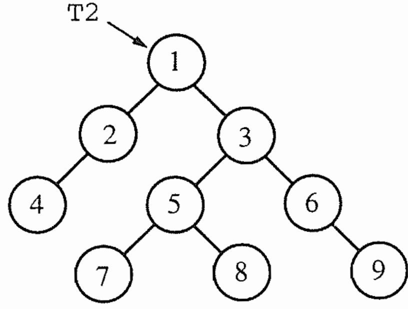
GO ON TO THE NEXT PAGE

Unauthorized copying or reusing any part of this page is illegal.

35. Consider the following function.

```
int TotalTwo(TreeNode * T)
{
  if (T == NULL)
  {
    return 0;
  }
  else if ((T->left == NULL) && (T->right == NULL))
  {
    return 0;
  }
  else
  {
    return (T->info + TotalTwo(T->left) + TotalTwo(T->right));
  }
}
```

Assume that T2 represents the tree shown below.



What value is returned by the call `TotalTwo(T2)` ?

(A) 1
(B) 6
(C) 17
(D) 28
(E) 45

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

Questions 36-37 assume that linked lists are implemented using the following declaration.

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

36. The function `Total`, partially defined below, is intended to return the sum of all the `info` data members in its linked list parameter.

```
int Total(ListNode * L)
{
  int sum = 0;

  while (<condition>)
  {
     sum += L->info;
     <statement>
  }
  return sum;
}
```

Which of the following can be used to replace *<condition>* and *<statement>* so that `Total` will work as intended?

	<condition>	*<statement>*
(A)	`L != NULL`	`L = L->next;`
(B)	`L != NULL`	`L->next = L->next->next;`
(C)	`L->next != NULL`	`L = L->next;`
(D)	`L->next == NULL`	`L = L->next;`
(E)	`L->next == NULL`	`L->next = L->next->next;`

Unauthorized copying or reusing any part of this page is illegal.

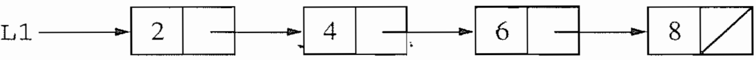
GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

37. Consider the following function.

```
int Mystery(ListNode * L)
{
  int num = 0;
  while (L->next != NULL)
  {
    num += L->info;
    L->next = L->next->next;
  }
  return num;
}
```

Assume that L1 represents the list shown below.



What value is returned by the call `Mystery(L1)` ?

(A) 2
(B) 6
(C) 12
(D) 18
(E) 20

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

USE THIS SPACE FOR SCRATCHWORK.

38. Consider designing a data structure to store names and grade point averages (GPA's) for *N* students. The data structure must be designed so that both of the following are possible.

    1. Given a student's name, determine and print that student's GPA in $O(\log N)$ time.
    2. In $O(N)$ time, print a list of students' names with their GPA's, in order from highest GPA to lowest GPA.

    Assume that the following declaration has been made.

```
struct StudentInfo
{
  apstring name;   // student's name
  double gpa;      // student's GPA
};
```

    Of the following, which is the best design for the data structure?

    (A) An array of `StudentInfo` structures, sorted by GPA from highest to lowest
    (B) An array of `StudentInfo` structures, sorted alphabetically by student name
    (C) Two arrays: one containing only students' names, sorted alphabetically, and the other containing only students' GPA's, sorted from highest to lowest
    (D) Two arrays of `StudentInfo` structures: one sorted alphabetically by student name and the other sorted by GPA from highest to lowest
    (E) Two arrays: one containing only student names, sorted alphabetically, and the other containing `StudentInfo` structures, sorted by GPA from highest to lowest

GO ON TO THE NEXT PAGE

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

39. Consider the following expression.

```
(A > B) && (C <= B)
```

Assume that `A`, `B`, and `C` are integer variables. Which of the expressions given below is (are) equivalent to the one given above?

```
I.   !(A < B) && !(C >= B)
II.  (A > B) && (B > C)
III. !((A <= B) || (B < C))
```

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

GO ON TO THE NEXT PAGE

Unauthorized copying or reusing any part of this page is illegal.

USE THIS SPACE FOR SCRATCHWORK.

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

```
struct ListNode
{
  char info;
  ListNode * next;
};
```

Consider function `Display`, which is intended to print the data in each node of the linked list represented by `p`.

```
void Display(ListNode * p)
{
  while (p->next != NULL)
  {
    cout << p->info << endl;
    p = p->next;
  }
}
```

Which of the following statements about function `Display` is (are) true?

I. If the list is not empty, all characters other than the first will be printed.

II. If the list is not empty, all characters other than the last will be printed.

III. If the list is empty, a `NULL` pointer will be dereferenced.

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

END OF SECTION I

IF YOU FINISH BEFORE TIME IS CALLED, YOU MAY CHECK YOUR WORK ON THIS SECTION.

DO NOT GO ON TO SECTION II UNTIL YOU ARE TOLD TO DO SO.

Unauthorized copying or reusing any part of this page is illegal.

SURVEY QUESTIONS

41. Approximately how many class periods did you spend using the Large Integer case study?

    (A) 0
    (B) 1-2
    (C) 3-5
    (D) 6-10
    (E) More than 10

42. Approximately how many hours did you spend working on the computer on problems related to the Large Integer case study?

    (A) 0
    (B) 1-2
    (C) 3-5
    (D) 6-10
    (E) More than 10

43. At what time during the school year did you use the Large Integer case study?

    (A) Did not use it.
    (B) Used it right before the AP examination.
    (C) Use it only in the middle of the year.
    (D) Used it only at the beginning of the year.
    (E) Used it at multiple times during the year.

44. How many students are in your AP Computer Science class?

    (A) Independent study
    (B) 2-5
    (C) 6-10
    (D) 11-20
    (E) More than 20

Unauthorized copying or reusing any part of this page is illegal.

GO ON TO THE NEXT PAGE

Quick Reference for apstring

```
extern const int npos;  // used to indicate not a position in the string

// public member functions

  // constructors/destructor
  apstring();                           // construct empty string ""
  apstring(const char * s);             // construct from string literal
  apstring(const apstring & str);       // copy constructor
  ~apstring();                          // destructor

  // assignment
  const apstring & operator= (const apstring & str); // assign str
  const apstring & operator= (const char * s);       // assign s
  const apstring & operator= (char ch);              // assign ch

  // accessors
  int length() const;                          // number of chars
  int find(const apstring & str) const;        // index of first occurrence of str
  int find(char ch) const;                     // index of first occurrence of ch
  apstring substr(int pos, int len) const;     // substring of len chars, starting at pos
  const char * c_str() const;                  // explicit conversion to char *

  // indexing
  char operator[ ](int k) const; // range-checked indexing
  char & operator[ ](int k);     // range-checked indexing

  // modifiers
  const apstring & operator+= (const apstring & str); // append str
  const apstring & operator+= (char ch);              // append char


// The following free (non-member) functions operate on strings

// I/O functions
ostream & operator<< ( ostream & os, const apstring & str );
istream & operator>> ( istream & is, apstring & str );
istream & getline( istream & is, apstring & str );

// comparison operators
bool operator== ( const apstring & lhs, const apstring & rhs );
bool operator!= ( const apstring & lhs, const apstring & rhs );
bool operator<  ( const apstring & lhs, const apstring & rhs );
bool operator<= ( const apstring & lhs, const apstring & rhs );
bool operator>  ( const apstring & lhs, const apstring & rhs );
bool operator>= ( const apstring & lhs, const apstring & rhs );

// concatenation operator +
apstring operator+ ( const apstring & lhs, const apstring & rhs );
apstring operator+ ( char ch, const apstring & str );
apstring operator+ ( const apstring & str, char ch );
```

Quick Reference for apvector and apmatrix

```
template <class itemType>
class apvector

  // public member functions

    // constructors/destructor
    apvector();                                      // default constructor (size==0)
    apvector(int size);                              // initial size of vector is size
    apvector(int size, const itemType & fillValue);  // all entries == fillValue
    apvector(const apvector & vec);                  // copy constructor
    ~apvector();                                     // destructor

    // assignment
    const apvector & operator= (const apvector & vec);

    // accessors
    int length() const;                             // capacity of vector

    // indexing
    itemType & operator[ ](int index);              // indexing with range checking
    const itemType & operator[ ](int index) const;  // indexing with range checking

    // modifiers
    void resize(int newSize); // change size dynamically; can result in losing values
```

```
template <class itemType>
class apmatrix

  // public member functions

    // constructors/destructor
    apmatrix();                                               // default size is 0 x 0
    apmatrix(int rows, int cols);                             // size is rows x cols
    apmatrix(int rows, int cols, const itemType & fillValue); // all entries == fillValue
    apmatrix(const apmatrix & mat);                           // copy constructor
    ~apmatrix( );                                             // destructor

    // assignment
    const apmatrix & operator = (const apmatrix & rhs);

    // accessors
    int numrows() const;                                  // number of rows
    int numcols() const;                                  // number of columns

    // indexing
    const apvector<itemType> & operator[ ](int k) const;  // range-checked indexing
    apvector<itemType> & operator[ ](int k);              // range-checked indexing

    // modifiers
    void resize(int newRows, int newCols);   // resizes matrix to newRows x newCols
                                             // (can result in losing values)
```

Quick Reference for apstack and apqueue

```
template <class itemType>
class apstack

  // public member functions

    // constructors/destructor
    apstack();                              // construct empty stack
    apstack(const apstack & s);             // copy constructor
    ~apstack();                             // destructor

    // assignment
    const apstack & operator = (const apstack & rhs);

    // accessors
    const itemType & top() const;           // return top element (NO pop)
    bool isEmpty() const;                   // return true if empty, else false
    int length() const;                     // return number of elements in stack

    // modifiers
    void push(const itemType & item);       // push item onto top of stack
    void pop();                             // pop top element
    void pop(itemType & item);              // combines pop and top
    void makeEmpty();                       // make stack empty (no elements)
```

```
template <class itemType>
class apqueue

  // public member functions

    // constructors/destructor
    apqueue();                                 // construct empty queue
    apqueue(const apqueue & q);                // copy constructor
    ~apqueue();                                // destructor

    // assignment
    const apqueue & operator= (const apqueue & rhs);

    // accessors
    const itemType & front() const;            // return front (no dequeue)
    bool isEmpty() const;                      // return true if empty else false
    int length() const;                        // return number of elements in queue

    // modifiers
    void enqueue(const itemType & item);       // insert item (at rear)
    void dequeue();                            // remove first element
    void dequeue(itemType & item);             // combine front and dequeue
    void makeEmpty();                          // make queue empty
```

Header File for the BigInt class

```
#ifndef _BIGINT_H
#define _BIGINT_H

//
// implements an arbitrary precision integer class
//
// constructors:
//
// BigInt()             -- default constructor, value of integers is 0
// BigInt(int n)        -- initialize to value of n (C++ int)
// BigInt(const apstring & s) -- initialize to value specified by s
//        it is an error if s is an invalid integer, e.g.,
//        "1234abc567".  In this case the bigint value is garbage
//
//
// *****  arithmetic operators:
//
// all arithmetic operators +, -, * are overloaded both
// in form +=, -=, *= and as binary operators
//
// multiplication also overloaded for *= int
// e.g., BigInt a *= 3 (mostly to facilitate implementation)
//
//  ***** logical operators:
//
// bool operator == (const BigInt & lhs, const BigInt & rhs)
// bool operator != (const BigInt & lhs, const BitInt & rhs)
// bool operator <  (const BigInt & lhs, const BigInt & rhs)
// bool operator <= (const BigInt & lhs, const BigInt & rhs)
// bool operator >  (const BigInt & lhs, const BigInt & rhs)
// bool operator >= (const BigInt & lhs, const BigInt & rhs)
//
//  ***** I/O operators:
//
//  void Print()
//        prints value of BigInt (member function)
//  ostream & operator << (ostream & os, const BigInt & b)
//        stream operator to print value
//
// istream & operator >> (istream & in, const BigInt & b)
//        reads whitespace delimited BigInt from input stream in
//
```

```
#include <iostream.h>
#include "apstring.h"              // for strings
#include "apvector.h"              // for sequence of digits

class BigInt
{
  public:

    BigInt();                      // default constructor, value = 0
    BigInt(int);                   // assign an integer value
    BigInt(const apstring &);      // assign a string

    // may need these in alternative implementation

    // BigInt(const BigInt &);     // copy constructor
    // ~BigInt();                  // destructor
    // const BigInt & operator = (const BigInt &); // assignment operator

    // operators: arithmetic, relational

    const BigInt & operator += (const BigInt &);
    const BigInt & operator -= (const BigInt &);
    const BigInt & operator *= (const BigInt &);
    const BigInt & operator *= (int num);

    apstring ToString() const;     // convert to string
    int      ToInt()    const;     // convert to int
    double   ToDouble() const;     // convert to double

    // facilitate operators ==, <, << without friends

    bool Equal(const BigInt & rhs)       const;
    bool LessThan(const BigInt & rhs)    const;
    void Print(ostream & os)             const;

  private:

    // other helper functions

    bool IsNegative()  const;      // return true iff number is negative
    bool IsPositive()  const;      // return true iff number is positive
    int NumDigits()    const;      // return # digits in number

    int GetDigit(int k) const;
    void AddSigDigit(int value);
    void ChangeDigit(int k, int value);

    void Normalize();

    // private state/instance variables

    enum Sign{positive,negative};
    Sign mySign;                       // is number positive or negative
    apvector<char> myDigits;       // stores all digits of number
    int myNumDigits;                   // stores # of digits of number
};
```

```
// free functions

ostream & operator <<(ostream &, const BigInt &);
istream & operator >>(istream &, BigInt &);

BigInt operator +(const BigInt & lhs, const BigInt & rhs);
BigInt operator -(const BigInt & lhs, const BigInt & rhs);
BigInt operator *(const BigInt & lhs, const BigInt & rhs);
BigInt operator *(const BigInt & lhs, int num);
BigInt operator *(int num, const BigInt & rhs);

bool operator == (const BigInt & lhs, const BigInt & rhs);
bool operator <  (const BigInt & lhs, const BigInt & rhs);
bool operator != (const BigInt & lhs, const BigInt & rhs);
bool operator >  (const BigInt & lhs, const BigInt & rhs);
bool operator >= (const BigInt & lhs, const BigInt & rhs);
bool operator <= (const BigInt & lhs, const BigInt & rhs);


#endif   // _BIGINT_H not defined
```

Index of functions in the BigInt class

BigInt::BigInt() 59
BigInt::BigInt(int num) 59
BigInt::BigInt(const apstring & s) 61
const BigInt & BigInt::operator -=(const BigInt & rhs) 63
const BigInt & BigInt::operator +=(const BigInt & rhs) 65
BigInt operator +(const BigInt & lhs, const BigInt & rhs) 65
BigInt operator -(const BigInt & lhs, const BigInt & lhs) 65
void Bigint::Print(ostream & os) const 67
apstring BigInt::ToString() const 67
int BigInt::ToInt() const 67
double BigInt::ToDouble() const 67
ostream & operator <<(ostream & out, const BigInt & big) 69
istream & operator >>(istream & in, BigInt & big) 69
bool operator ==(const BigInt & lhs, const BigInt & rhs) 69
bool BigInt::Equal(const BigInt & rhs) const 69
bool operator !=(const BigInt &lhs, const BigInt & rhs) 69
bool operator <(const BigInt & lhs, const BigInt & rhs) 69
bool BigInt:: LessThan(const BigInt &rhs) const 71
bool operator >(const BigInt & lhs, const BigInt & rhs) 71
bool operator <=(const BigInt & lhs, const BigInt & rhs) 71
bool operator >=(const BigInt & lhs, const BigInt & rhs) 71
void BigInt::Normalize() 71
const BigInt & BigInt:: operator *=(int num) 73
BigInt operator *(const BigInt & a, int num) 73
BigInt operator *(int num, const BigInt & a) 73
const BigInt & BigInt::operator *=(const BigInt & rhs) 75
BigInt operator *(const BigInt & lhs, const BigInt & rhs) 75
int BigInt::NumDigits() const 75
int BigInt::GetDigits() const 75
void BigInt::ChangeDigit(int k, int value) 75
void BigInt::AddSigDigit(int value) 77
bool BigInt::IsNegative() const 77
bool BigInt::IsPositive() const 77

Implementation of BigInt

```
#include <iostream.h>
#include <stdlib.h>
#include <ctype.h>
#include <limits.h>
#include "bigint.h"
#include "apvector.h"

const int BASE = 10;

//
// BigInts are implemented using a Vector<char> to store
// the digits of a BigInt
// Currently a number like 5,879 is stored as the vector {9,7,8,5}
// i.e., the least significant digit is the first digit in the vector;
// for example, GetDigit(0) returns 9 and getDigit(3) returns 5.
// All operations on digits should be done using private
// helper functions:
//
// int  GetDigit(k)        -- return k-th digit
// void ChangeDigit(k,val) -- set k-th digit to val
// void AddSigDigit(val)   -- add new most significant digit val
//
// by performing all ops in terms of these private functions we
// make implementation changes simpler
//
// I/O operations are facilitated by the ToString() member function
// which converts a BigInt to its string (ASCII) representation


BigInt::BigInt()
// postcondition: bigint initialized to 0
   : mySign(positive),
     myDigits(1,'0'),
     myNumDigits(1)
{
   // all fields initialized in initializer list
}

BigInt::BigInt(int num)
// postcondition: bigint initialized to num
   : mySign(positive),
     myDigits(1,'0'),
     myNumDigits(0)
{
    // check if num is negative, change state and num accordingly

    if (num < 0)
    {
        mySign = negative;
        num = -1 * num;
    }


    // handle least-significant digit of num (handles num == 0)

    AddSigDigit(num % BASE);
    num = num / BASE;

    // handle remaining digits of num

    while (num != 0)
    {
        AddSigDigit(num % BASE);
        num = num / BASE;
    }
}
```

```
BigInt::BigInt(const apstring & s)
// precondition: s consists of digits only, optionally preceded by + or -
// postcondition: Bigint initialized to integer represented by s
//                if s is not a well-formed BigInt (e.g., contains non-digit
//                characters) then an error message is printed and abort called
  : mySign(positive),
    myDigits(s.length(),'0'),
    myNumDigits(0)
{
    int k;
    int limit = 0;

    if (s.length() == 0)
    {
        myDigits.resize(1);
        AddSigDigit(0);
        return;
    }
    if (s[0] == '-')
    {
        mySign = negative;
        limit = 1;
    }
    if (s[0] == '+')
    {
        limit = 1;
    }
    // start at least significant digit

    for(k=s.length() - 1; k >= limit; k--)
    {
        if (! isdigit(s[k]))
        {
            cerr << "badly formed BigInt value = " << s << endl;
            abort();
        }
        AddSigDigit(s[k]-'0');
    }
    Normalize();
}
```

```
const BigInt & BigInt::operator -=(const BigInt & rhs)
// postcondition: returns value of bigint - rhs after subtraction
{
    int diff;
    int borrow = 0;

    int k;
    int len = NumDigits();

    if (this == &rhs)          // subtracting self?
    {
        *this = 0;
        return *this;
    }

    // signs opposite? then lhs - (-rhs) = lhs + rhs

    if (IsNegative() != rhs.IsNegative())
    {
        *this +=(-1 * rhs);
        return *this;
    }
    // signs are the same, check which number is larger
    // and switch to get larger number "on top" if necessary
    // since sign can change when subtracting
    // examples: 7 - 3 no sign change,      3 - 7 sign changes
    //          -7 - (-3) no sign change, -3 -(-7) sign changes
    if (IsPositive() && (*this) < rhs ||
        IsNegative() && (*this) > rhs)
    {
        *this = rhs - *this;
        if (IsPositive()) mySign = negative;
        else              mySign = positive;    // toggle sign
        return *this;
    }
    // same sign and larger number on top

    for(k=0; k < len; k++)
    {
        diff = GetDigit(k) - rhs.GetDigit(k) - borrow;
        borrow = 0;
        if (diff < 0)
        {
            diff += 10;
            borrow = 1;
        }
        ChangeDigit(k,diff);
    }
    Normalize();
    return *this;
}
```

```
const BigInt & BigInt::operator +=(const BigInt & rhs)
// postcondition: returns value of bigint + rhs after addition
{

    int sum;
    int carry = 0;

    int k;
    int len = NumDigits();          // length of larger addend

    if (this == &rhs)               // to add self, multiply by 2
    {
      *this *= 2;
      return *this;
    }

    if (rhs == 0)                    // zero is special case
    {
      return *this;
    }

    if (IsPositive() != rhs.IsPositive())    // signs not the same, subtract
    {
        *this -= (-1 * rhs);
        return *this;
    }

    // process both numbers until one is exhausted

    if (len < rhs.NumDigits())
    {
        len = rhs.NumDigits();
    }
    for(k=0; k < len; k++)
    {
        sum = GetDigit(k) + rhs.GetDigit(k) + carry;
        carry = sum / BASE;
        sum = sum % BASE;

        if (k < myNumDigits)
        {
            ChangeDigit(k,sum);
        }
        else
        {
            AddSigDigit(sum);
        }
    }
    if (carry != 0)
    {
        AddSigDigit(carry);
    }
    return *this;
}

BigInt operator +(const BigInt & lhs, const BigInt & rhs)
// postcondition: returns a bigint whose value is lhs + rhs
{
    BigInt result(lhs);
    result += rhs;
    return result;
}

BigInt operator -(const BigInt & lhs, const BigInt & rhs)
// postcondition: returns a bigint whose value is lhs - rhs
{
    BigInt result(lhs);
    result -= rhs;
    return result;
}
```

```
void BigInt::Print(ostream & os) const
// postcondition: BigInt inserted onto stream os
{
    os << ToString();
}

apstring BigInt::ToString() const
// postcondition: returns string equivalent of BigInt
{
    int k;
    int len = NumDigits();
    apstring s = "";

    if (IsNegative())
    {
        s = '-';
    }
    for(k=len-1; k >= 0; k--)
    {
        s += char('0'+GetDigit(k));
    }
    return s;
}

int BigInt::ToInt() const
// precondition: INT_MIN <= self <= INT_MAX
// postcondition: returns int equivalent of self
{
    int result = 0;
    int k;
    if (INT_MAX < *this) return INT_MAX;
    if (*this < INT_MIN) return INT_MIN;

    for(k=NumDigits()-1; k >= 0; k--)
    {
        result = result * 10 + GetDigit(k);
    }
    if (IsNegative())
    {
        result *= -1;
    }
    return result;
}

double BigInt::ToDouble() const
// precondition: DBL_MIN <= self <= DLB_MAX
// postcondition: returns double equivalent of self
{
    double result = 0.0;
    int k;
    for(k=NumDigits()-1; k >= 0; k--)
    {
        result = result * 10 + GetDigit(k);
    }
    if (IsNegative())
    {
        result *= -1;
    }
    return result;
}
```

```
ostream & operator <<(ostream & out, const BigInt & big)
// postcondition: big inserted onto stream out
{
    big.Print(out);
    return out;
}

istream & operator >> (istream & in, BigInt & big)
// postcondition: big extracted from in, must be whitespace delimited
{
    apstring s;
    in >> s;
    big = BigInt(s);
    return in;
}

bool operator == (const BigInt & lhs, const BigInt & rhs)
// postcondition: returns true if lhs == rhs, else returns false
{
    return lhs.Equal(rhs);
}

bool BigInt::Equal(const BigInt & rhs) const
// postcondition: returns true if self == rhs, else returns false
{

    if (NumDigits() != rhs.NumDigits() || IsNegative() != rhs.IsNegative())
    {
        return false;
    }
    // assert: same sign, same number of digits;

    int k;
    int len = NumDigits();
    for(k=0; k < len; k++)
    {
        if (GetDigit(k) != rhs.GetDigit(k)) return false;
    }

    return true;
}

bool operator != (const BigInt & lhs, const BigInt & rhs)
// postcondition: returns true if lhs != rhs, else returns false
{
    return ! (lhs == rhs);
}

bool operator < (const BigInt & lhs, const BigInt & rhs)
// postcondition: return true if lhs < rhs, else returns false
{
    return lhs.LessThan(rhs);
}
```

```
bool BigInt::LessThan(const BigInt & rhs) const
// postcondition: return true if self < rhs, else returns false
{
    // if signs aren't equal, self < rhs only if self is negative

    if (IsNegative() != rhs.IsNegative())
    {
        return IsNegative();
    }

    // if # digits aren't the same must check # digits and sign

    if (NumDigits() != rhs.NumDigits())
    {
        return (NumDigits() < rhs.NumDigits() && IsPositive()) ||
               (NumDigits() > rhs.NumDigits() && IsNegative());
    }

    // assert: # digits same, signs the same

    int k;
    int len = NumDigits();

    for(k=len-1; k >= 0; k--)
    {
        if (GetDigit(k) < rhs.GetDigit(k)) return IsPositive();
        if (GetDigit(k) > rhs.GetDigit(k)) return IsNegative();
    }
    return false;       // self == rhs
}

bool operator > (const BigInt & lhs, const BigInt & rhs)
// postcondition: return true if lhs > rhs, else returns false
{
    return rhs < lhs;
}

bool operator <= (const BigInt & lhs, const BigInt & rhs)
// postcondition: return true if lhs <= rhs, else returns false
{
    return lhs == rhs || lhs < rhs;
}

bool operator >= (const BigInt & lhs, const BigInt & rhs)
// postcondition: return true if lhs >= rhs, else returns false
{
    return lhs == rhs || lhs > rhs;
}

void BigInt::Normalize()
// postcondition: all leading zeros removed
{
    int k;
    int len = NumDigits();
    for(k=len-1; k >= 0; k--)          // find a non-zero digit
    {
        if (GetDigit(k) != 0) break;
        myNumDigits--;                 // "chop" off zeros
    }
    if (k < 0)    // all zeros
    {
        myNumDigits = 1;
        mySign = positive;
    }
}
```

```
const BigInt & BigInt::operator *=(int num)
// postcondition: returns num * value of BigInt after multiplication
{
    int carry = 0;
    int product;                  // product of num and one digit + carry
    int k;
    int len = NumDigits();

    if (0 == num)                 // treat zero as special case and stop
    {
        *this = 0;
        return *this;
    }

    if (BASE < num|| num < 0)           // handle pre-condition failure
    {
        *this *= BigInt(num);
        return *this;
    }

    if (1 == num)                 // treat one as special case, no work
    {
        return *this;
    }

    for(k=0; k < len; k++)        // once for each digit
    {
        product = num * GetDigit(k) + carry;
        carry = product/BASE;
        ChangeDigit(k,product % BASE);
    }

    while (carry != 0)            // carry all digits
    {
        AddSigDigit(carry % BASE);
        carry /= BASE;
    }
    return *this;
}


BigInt operator *(const BigInt & a, int num)
// postcondition: returns a * num
{
    BigInt result(a);
    result *= num;
    return result;
}

BigInt operator *(int num, const BigInt & a)
// postcondition: returns num * a
{
    BigInt result(a);
    result *= num;
    return result;
}
```

```
const BigInt & BigInt::operator *=(const BigInt & rhs)
// postcondition: returns value of bigint * rhs after multiplication
{
    // uses standard "grade school method" for multiplying

   if (IsNegative() != rhs.IsNegative())
   {
      mySign = negative;
   }
   else
   {
      mySign = positive;
   }

   BigInt self(*this);                          // copy of self
   BigInt sum(0);                               // to accumulate sum
   int k;
   int len = rhs.NumDigits();                   // # digits in multiplier

   for(k=0; k < len; k++)
   {
      sum += self * rhs.GetDigit(k);            // k-th digit * self
      self *= 10;                               // add a zero
   }
   *this = sum;
   return *this;
}

BigInt operator *(const BigInt & lhs, const BigInt & rhs)
// postcondition: returns a bigint whose value is lhs * rhs
{
    BigInt result(lhs);
    result *= rhs;
    return result;
}

int BigInt::NumDigits() const
// postcondition: returns # digits in BigInt
{
    return myNumDigits;
}

int BigInt::GetDigit(int k) const
// precondition: 0 <= k < NumDigits()
// postcondition: returns k-th digit
//                (0 if precondition is false)
//                Note: 0th digit is least significant digit
{
    if (0 <= k && k < NumDigits())
    {
        return myDigits[k] - '0';
    }
    return 0;
}

void BigInt::ChangeDigit(int k, int value)
// precondition: 0 <= k < NumDigits()
// postcondition: k-th digit changed to value
//                Note: 0th digit is least significant digit
{
    if (0 <= k && k < NumDigits())
    {
        myDigits[k] = char('0' + value);
    }
    else
    {
        cerr << "error changeDigit " << k << " " << myNumDigits << endl;
    }
}
```

```
void BigInt::AddSigDigit(int value)
// postcondition: value added to BigInt as most significant digit
//                Note: 0th digit is least significant digit
{
    if (myNumDigits >= myDigits.length())
    {
        myDigits.resize(myDigits.length() * 2);
    }
    myDigits[myNumDigits] = char('0' + value);
    myNumDigits++;
}

bool BigInt::IsNegative() const
// postcondition: returns true iff BigInt is negative
{
    return mySign == negative;
}

bool BigInt::IsPositive() const
// postcondition: returns true iff BigInt is positive
{
   return mySign == positive;
}
```

COMPUTER SCIENCE AB

SECTION II

Time—1 hour and 45 minutes

Number of questions—4

Percent of total grade—50

Some questions in the free-response section require you to write program segments. These are to be written in C++.

The questions are printed in this booklet and on the green insert. You are to use the green insert only to organize your responses and for scratchwork, but you must write all your answers in the pink booklet. Write your answers in pencil only. Be sure to write CLEARLY and LEGIBLY. If you make an error, you may save time by crossing it out rather than trying to erase it. All questions are given equal weight. Credit for partial solutions will be given. Do not spend too much time on any one problem.

When you are told to begin, open your booklet, carefully tear out the green insert, and start to work.

DO NOT OPEN THIS BOOKLET UNTIL YOU ARE TOLD TO DO SO.

Copyright © 1999 College Entrance Examination Board and Educational Testing Service. All rights reserved.

COMPUTER SCIENCE AB
SECTION II
Time—1 hour and 45 minutes
Number of questions—4
Percent of total grade—50

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

Note: Assume that the standard libraries (`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. A Quick Reference to the AP C++ classes is included in the case study insert.

GO ON TO THE NEXT PAGE

1. A patchwork quilt can be made by sewing together many blocks, all of the same size. Each individual block is made up of a number of small squares cut from fabric. A block can be represented as a two-dimensional array of nonblank characters, each of which stands for one small square of fabric. The entire quilt can also be represented as a two-dimensional array of completed blocks. The example below shows an array that represents a quilt made of 9 blocks (in 3 rows and 3 columns). Each block contains 20 small squares (of 4 rows by 5 columns). The quilt uses 2 different fabric squares, represented by the characters `'x'` and `'.'`. We consider only quilts where the main block alternates with the same block flipped upside down (i.e., reflected about a horizontal line through the block's center), as in the example below.

```
x...x|..x..|x...x
.x.x.|..x..|.x.x.
..x..|.x.x.|..x..
..x..|x...x|..x..
-----+-----+-----
..x..|x...x|..x..
..x..|.x.x.|..x..
.x.x.|..x..|.x.x.
x...x|..x..|x...x
-----+-----+-----
x...x|..x..|x...x
.x.x.|..x..|.x.x.
..x..|.x.x.|..x..
..x..|x...x|..x..
```

Consider the problem of storing and displaying information about a quilt.

The class `Quilt`, whose declaration is shown below, is used to keep track of the blocks for an entire quilt. Since the pattern is based on one block, we only store that block and the number of rows and columns of blocks. For the example shown above, we would store the upper left 4×5 block, 3 for the number of rows of blocks in the quilt and 3 for the number of columns of blocks in the quilt.

```
class Quilt
{
  public:
    Quilt(istream & inFile, int rowsOfBlocks, int colsOfBlocks);
    // constructor, given number of blocks in each row and column

    apmatrix<char> QuiltToMat();
    // returns a matrix with the entire quilt stored in it

  private:
    apmatrix<char> myBlock; // stores pattern for one block
    int myRowsOfBlocks;     // number of rows of blocks in the quilt
    int myColsOfBlocks;     // number of columns of blocks in the quilt

    void PlaceBlock(int startRow, int startCol,
                    apmatrix<char> & qmat);
    void PlaceFlipped(int startRow, int startCol,
                      apmatrix<char> & qmat);
};
```

GO ON TO THE NEXT PAGE

(a) Write the code for the constructor that initializes a quilt, as started below. The constructor reads the block pattern for the main block from a file represented by the parameter `inFile.` You may assume the file is open and that the file contains the number of rows followed by the number of columns for the block, followed by the characters representing the pattern. For example, the file `pattern,` which contains the pattern for the first block in the quilt shown above, would look like this:

```
4 5
x...x
.x.x.
..x..
..x..
```

The constructor also sets the number of rows and columns of blocks which make up the entire quilt in the initializer list.

Complete the constructor below. Assume that the constructor is called only with parameters that satisfy its precondition.

```
Quilt::Quilt(istream & inFile, int rowsOfBlocks, int colsOfBlocks)
  : myBlock(0, 0), myRowsOfBlocks(rowsOfBlocks),
    myColsOfBlocks(colsOfBlocks)
// precondition:  inFile is open, rowsOfBlocks > 0, colsOfBlocks > 0
// postcondition: myRowsOfBlocks and myColsOfBlocks are initialized to
//                the number of rows and columns of blocks that make up
//                the quilt; myBlock has been resized and
//                initialized to the block pattern from the
//                stream inFile.
```

GO ON TO THE NEXT PAGE

(b) Write the private member function `PlaceFlipped`, as started below. `PlaceFlipped` is intended to place a flipped (upside-down) version of the block into the matrix `qmat` with the flipped block's upper left corner located at the `startRow, startCol` position in `qmat`.

For example, if quilt `Q` contains the block shown in part (a) and if `M` is a matrix large enough to hold the characters in the whole quilt, then the call

```
Q.PlaceFlipped(4, 10, M)
```

would place the flipped version of `Q`'s quilt block into matrix `M` as the third block in the second row of quilt blocks. This is the block whose upper-left corner is at position `M[4][10]`. In the diagram below, the upper-left corner of the flipped block being placed into `M` is circled.

```
x...x|..x..|x...x
.x.x.|..x..|.x.x.
..x..|.x.x.|..x..
..x..|x...x|..x..
-----+-----+-----
..x..|x...x|..x..
..x..|.x.x.|..x..
.x.x.|..x..|.x.x.
x...x|..x..|x...x
-----+-----+-----
x...x|..x..|x...x
.x.x.|..x..|.x.x.
..x..|.x.x.|..x..
..x..|x...x|..x..
```

You may adapt the code of the private member function `PlaceBlock`, given below, which places the block (not inverted) into the matrix `qmat` with the block's upper left corner located at the `startRow, startCol` position.

```
void Quilt::PlaceBlock(int startRow, int startCol,
                       apmatrix<char> & qmat)
// precondition:  startRow ≥ 0; startCol ≥ 0;
//                startRow + myBlock.numrows() ≤ qmat.numrows();
//                startCol + myBlock.numcols() ≤ qmat.numcols();
// postcondition: myBlock has been copied into the matrix
//                qmat with its upper-left corner at the position
//                startRow, startCol
{
  int r, c;
  for (r = 0; r < myBlock.numrows(); r++)
  {
    for (c = 0; c < myBlock.numcols(); c++)
    {
      qmat[startRow + r][startCol + c] = myBlock[r][c];
    }
  }
}
```

Complete the member function `PlaceFlipped` below. Assume that `PlaceFlipped` is called only with parameters that satisfy its precondition.

```
void Quilt::PlaceFlipped(int startRow, int startCol,
                         apmatrix<char> & qmat)
// precondition:  startRow ≥ 0; startCol ≥ 0;
//                startRow + myBlock.numrows() ≤ qmat.numrows();
//                startCol + myBlock.numcols() ≤ qmat.numcols();
// postcondition: a flipped version of myBlock has been copied into the
//                matrix qmat with its upper-left corner at the position
//                startRow, startCol
{
  int r, c;

  for (r = 0; r < myBlock.numrows(); r++)
  {
    for (c = 0; c < myBlock.numcols(); c++)
    {


    }
  }
}
```

GO ON TO THE NEXT PAGE

(c) Write the member function `QuiltToMat`, as started below. `QuiltToMat` returns a matrix representing the whole quilt in such a way that the main block alternates with the flipped version of the main block, as shown in the original example. If `Q` represents the example quilt, then the call `Q.QuiltToMat()` would return a matrix of characters with the given block placed starting with the upper-left corner at position 0, 0; the flipped block placed with its upper-left corner at position 0, 5; the given block placed with its upper-left corner at position 0, 10; the flipped block placed with its upper-left corner at position 4, 0, and so on.

In writing `QuiltToMat`, you may call functions `PlaceBlock` and `PlaceFlipped` specified in part (b). Assume that `PlaceBlock` and `PlaceFlipped` work as specified, regardless of what you wrote in part (b).

Complete the member function `QuiltToMat` below.

```
apmatrix<char> Quilt::QuiltToMat()
```

GO ON TO THE NEXT PAGE

2. This question involves reasoning about the code from the Large Integer case study. A copy of the code is provided as part of this examination.

(a) Write a new `BigInt` member function `Div2`, as started below. `Div2` should change the value of the `BigInt` to be the original value divided by 2 (integer division). Assume the `BigInt` is greater than or equal to `0`. One algorithm for implementing `Div2` is:

1. Initialize a variable `carryDown` to `0`.

2. For each digit, `d`, starting with the most significant digit,

   2.1 replace that digit with `(d / 2) + carryDown`

   2.2 let `carryDown` be `(d % 2) * 5`

3. Normalize the result.

Complete member function `Div2` below.

```
void BigInt::Div2()
// precondition:  BigInt ≥ 0
```

GO ON TO THE NEXT PAGE

(b) Write a definition to overload the `/` operator, as started below. Assume that `dividend` and `divisor` are both positive values of type `BigInt`.

For example, assume that `bigNum1` and `bigNum2` are positive values of type `BigInt`:

bigNum1	bigNum2	bigNum1 / bigNum2
18	9	2
17	2	8
8714	2178	4
9990	999	10

There are many ways to implement division; however, you must use a binary search algorithm to find the quotient of `dividend` divided by `divisor` in this problem. You will receive no credit on this part if you do not use a binary search algorithm.

One possible algorithm for implementing division using binary search is as follows:

Let `low` and `high` represent a range in which the quotient is found.

Initialize `low` to `0` and `high` to `dividend`.

For each iteration, compute `mid = (low + high + 1)`, divide `mid` by `2`, and compare

`mid * divisor` with `dividend` to maintain the invariant that `low ≤ quotient` and

`high ≥ quotient`.

When `low == high`, the quotient has been found.

In writing function `operator/` you may call function `Div2` specified in part (a). Assume that `Div2` works as specified, regardless of what you wrote in part (a). You will receive NO credit on this part if you do not use a binary search algorithm.

Complete `operator/` below. Assume that `operator/` is called only with parameters that satisfy its precondition.

```
BigInt operator/ (const BigInt & dividend, const BigInt & divisor)
// precondition:  dividend > 0, divisor > 0
```

GO ON TO THE NEXT PAGE

3. Consider designing a data structure to represent a high school club of students with a common interest. The information to be stored in the data structure is as follows:

   1. The name of the club.

   2. A linked list of club members. For each member, the student's name and grade level in high school is stored.

   One way to do this is to use the declarations given below. The first is for a node in the linked list of club members and the second is for the club itself.

```
struct Member
{
  apstring name;        // name of club member
  int level;            // grade 9, 10, 11, or 12
  Member * next;        // next member on the list

  Member(const apstring & nm, int lv, Member * nx);
  // constructor
};

Member::Member(const apstring & nm, int lv, Member * nx)
  : name(nm), level(lv), next(nx)
{}

struct Club
{
  apstring clubName;        // name of club
  Member * memberList;      // list of members in the club

  Club();                           // default constructor
  Club(const apstring & clubnm);    // constructor

};
```

   For example, shown below are two variables of type `Club`; the first represents the German club and the second represents the Computer club.

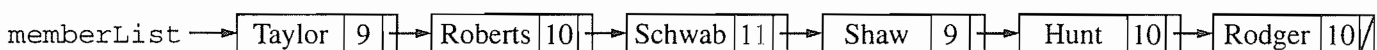
```
      Club1
clubName:  German

memberList --> [Roberts|10|] --> [Schwab|11|] --> [Shaw|9|] --> [Hunt|10|] --> [Rodger|10|/]

      Club2
clubName:  Computer

memberList --> [Schwab|11|] --> [Chase|9|] --> [Taylor|9|] --> [Chen|12|] --> [Larson|11|] --> [Chapman|12|/]
```

GO ON TO THE NEXT PAGE

(a) Write function `InsertMember`, as started below. `InsertMember` adds a student with the given name and respective level in high school to the given club.

For example, after the call `InsertMember("Taylor", 9, Club1)`, variable `Club1` might be as shown below. The diagram shows that the new member `"Taylor"` was inserted at the beginning of the list of members, but the student could have been inserted anywhere in the list.

```
    Club1
clubName:   German
```



Complete function `InsertMember` below. Assume that `InsertMember` is called only with parameters that satisfy its precondition.

```
void InsertMember(const apstring a name, int level, Club & anyClub)
// precondition:  anyClub contains zero or more members, name does not
//                appear in anyClub, and level is 9, 10, 11, or 12
// postcondition: a new member with the given name and respective level
//                has been added to anyClub
```

GO ON TO THE NEXT PAGE

(b) Write function `CountLevel`, as started below. `CountLevel` counts and returns the number of club members of the specified level in `anyClub`.

For example, the call `CountLevel(Club1, 10)` returns 3, since there are 3 tenth graders in the German club. The call `CountLevel(Club2, 10)` returns 0 since there are no tenth graders in the Computer club.

Complete function `CountLevel` below. Assume that `CountLevel` is called only with parameters that satisfy its precondition.

```
int CountLevel(const Club & anyClub, int level)
// precondition:  level is 9, 10, 11, or 12
// postcondition: returns the number of members in anyClub
//                of that level
```

GO ON TO THE NEXT PAGE

(c) Write function `PrintClubsWithMostInLevel`, as started below. `PrintClubsWithMostInLevel` is given an array of clubs and a grade level; it determines which of the clubs in the array has the most members in the given level in high school, and prints the name of that club. If there is a tie, multiple clubs are printed, one club per line.

For example if `clubs` is the array shown below,

`PrintClubsWithMostInLevel(clubs, 10)` would print

```
German

Singing
```

as both of these clubs contain the largest number of tenth graders (3).

```
  clubName: German
0
  memberList → Taylor 9 → Roberts 10 → Schwab 11 → Shaw 9 → Hunt 10 → Rodger 10 /

  clubName: Computer
1
  memberList → Schwab 11 → Chase 9 → Taylor 9 → Chen 12 → Larson 11 → Chapman 12 /

  clubName: Camera
2
  memberList → Clancy 10 → Matsuoka 11 → Sauls 11 → Stehlik 12 → Nevison 12 /

  clubName: Singing
3
  memberList → Hunt 10 → Clancy 10 → Carter 12 → Owens 11 → Lee 10 → Tilton 12 → Brady 12 /

  clubName: Running
4
  memberList → Weiss 10 → Astrachan 12 → Johnson 11 → Lewis 10 /
```

GO ON TO THE NEXT PAGE

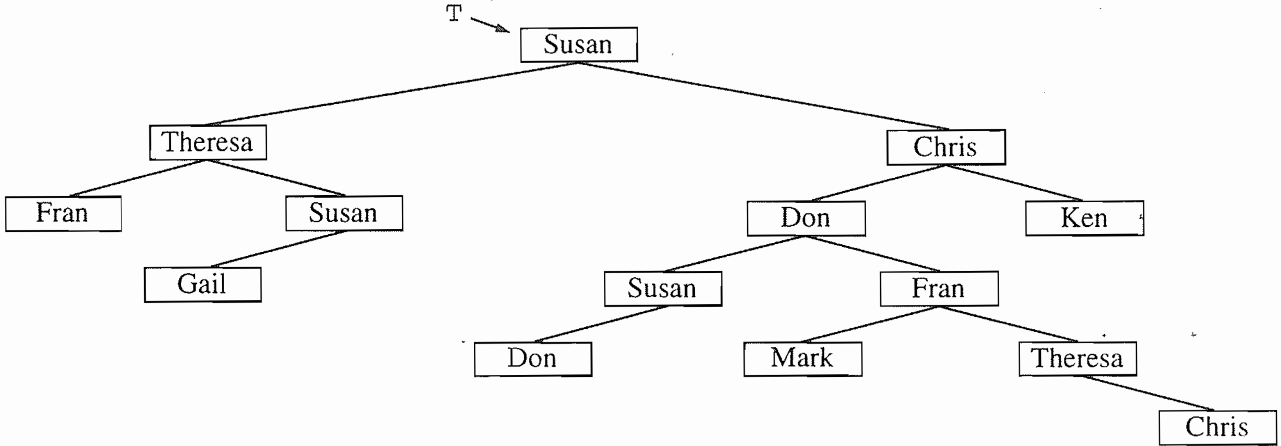
In writing PrintClubsWithMostInLevel, you may call function CountLevel specified in part (b). Assume CountLevel works as specified, regardless of what you wrote in part (b).

Complete function PrintClubsWithMostInLevel below. Assume that PrintClubsWithMostInLevel is called only with parameters that satisfy its precondition.

```
void PrintClubsWithMostInLevel(const apvector<Club> & clubsArray,
                               int level)
// precondition:  clubsArray contains clubsArray.length() clubs
// postcondition: prints the name of the club or clubs in clubsArray
//                that contain the largest number of members in a given
//                level in high school (9 - 12), one club per line.
```

GO ON TO THE NEXT PAGE

4. Consider a binary tree of names. Names may appear more than once in the tree as shown in the example below.



Assume that a binary tree of names is implemented using the following declaration.

```
struct TreeNode
{
  apstring name;
  TreeNode * left;
  TreeNode * right;
};
```

Assume that the integer function `Max` has been defined. `Max` returns the greater of its two integer parameters, as specified below.

```
int Max(int x, int y)
// postcondition: returns the maximum of x and y
```

A path in a tree is a sequence of nodes

$$node_1,\ node_2,\ \ldots,\ node_k$$

such that for any j, $node_{j+1}$ is either the left child or right child of $node_j$. The length of a path is the number of nodes in the path (k in the example just given).

GO ON TO THE NEXT PAGE

(a) Write function `PathLength`, as started below. If person `P` is in tree `T`, then `PathLength(T, P, 1)` should return the length of the longest path from the root of `T` to a node containing `P`; if person `P` does not appear in tree `T`, then `PathLength(T, P, 1)` should return 0. Note that parameter `level` can be used to keep track of the current level of the tree.

For the tree given above, the following are examples of calls to `PathLength`.

Function Call	Value Returned
`PathLength(T, "Susan", 1)`	4
`PathLength(T, "Ken", 1)`	3
`PathLength(T, "Chris", 1)`	6
`PathLength(T, "David", 1)`	0
`PathLength(T->left, "Theresa", 1)`	1
`PathLength(T->right->left, "Don", 1)`	3

In writing `PathLength`, you may call function `Max` as specified in the beginning of this question. Assume that `Max` works as specified.

Complete function `PathLength` below.

```
int PathLength(TreeNode * T, const apstring & someName, int level)
```

GO ON TO THE NEXT PAGE

(b) Write function `RootPath`, as started below. `RootPath` should return the length of the longest path from the root of the tree to a node containing the same name as the root; if no node other than the root contains that name, then `RootPath` returns 1; if the tree is empty, `RootPath` should return 0. For the tree given above, the following are examples of calls to `RootPath`.

Function Call	Value Returned
`RootPath(T)`	4
`RootPath(T->left)`	1
`RootPath(T->right)`	5
`RootPath(T->left->left->left)`	0

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

Complete function `RootPath` below.

```
int RootPath(TreeNode * T)
```

END OF EXAMINATION

Chapter III

Answers to the 1999 AP Computer Science A and Computer Science AB Examinations

- Section I: Multiple Choice
- Section II: Free Response
  - Student Preparation for the Exams
  - Overview of the Questions, Scoring Guidelines, and Sample Student Responses
  - Computer Science A
  - Computer Science AB

Section I: Multiple Choice

Listed below are the correct answers to the multiple-choice questions and the percentage of AP candidates who answered each question correctly by AP grade, and the total percentage answering correctly.

Section 1 Answer Key and Percent Answering Correctly Computer Science A

Item No.	Correct Answer	Percent Correct by Grade 5	4	3	2	1	Total Percent Correct
1	C	96	90	82	76	62	79
2	D	99	99	99	99	93	97
3	C	94	88	82	80	69	81
4	A	99	97	96	94	72	89
5	A	98	96	93	87	56	83
6	B	97	92	80	66	32	70
7	D	93	84	77	69	50	72
8	E	93	82	73	64	46	70
9	E	95	87	83	78	52	76
10	C	92	85	80	74	48	73
11	B	99	98	94	89	51	82
12	B	96	84	74	63	40	68
13	A	98	84	61	41	19	57
14	A	96	90	80	70	35	70
15	B	84	70	53	43	24	52
16	E	70	45	31	25	19	36
17	A	92	64	40	29	20	46
18	D	91	67	49	42	34	55
19	A	72	41	27	21	18	33
20	C	89	78	62	53	35	61
21	E	82	64	54	49	37	55
22	A	87	73	61	52	40	61
23	D	99	98	93	84	42	79
24	D	93	84	70	54	24	62
25	D	88	80	74	68	56	71
26	C	83	53	36	23	15	39
27	E	94	83	70	62	42	68
28	D	80	61	54	51	48	58
29	C	88	73	61	47	26	56
30	E	90	75	55	33	11	50
31	C	86	78	65	57	35	61
32	E	96	87	80	68	43	72
33	A	89	68	50	36	19	50
34	D	93	80	63	44	24	59
35	E	72	56	37	24	12	39
36	D	79	52	32	26	16	39
37	C	95	88	70	63	31	66
38	C	98	90	79	69	47	75
39	B	91	72	49	30	13	49
40	A	84	56	31	20	15	42

Section 1 Answer Key and Percent Answering Correctly Computer Science AB

Item No.	Correct Answer	Percent Correct by Grade 5	4	3	2	1	Total Percent Correct
1	C	87	80	74	62	42	72
2	E	98	98	95	91	66	91
3	D	93	85	78	65	51	77
4	E	98	94	90	80	57	86
5	B	96	94	89	85	56	86
6	E	96	92	83	76	48	81
7	B	99	98	94	86	67	90
8	A	97	92	88	80	61	86
9	B	93	87	80	72	59	80
10	D	88	84	75	62	44	73
11	D	93	89	80	73	51	80
12	E	33	27	23	21	17	25
13	A	95	86	79	70	54	79
14	B	45	28	23	15	14	28
15	B	84	72	64	52	32	64
16	D	94	89	76	64	47	77
17	D	93	80	64	47	35	68
18	C	72	48	31	21	15	42
19	A	75	52	36	28	26	48
20	E	67	44	35	26	19	42
21	B	97	92	88	75	43	82
22	A	94	88	78	66	37	76
23	B	89	84	79	74	60	79
24	C	89	78	68	49	31	67
25	C	73	56	40	28	16	47
26	E	97	94	80	58	25	75
27	C	97	92	82	65	30	77
28	E	93	78	62	48	27	66
29	D	83	62	42	26	19	52
30	B	77	58	44	30	19	51
31	E	60	39	33	23	17	39
32	B	76	60	51	35	22	53
33	E	89	75	67	55	34	68
34	E	93	84	72	60	32	72
35	C	87	75	61	43	21	63
36	A	96	88	80	65	43	79
37	B	86	69	47	30	15	56
38	D	70	49	35	24	21	45
39	C	68	54	48	38	33	52
40	E	83	62	46	36	28	58

Section II: Free Response

Student Preparation for the Exams

The 1999 AP Computer Science A and Computer Science AB Examinations each had four free-response questions, two of which were common questions appearing on both examinations. Correct solutions, scoring guidelines, and actual student answers are provided on the following pages for each of the six free-response questions. The guidelines can be read as examples of what faculty consultants thought were important in doing these problems.

1999 was a remarkable year for the AP Computer Science program. It marked the initial delivery year of the A and AB exams in a new programming language, C++, and saw a record number of exams taken (almost 19,000 total; 12,067 A and 6,591 AB, up from approximately 10,500 in 1998 and 11,750, the previous high, in 1997). The change in language allowed questions that explored the new territory that C++ has opened up, while resulting in an exam that graded very similarly to past AP Computer Science Exams. In particular, while one might expect dramatic differences in student performance due to the language shift, this was not observed. Students made conceptual errors similar to those they made in the past and did not seem stymied at all by the new delivery language.

As had been done in 1998, the exam contained slightly different versions of an overlap question to make the question work reasonably well for both A and AB students. To accomplish this goal, a more detailed algorithm was provided on part (b) of the A version of the Case Study question than was provided on the AB version. Looking at each exam overall, the A exam was slightly more difficult than the 1998 A exam, with question A1 being easier and questions A2 and A4 being harder than similar questions in the past. The AB questions were well balanced, with the question AB3 being a slightly easier than usual linked list problem. See Chapter IV and the Technical Corner of the AP website (www.collegeboard.org/ap/techman) for information on how the difficulty level of questions is handled when determining AP grade boundaries each year.

In general, students need to be familiar with the member functions and operators available to them for the AP classes. Teachers should teach these classes early, as well as introduce other classes, to ensure that students are comfortable with writing constructors and member functions.

Students should be encouraged to use abstractions that are provided. Some excellent examples of this principle are using the overloaded relational operators for apstrings (e.g., `==`, `<`, `>`) instead of performing character-by-character comparisons and the private helper functions `GetDigit`, `AddSigDigit`, `ChangeDigit`, and `NumDigits` in the Large Integer Case Study. Invariably, when students try to code these themselves, they get them wrong.

On array questions, where students are provided with an explicit number of entries to examine, they typically confuse that number with the length of the array (e.g., `numStudents` vs. `roster.length()` on Question A1, part (a)). Students also lose points by being off by one in their loops which traverse the array, e.g., using

```
for (i = 0; i <= numStudents; i++)
```

instead of

```
for (i = 0; i < numStudents; i++)
```

on Question A1. Students should also be discouraged from arbitrarily resizing arrays. If a question indicates that there is enough room to store the data (as in Question A1 and A2), there's no need to resize. Benign resizes were not reasons to lose points on the 1999 exam, but destructive resizes did lose points.

Students are also expected to be able to read from external files. Files on AP Exams are well formatted, so there's almost no need to resort to strange loops or member functions to read the data. A file `(infile)` that contained an arbitrary number of lines of text where each line contained a name (as an apstring), an age (as an int), and a QPA (as a double) would be read as follows:

```
while (infile >> name >> age >> qpa)
{
    process name, age, and qpa
}
```

Note that no reference is made to `eof`, and the reads are happening as extractions in the loop guard.

Overview of the Questions, Scoring Guidelines, and Sample Student Responses

The answers presented here are actual student responses to the six free-response questions on the 1999 AP Computer Science Examinations. The students gave permission to have their work reproduced at the time they took the exam. These responses were read and scored by the question leaders and faculty consultants assigned to each particular question and were used as sample responses for the training of faculty consultants during the AP Reading in June 1999. The actual scores that these students received, as well as a brief explanation of why, are indicated.

You will find a rubric for each question, followed by three sample solutions illustrating the application of the rubric, with commentary. In addition to the rubrics, each faculty consultant was provided with a Usage Sheet, which is printed below. Usage deductions are taken for egregious syntax violations (the maximum deduction for even the worst error is 1 point), and can only be deducted if the part has earned credit. In general, no usage points are deducted for usage mistakes for which evidence of understanding appears elsewhere in the problem. For example, if there are *no* variables declared in a problem, then usage points may be deducted. However, a missing declaration in the presence of other declarations does *not* lose points. Some usage errors may be addressed specifically in rubrics with points deducted in a manner different from that indicated on the "general" Usage Sheet.

Text of the questions, scoring guidelines, and canonical solutions can also be accessed from the AP Computer Science website at www.collegeboard.org/ap/computer-science.

1999 Usage Sheet

In general, no usage points are deducted for usage mistakes for which evidence of understanding appears elsewhere in the problem. For example, if there are *no* variables declared in a problem, then usage points may be deducted. However, a missing declaration in the presence of other declarations does *not* lose points. Also, you should not take off usage points for syntactically correct code that goes beyond the AP subset (e.g., using printf or scanf, or returning 0 instead of false for a bool).

Usage points can only be deducted if the PART has earned credit. Some usage errors may be addressed specifically in rubrics with points deducted in a manner other than indicated on this sheet.

Non-penalized errors	**Minor errors (1/2 point)**	**Major errors (1 point)**
Case discrepancies, unless confuses identifiers	Misspelled/confused identifier (e.g., link/next)	Reads new values for parameters (write prompts part of this point)
Missing ;'s	No variables declared	Function result written to output
Missing { }'s where indentation clearly conveys intent	`MemberFunction(obj)` instead of `obj.MemberFunction( )`	Type error (uses type name instead of variable identifier)
Default constructor called with parens, e.g., `BigInt b( )`	`Param.FreeFunction( )` instead of `FreeFunction(param)`	
`obj.Func` instead of `obj.Func( )`	Void function returns a value	
Loop variables used outside loop	Modifying a `const` parameter	
`[r, c]` instead of `[r][c]`	Unnecessary `cout << "done"`	
= instead of == (and vice-versa)	Unnecessary `cin` (to pause)	
Missing ( )'s around if/while tests		
<< instead of >> (and vice-versa)		
`*foo.data` instead of `(*foo).data`		

Computer Science A: Question 1

Overview

A "standard" one-dimensional array question which measures the student's facility with one of the most basic data structures, an array (apvector) of records (structs). Overall, most students did very well on this question. The mean score was 5.51 with 37 percent of the students scoring an 8 or 9. The question discriminated best at the 2-3 grade cutpoint.

In part (a), students were asked to traverse the data structure and perform a simple computation to set the GPA field for each student. They needed to identify the case where `creditHours` was zero, as this would result in a divide-by-zero error. Many students neglected to handle the zero case by failing to test appropriately.

In part (b), students were asked to examine individual fields of the struct and return an appropriate value. There were very few errors on this part.

In part (c), students were asked to select the seniors from the initial array and place them in a new array. During the leaders' meetings prior to the beginning of the reading of the exam, it was determined that this part of the question was ambiguous. No mention was made as to whether the `seniors` array was to be viewed as a "new" array containing only the new seniors, or if it already had data to which the new seniors were to be appended. It was decided that credit would be given in either case. Thus, a student who failed to initialize `numSeniors` to `0` was assumed to be appending to an extant array, and students who initialized `numSeniors` to `0` were assumed to be creating a new array. This resulted in students not necessarily losing any points for failing to initialize `numSeniors` to `0` (as you would not do so if you were appending to an extant list of seniors). Of course, the student needed to remain consistent in order not to lose points.

Scoring Guidelines

Part A: `ComputeGPA` (3 points)

- **+1** loop over all elements (must use loop variable to access array somewhere in loop body)
  - **+1/2** attempt
  - **+1/2** correct (use of `roster.length()` loses this half)
- **+1** correct computation in zero case
  - **+1/2** test to identify `hours == 0`
  - **+1/2** assignment of 0 and no further reassignment
- **+1** correct computation in other/only case

Usage: -1 improper struct notation

Part B: `IsSenior` (2 points)

- **+1** attempt (looks at only 1 student record and "examines" both `creditHours` and `GPA`)
- **+1** correct (includes return, correct struct notation, uses `student` instead of `StudentInfo`)

Part C: `FillSeniorList` (4 points)

- **+1** loop over all elements (must use loop variable to access array somewhere in loop body)
  - **+1/2** attempt
  - **+1/2** correct
- **+1** correct test to identify senior (function call must be perfect)
- **+1** assign appropriate array element to correct position in seniors
- **+1** correct value returned for `numSeniors` (ignore missing init)

Usage: -1/2 `numSeniors/seniors` redeclared as a local variable

Sample Student Response

Excellent Solution: 9 points

1. Assume that student records are implemented using the following declaration.

```
struct StudentInfo
{
   apstring name;
   int creditHours;
   double gradePoints;
   double GPA;
};
```

(a) Write function ComputeGPA, as started below. ComputeGPA should fill in the GPA data member for the first numStudents records in its apvector parameter roster. A student's GPA (grade point average) is computed by dividing gradePoints by creditHours. The GPA for a student with 0 credit hours should be set to 0.

Complete function ComputeGPA below. Assume that ComputeGPA is called only with parameters that satisfy its precondition.

```
void ComputeGPA(apvector<StudentInfo> & roster, int numStudents)
// precondition:  roster contains numStudents records,
//                0 < numStudents ≤ roster.length(), in which the
//                name, creditHours and gradePoints data members
//                have been initialized.
// postcondition: The GPA data member for the first numStudents records
//                in roster has been calculated.
{
   int count;
   for(count=0; count < numStudents; count++)
   {
      if (roster[count].creditHours == 0)
           roster[count]. GPA = 0;
      else roster[count]. GPA = roster[count]. gradepoints / roster[count]. creditHours;
   }
}
```

(b) Write function `IsSenior`, as started below. `IsSenior` should return `true` if the given student has at least 125 credit hours and has a GPA of at least 2.0; otherwise, `IsSenior` should return `false`.

For example:

student				Result of the call `IsSenior(student)`
name	creditHours	gradePoints	GPA	
King	45	171	3.8	`false` (not enough credit hours)
Norton	128	448	3.5	`true`
Solo	125	350	2.8	`true`
Kramden	150	150	1.0	`false` (GPA too low)

Complete function `IsSenior` below.

```
bool IsSenior(const StudentInfo & student)
// postcondition: returns true if this student's credit hours ≥ 125
//                and GPA ≥ 2.0; otherwise, returns false
{
   return((student.creditHours >=125) && (student.GPA >= 2.0))
}
```

(c) Write function `FillSeniorList`, as started below. `FillSeniorList` determines which students in the array `roster` are seniors and copies those students' records to the array `seniors`. It should also set the value of parameter `numSeniors` to be the number of seniors in the array `seniors`.

In writing `FillSeniorList`, you may call function `IsSenior` specified in part (b). Assume that `IsSenior` works as specified, regardless of what you wrote in part (b).

Complete function `FillSeniorList` below. Assume that `FillSeniorList` is called only with parameters that satisfy its precondition.

```
void FillSeniorList(const apvector<StudentInfo> & roster,
                    int numStudents, apvector<StudentInfo> & seniors,
                    int & numSeniors)
// precondition: roster contains numStudents records,
//               0 < numStudents ≤ roster.length(),
//               and seniors is large enough to hold all of
//               the seniors' records
{
   int count;

  for(count=0; count < numStudents; count++)
  {
      if(IsSenior(roster[count])
      {
         seniors[numSeniors] = roster[count];
         numSeniors++;
      }
}
```

Commentary:

The student earns all 9 points, even though there's a missing semi-colon in part (b) (an unpenalized usage error) and a "missing" initialization of `numSeniors` in part (c). As indicated in the overview on page 168, the missing initialization was allowed as correct in this problem.

Good Solution: 6 points (5½, rounds up)

1. Assume that student records are implemented using the following declaration.

```
struct StudentInfo
{
   apstring name;
   int creditHours;
   double gradePoints;
   double GPA;
};
```

(a) Write function `ComputeGPA`, as started below. `ComputeGPA` should fill in the `GPA` data member for the first `numStudents` records in its `apvector` parameter `roster`. A student's GPA (grade-point average) is computed by dividing `gradePoints` by `creditHours`. The GPA for a student with 0 credit hours should be set to 0.

Complete function `ComputeGPA` below. Assume that `ComputeGPA` is called only with parameters that satisfy its precondition.

```
void ComputeGPA(apvector<StudentInfo> & roster, int numStudents)
// precondition:  roster contains numStudents records,
//                0 < numStudents ≤ roster.length(), in which the
//                name, creditHours and gradePoints data members
//                have been initialized.
// postcondition: The GPA data member for the first numStudents records
//                in roster has been calculated.
{
  for(int a=0 ; a ≤numStudents ;a++)
   {
   roster[a].GPA = roster[a].gradePoints/roster[a].creditHours;
   }
}
```

(b) Write function `IsSenior`, as started below. `IsSenior` should return `true` if the given student has at least 125 credit hours and has a GPA of at least 2.0; otherwise, `IsSenior` should return `false`.

For example:

student				Result of the call `IsSenior(student)`
name	creditHours	gradePoints	GPA	
King	45	171	3.8	`false` (not enough credit hours)
Norton	128	448	3.5	`true`
Solo	125	350	2.8	`true`
Kramden	150	150	1.0	`false` (GPA too low)

Complete function `IsSenior` below.

```
bool IsSenior(const StudentInfo & student)
// postcondition: returns true if this student's credit hours ≥ 125
//                and GPA ≥ 2.0; otherwise, returns false
{
  if (student.creditHours ≥125)
    {
      if (student.GPA ≥ 2.0)
        {
          return true;
        }
    }
  else
    return false;
}
```

(c) Write function `FillSeniorList`, as started below. `FillSeniorList` determines which students in the array `roster` are seniors and copies those students' records to the array `seniors`. It should also set the value of parameter `numSeniors` to be the number of seniors in the array `seniors`.

In writing `FillSeniorList`, you may call function `IsSenior` specified in part (b). Assume that `IsSenior` works as specified, regardless of what you wrote in part (b).

Complete function `FillSeniorList` below. Assume that `FillSeniorList` is called only with parameters that satisfy its precondition.

```
void FillSeniorList(const apvector<StudentInfo> & roster,
                    int numStudents, apvector<StudentInfo> & seniors,
                    int & numSeniors)
// precondition: roster contains numStudents records,
//               0 < numStudents ≤ roster.length(),
//               and seniors is large enough to hold all of
//               the seniors' records
```

```
{
  bool a;
  int b=0;
  numSeniors =0;
  for (int i =0; i<numStudents; i++)
     { a= IsSenior (roster[i].name)
       if(a==true)
          {
           seniors [b].name = roster[i].name;
           seniors [b].creditHours = roster[i].creditHours;
           seniors [b].gradePoints = roster[i].gradePoints;
           seniors [b].GPA = roster[i].GPA;
           numSeniors ++;
           b++;
          }
     }
}
```

Commentary:

(a) 1½ out of 3. Student commits an array bounds error (<= instead of <) and does not handle the `creditHours == 0` case.

(b) 1 out of 2. Poor choice to use nested `if`'s results in missing case.

(c) 3 out of 4. Incorrect call to `IsSenior`.

Poor Solution: 3 points ($2^1/_2$, rounds up)

1. Assume that student records are implemented using the following declaration.

```
struct StudentInfo
{
   apstring name;
   int creditHours;
   double gradePoints;
   double GPA;
};
```

(a) Write function `ComputeGPA`, as started below. `ComputeGPA` should fill in the `GPA` data member for the first `numStudents` records in its `apvector` parameter `roster`. A student's GPA (grade point average) is computed by dividing `gradePoints` by `creditHours`. The GPA for a student with 0 credit hours should be set to 0.

Complete function `ComputeGPA` below. Assume that `ComputeGPA` is called only with parameters that satisfy its precondition.

```
void ComputeGPA(apvector<StudentInfo> & roster, int numStudents)
// precondition:  roster contains numStudents records,
//                0 < numStudents ≤ roster.length(), in which the
//                name, creditHours and gradePoints data members
//                have been initialized.
// postcondition: The GPA data member for the first numStudents records
//                in roster has been calculated.
{
 if(creditHours == 0)
      GPA= 0;

 else
      GPA= gradePoints/creditHours;

 return  GPA;
}
```

(b) Write function `IsSenior`, as started below. `IsSenior` should return `true` if the given student has at least 125 credit hours and has a GPA of at least 2.0; otherwise, `IsSenior` should return `false`.

For example:

student				Result of the call `IsSenior(student)`
name	creditHours	gradePoints	GPA	
King	45	171	3.8	`false` (not enough credit hours)
Norton	128	448	3.5	`true`
Solo	125	350	2.8	`true`
Kramden	150	150	1.0	`false` (GPA too low)

Complete function `IsSenior` below.

```
bool IsSenior(const StudentInfo & student)
// postcondition: returns true if this student's credit hours ≥ 125
//                and GPA ≥ 2.0; otherwise, returns false
{
    if (creditHours ==125 && GPA >= 2.0)
        return true;
    else
        return false;
}
```

(c) Write function `FillSeniorList`, as started below. `FillSeniorList` determines which students in the array `roster` are seniors and copies those students' records to the array `seniors`. It should also set the value of parameter `numSeniors` to be the number of seniors in the array `seniors`.

In writing `FillSeniorList`, you may call function `IsSenior` specified in part (b). Assume that `IsSenior` works as specified, regardless of what you wrote in part (b).

Complete function `FillSeniorList` below. Assume that `FillSeniorList` is called only with parameters that satisfy its precondition.

```
void FillSeniorList(const apvector<StudentInfo> & roster,
                    int numStudents, apvector<StudentInfo> & seniors,
                    int & numSeniors)
// precondition: roster contains numStudents records,
//               0 < numStudents ≤ roster.length(),
//               and seniors is large enough to hold all of
//               the seniors' records

{
    if (numStudents == IsSenior)
        roster(seniors);


    numSeniors.resize (seniors.length());
    return 0;
}
```

Commentary:

(a) $^1/_2$ out of 3. Generously earns the two points for identifying and calculating GPA in the zero and non-zero case *but* has two usage errors: improper (non-existent) struct notation (-1), and returns GPA when the function is void ($-^1/_2$). Thus, 2 points were earned and there were $1^1/_2$ usage deductions on this part, for a net of $^1/_2$.

(b) 2 out of 2.

(c) 0 out of 4.

Computer Science A: Question 2

Overview

This question, a harder one-dimensional array question, asked the students to reason about an ordered array of strings. Students needed to both understand the string abstraction and develop correct algorithms for manipulating the ordered array. The solution-space for this question was very large, making it both very interesting and very difficult to grade. The question had a mean of 3.72 (4.89 if 0's and –'s are not counted) and a very even distribution of scores, making it an effective discriminator across all grade cutpoints.

In part (a), the student was required to return the "insertion position" of a given string in the ordered array. Many students checked for equality only (a standard search) instead of what was required. As is often the case with array questions, students also went beyond the valid region of the array (by looping to `wordList.length()` instead of `numWords`). In addition, many students did not correctly handle the `apstring` abstraction, attempting to use invalid comparisons (e.g., `strcmp`) instead of valid `apstring` member functions and operators (e.g., `==`). In the same vein, students also resorted to a character-by-character comparison of strings, which was rarely done correctly.

In part (b), the student was asked to (potentially) insert a new word in the correct position in the ordered array, shifting elements as appropriate. Many students failed to properly exploit the `WordIndex` abstraction that part (a) afforded them in solving part (b). Another common error was to use the element at the position returned by `WordIndex` to determine whether to insert the new element. If this position was equal to `numWords`, a comparison to an array location whose contents were unknown was being performed. Finally, many students inserted the new word multiple times or had problems shifting data in the array after a correct insertion.

Scoring Guidelines

Part A: `WordIndex` (4 points)

- **+2** find and return insertion point or current location for any word >= **first** item and <= **last** (must have loop/recursion and comparison of `wordList[i]` with `word`)
  - **+1** attempt
  - **+1** correct
- **+2** boundary cases
  - **+1** determines if before the first item (returns 0) OR after the last item (returns `numWords`)
  - **+1** correct value returned on both boundaries (including empty list)

Note: a solution that consistently treats the list in decreasing alphabetical order can get both boundary case points (as well as first attempt point)

Part B: `InsertInOrder` (5 points)

- **+1** guard against insertion if and only if word is already in the list
  - **+1/2** attempt (insufficient to examine only one element or only compare `WordIndex` to 0)
  - **+1/2** correct
- **+2** shift list items or sort list
  - **+1** attempt (must use a loop/recursion to copy multiple list items from at least two different source locations to at least two different destination locations)
  - **+1** correct items in positions [`numWords` down to `WordIndex(word, wordList, numWords) + 1`] are modified
- **+1 1/2** locate position and insert `word` exactly once (insert almost always needs to follow shift)
  - **+1/2** attempt to locate **or** insert `word`
  - **+1** correct
- **+1/2** increment `numWords` at most once and in conjunction with an insert attempt

Usage:

- **-1** `cout << result`
- **-1** destructive resize (benign resize loses 0 usage points)
- **-1/2** `WordIndex(___, ___, ___) = ___`
- **-1/2** `WordIndex(const apstring & word, const apvector<apstring> & wordList, int numWords)`
- **-1/2** vector access as . instead of [ ]
- **-1/2** temp variable declared improperly (e.g., wrong type or length)
- **-0** no variables declared

Note: assignment to `word` not allowed (loses "correct")

Excellent Solution: 9 points

2.

(a) Write function `WordIndex`, as started below. The array `wordList` contains `numWords` strings in alphabetical order. If `word` is already in `wordList`, then `WordIndex` should return the index of `word` in `wordList`. Otherwise, `WordIndex` should return the index of the first string in `wordList` that comes after `word` in alphabetical order; it should return `numWords` if `word` comes after all of the strings in `wordList` in alphabetical order.

For example, assume that array `wordList` is as follows:

0	1	2	3
"apple"	"berry"	"pear"	"quince"

Function Call	Value Returned
`WordIndex("air", wordList, 4)`	0
`WordIndex("apple", wordList, 4)`	0
`WordIndex("orange", wordList, 4)`	2
`WordIndex("raspberry", wordList, 4)`	4

Complete function `WordIndex` below. Assume that `WordIndex` is called only with parameters that satisfy its precondition.

```
int WordIndex(const apstring & word,
              const apvector<apstring> & wordList, int numWords)
// precondition: wordList contains numWords strings in alphabetical
//               order, 0 ≤ numWords < wordList.length()
```

```
{
    int x=0;
    while((x<numWords) && (wordList[x]<word))
        x++;

    return x;
}
```

(b) Write function `InsertInOrder`, as started below. The array `wordList` contains `numWords` strings in alphabetical order. If the string `word` is already in `wordList`, `InsertInOrder` should not change any of its parameters. Otherwise, it should insert `word` into `wordList` in alphabetical order (i.e., all values greater than `word` should be moved one place to the right to make room for `word`), and it should also increment `numWords` by 1. Assume that `wordList.length()` is greater than `numWords`.

In the examples below, `numWords = 3` before the following call is made.

```
InsertInOrder("pear", wordList, numWords)
```

Before the call wordList	After the call wordList	numWords
"apple" "berry" "quince"	"apple" "berry" "pear" "quince"	4
"apple" "berry" "pear"	"apple" "berry" "pear"	3
"apple" "fig" "peach"	"apple" "fig" "peach" "pear"	4
"quince" "raisin" "tart"	"pear" "quince" "raisin" "tart"	4

In writing `InsertInOrder`, you may include calls to function `WordIndex` specified in part (a). Assume that `WordIndex` works as specified, regardless of what you wrote in part (a).

Complete function `InsertInOrder` below. Assume that `InsertInOrder` is called only with parameters that satisfy its precondition.

```
void InsertInOrder(const apstring & word,
                   apvector <apstring> & wordList, int & numWords)
// precondition:  wordList contains numWords strings in alphabetical
//                order, 0 ≤ numWords < wordList.length()
// postcondition: if word was already in wordList, then wordList and
//                numWords are unchanged;
//                otherwise, word has been inserted into wordList in
//                sorted order, and numWords has been incremented by 1
{
    int x, index;
    for(x=0; x< numWords; x++)
        if(wordList[x]==word)
            return;
    index=WordIndex(word, wordList, numWords);
    for( x=numWords; x>index; x--)
        wordList[x]= wordList[x-1];
    wordList[index]=word;
    numWords++;
}
```

Commentary:

(a) Student uses a standard while loop solution.
(b) Student makes a separate, explicit check to see if the word was already in the array, which is fine.

Good Solution: 6 points

2.

(a) Write function WordIndex, as started below. The array wordList contains numWords strings in alphabetical order. If word is already in wordList, then WordIndex should return the index of word in wordList. Otherwise, WordIndex should return the index of the first string in wordList that comes after word in alphabetical order; it should return numWords if word comes after all of the strings in wordList in alphabetical order.

For example, assume that array wordList is as follows:

0	1	2	3
"apple"	"berry"	"pear"	"quince"

Function Call	Value Returned
WordIndex("air", wordList, 4)	0
WordIndex("apple", wordList, 4)	0
WordIndex("orange", wordList, 4)	2
WordIndex("raspberry", wordList, 4)	4

Complete function WordIndex below. Assume that WordIndex is called only with parameters that satisfy its precondition.

```
int WordIndex(const apstring & word,
              const apvector<apstring> & wordList, int numWords)
// precondition: wordList contains numWords strings in alphabetical
//               order, 0 ≤ numWords < wordList.length()
```

```
{ int i;

  for (    i = 0; i < numWords; i++)
      if((word == wordList[i])
         ||(wordList[i] >word))
      return i;

}
```

(b) Write function `InsertInOrder`, as started below. The array `wordList` contains `numWords` strings in alphabetical order. If the string `word` is already in `wordList`, `InsertInOrder` should not change any of its parameters. Otherwise, it should insert `word` into `wordList` in alphabetical order (i.e., all values greater than `word` should be moved one place to the right to make room for `word`), and it should also increment `numWords` by 1. Assume that `wordList.length()` is greater than `numWords`.

In the examples below, `numWords = 3` before the following call is made.

`InsertInOrder("pear", wordList, numWords)`

Before the call	After the call	
wordList	wordList	numWords
"apple" "berry" "quince"	"apple" "berry" "pear" "quince"	4
"apple" "berry" "pear"	"apple" "berry" "pear"	3
"apple" "fig" "peach"	"apple" "fig" "peach" "pear"	4
"quince" "raisin" "tart"	"pear" "quince" "raisin" "tart"	4

In writing `InsertInOrder`, you may include calls to function `WordIndex` specified in part (a). Assume that `WordIndex` works as specified, regardless of what you wrote in part (a).

Complete function `InsertInOrder` below. Assume that `InsertInOrder` is called only with parameters that satisfy its precondition.

```
void InsertInOrder(const apstring & word,
                   apvector <apstring> & wordList, int & numWords)
// precondition:  wordList contains numWords strings in alphabetical
//                order, 0 ≤ numWords < wordList.length()
// postcondition: if word was already in wordList, then wordList and
//                numWords are unchanged;
//                otherwise, word has been inserted into wordList in
//                sorted order, and numWords has been incremented by 1
```

```
{ int i;
  int insPos = WordIndex(word, WordList, numWords);

   if (wordlist[insPos] != word)
   {
      wordList.resize(wordList.length() + 1);
      for (i = WordList.length(); i ≥ insPos; i--)
         wordList[i] = wordList[i-1]
      WordList[insPos] = word
   }
} // end function
```

Commentary:

(a) 3 out of 4. Student does not get all boundary cases correct.

(b) 3 out of 5. Student loses correctness 1/2 point if word is already there and correctness point on shift. The student also does not increment `numWords` after inserting. Note that the resize that is performed is non-destructive (benign) and so does not lose any points.

Poor Solution: 3 points

2.

(a) Write function `WordIndex`, as started below. The array `wordList` contains `numWords` strings in alphabetical order. If `word` is already in `wordList`, then `WordIndex` should return the index of `word` in `wordList`. Otherwise, `WordIndex` should return the index of the first string in `wordList` that comes after `word` in alphabetical order; it should return `numWords` if `word` comes after all of the strings in `wordList` in alphabetical order.

For example, assume that array `wordList` is as follows:

0	1	2	3
"apple"	"berry"	"pear"	"quince"

<u>Function Call</u>	<u>Value Returned</u>
`WordIndex("air", wordList, 4)`	0
`WordIndex("apple", wordList, 4)`	0
`WordIndex("orange", wordList, 4)`	2
`WordIndex("raspberry", wordList, 4)`	4

Complete function `WordIndex` below. Assume that `WordIndex` is called only with parameters that satisfy its precondition.

```
int WordIndex(const apstring & word,
              const apvector<apstring> & wordList, int numWords)
// precondition: wordList contains numWords strings in alphabetical
//               order, 0 ≤ numWords < wordList.length()
{
  int x;
  for(x=0; x<=numwords; x++)
  if(word > wordlist[x])
   return (x+1);
  else if (word == Wordlist[x])
   return (x);
  else if(word < wordlist [x])
   return (x-1);
}
```

(b) Write function `InsertInOrder`, as started below. The array `wordList` contains `numWords` strings in alphabetical order. If the string `word` is already in `wordList`, `InsertInOrder` should not change any of its parameters. Otherwise, it should insert `word` into `wordList` in alphabetical order (i.e., all values greater than `word` should be moved one place to the right to make room for `word`), and it should also increment `numWords` by 1. Assume that `wordList.length()` is greater than `numWords`.

In the examples below, `numWords = 3` before the following call is made.

```
InsertInOrder("pear", wordList, numWords)
```

Before the call	After the call	
wordList	wordList	numWords
"apple" "berry" "quince"	"apple" "berry" "pear" "quince"	4
"apple" "berry" "pear"	"apple" "berry" "pear"	3
"apple" "fig" "peach"	"apple" "fig" "peach" "pear"	4
"quince" "raisin" "tart"	"pear" "quince" "raisin" "tart"	4

In writing `InsertInOrder`, you may include calls to function `WordIndex` specified in part (a). Assume that `WordIndex` works as specified, regardless of what you wrote in part (a).

Complete function `InsertInOrder` below. Assume that `InsertInOrder` is called only with parameters that satisfy its precondition.

```
void InsertInOrder(const apstring & word,
                   apvector <apstring> & wordList, int & numWords)
// precondition:  wordList contains numWords strings in alphabetical
//                order, 0 ≤ numWords < wordList.length()
// postcondition: if word was already in wordList, then wordList and
//                numWords are unchanged;
//                otherwise, word has been inserted into wordList in
//                sorted order, and numWords has been incremented by 1
```

```
{ int x, y, z;
  for (x = 0; x <= wordlist.length; x++ )
     if (wordlist [x]< word )
      { for y = x; y <= wordlist.length; y++)
        {wordlist [x+1] = wordlist[x];
        }
        wordlist [x]= word;
      }

  for (y=0; x<= wordlist.length; x++)
     if(wordlist [x] >word)
      {for(z=x; z <= wordlist.length; z--)
        {wordlist[x-1]= wordlist[x])
        }
        wordlist[x] = word)
        }
}
```

Commentary:

(a) 1 out of 4. Student earns only the general attempt point.

(b) 2 out of 5. Student earns only the attempt points.

Computer Science A: Question 3; AB: Question 2

Overview

This question involved reasoning about the AP Computer Science Large Integer Case Study. In particular, students were required to add a member function to the `BigInt` class as well as implement division. As a division question using repeated subtraction had been asked on the 1997 exam, students were constrained to use binary search in their division algorithm for part (b) of this exam. It was explicitly prohibited to use any algorithm other than binary search, so a "broader than textbook" interpretation of binary search was used to determine whether a student's solution was headed down a potentially correct path and thus could be graded.

As was the case in years past, many A students (just under 40) either omitted or zeroed the entire question, resulting in a mean score of 2.88 (but this rises to 4.70 if 0's and –'s are not counted). In contrast, only 17.5 percent of AB students omitted or zeroed the question, resulting in a mean of 3.75 for AB students (4.56 without 0's and –'s). As was done on a non-case study question in 1998, the students taking the A exam were provided with a more specific algorithmic hint in the construction of their code on part (b) than were the students taking the AB exam. This explains the higher mean for A students when 0's and –'s are not counted. This question discriminated well at the 3-4 grade cutpoint on the A exam and the 3-4 and 4-5 grade cutpoints on the AB exam.

Since the algorithm for part (a) was very tightly prescribed on both exams, most students did well. Common errors included loop bounds that were incorrect or loops that went in the wrong direction. It was acceptable to access the digits directly (instead of using abstractions such as `GetDigit` or `ChangeDigit`), but students who did so usually neglected to compensate for the fact that the digits are stored as an array of characters, not integers.

In part (b), the A students were aided by the algorithmic hint and so tended to write solutions that were closer to the mark. Most AB students seemed to have a difficult time translating the invariant into working code that moved the boundaries of the search appropriately. There were also many different errors with respect to calling the `Div2()` member function written in part (a).

Scoring Guidelines

Part A: `Div2` (4 points)

+1/2	init `carryDown` (must later attempt to assign value based on a digit)	
+1	loop over digits	
	+1/2	begin at most significant end (e.g., `NumDigits( ) - 1`)
	+1/2	correct
+1	compute and store current digit (repeatedly)	
+1	compute and store `carryDown` (repeatedly)	
+1/2	normalize quotient (must have attempted to change a digit)	

Part B: `DivPos` (and `operator/`) (5 points)

In order to receive any points for Part B, must repeatedly

– compute the approximate midpoint

OR

– potentially update both bounds based on a comparison

+1/2	declare and init `low`, `high`, and `mid` appropriately	
+1/2	loop until quotient found	
+1 1/2	compute and store `mid`	
	+1/2	sum bounds
	+1	divide by 2 and store (integer division is not defined for `BigInts`)
+1	identify which side of interval to search next	
+1	correctly assign to next value of `high/low`	
+1/2	return quotient	

Usage:

-1/2	improper syntax on `Div2` call
-1/2	`BigInt.NumDigits();`

Sample Student Responses

Excellent Solution: 9 points (8½, rounds up)

3. This question involves reasoning about the code from the Large Integer case study. A copy of the code is provided as part of this examination.

   (a) Write a new `BigInt` member function `Div2`, as started below. `Div2` should change the value of the `BigInt` to be the original value divided by 2 (integer division). Assume the `BigInt` is greater than or equal to 0. One algorithm for implementing `Div2` is:

   ```
   1. Initialize a variable carryDown to 0.

   2. For each digit, d, starting with the most significant digit,

      2.1 replace that digit with (d / 2) + carryDown

      2.2 let carryDown be (d % 2) * 5

   3. Normalize the result
   ```

   Complete member function `Div2` below.

```
void BigInt::Div2()
// precondition:  BigInt ≥ 0
{
        int carryDown=0;

        for (int x = myNumDigits-1; x>=0; x--)
        {
             int tempDig = GetDigit(x);
            ChangeDigit(x, ((tempDig /2)+carryDown));
             carryDown = ((tempDig % 2)*5);
              Normalize;
        }

}
```

(b) Write function `DivPos`, as started below. `DivPos` returns the quotient of the integer division of `dividend` by `divisor`. Assume that `dividend` and `divisor` are both positive values of type `BigInt`.

For example, assume that `bigNum1` and `bigNum2` are positive values of type `BigInt`:

bigNum1	bigNum2	DivPos(bigNum1, bigNum2)
18	9	2
17	2	8
8714	2178	4
9990	999	10

There are many ways to implement division; however, you must use a binary search algorithm to find the quotient of `dividend` divided by `divisor` in this problem. You will receive no credit on this part if you do not use a binary search algorithm.

One algorithm for implementing division using binary search is as follows:

1. Initialize `low` to `0` and `high` to `dividend`.
2. For each iteration,

   2.1 compute `mid = (low + high + 1)`

   2.2 divide `mid` by `2`

   2.3 if `mid * divisor` is larger than `dividend` (`mid` is too large to be the quotient) then set `high` equal to `mid - 1` else set `low` equal to `mid`.

3. When `low == high` the search terminates, and you should return `low`.

In writing function `DivPos`, you may call function `Div2` specified in part (a). Assume that `Div2` works as specified, regardless of what you wrote in part (a). You will receive no credit on this part if you do not use a binary search algorithm.

Complete function `DivPos` below. Assume that `DivPos` is called only with parameters that satisfy its precondition.

```
BigInt DivPos(const BigInt & dividend, const BigInt & divisor)
// precondition: dividend > 0, divisor > 0
{
    BigInt  low(0);
    BigInt  high(dividend);
    BigInt  mid;

    while (low != high)
    {
        mid = (low+high +1);
        mid.Div2;
        if ((mid* divisor) > dividend)
            high= (mid-1);
        else
            low = mid;
    }
    return(low);
}
```

Commentary:

(a) $3^1/_2$ out of 4. Student loses loop correctness $^1/_2$ point due to `x++` instead of `x--`.

(b) 5 out of 5. Standard solution, albeit with some missing `()`'s on the call to the `Div2` member function (an unpenalized usage error).

Good Solution: 6 points

2. This question involves reasoning about the code from the Large Integer case study. A copy of the code is provided as part of this examination.

   (a) Write a new `BigInt` member function `Div2`, as started below. `Div2` should change the value of the `BigInt` to be the original value divided by 2 (integer division). Assume the `BigInt` is greater than or equal to `0`. One algorithm for implementing `Div2` is:

   1. Initialize a variable `carryDown` to `0`.

   2. For each digit, `d`, starting with the most significant digit,

      2.1 replace that digit with `(d / 2) + carryDown`

      2.2 let `carryDown` be `(d % 2) * 5`

   3. Normalize the result.

   Complete member function `Div2` below.

```
void BigInt::Div2()
// precondition:  BigInt ≥ 0
{

    int carryDown = 0;

    for (int digit = myNumDigits - 1; digit >= 0; digit --) {
        changeDigit(digit, (myDigits [digit] / 2) + carryDown);
        carryDown = (myDigits [digit] % 2) * 5;
    }

    Normalize();

}
```

(b) Write a definition to overload the `/` operator, as started below. Assume that `dividend` and `divisor` are both positive values of type `BigInt`.

For example, assume that `bigNum1` and `bigNum2` are positive values of type `BigInt`:

bigNum1	bigNum2	bigNum1 / bigNum2
18	9	2
17	2	8
8714	2178	4
9990	999	10

There are many ways to implement division; however, you must use a binary search algorithm to find the quotient of `dividend` divided by `divisor` in this problem. You will receive no credit on this part if you do not use a binary search algorithm.

One possible algorithm for implementing division using binary search is as follows:

Let `low` and `high` represent a range in which the quotient is found.

Initialize `low` to `0` and `high` to `dividend`.

For each iteration, compute `mid = (low + high + 1)`, divide `mid` by `2`, and compare `mid * divisor` with `dividend` to maintain the invariant that `low ≤ quotient` and `high ≥ quotient`.

When `low == high`, the quotient has been found.

In writing function `operator/` you may call function `Div2` specified in part (a). Assume that `Div2` works as specified, regardless of what you wrote in part (a). You will receive NO credit on this part if you do not use a binary search algorithm.

Complete `operator/` below. Assume that `operator/` is called only with parameters that satisfy its precondition.

```
BigInt operator/ (const BigInt & dividend, const BigInt & divisor)
// precondition:  dividend > 0, divisor > 0
{
  bigInt low(0), high = dividend, one(1);
  bigInt mid
  while (low != high) {
     mid = low + high + one;
     mid.Div2();
     if (mid * divisor >= dividend)
        high = mid;
     else low = mid;
  }
  return high
}
```

Commentary:

(a) 2 out of 4. Using `myDigits` costs this student, as digits are stored as an array of `char`, not `int`. Even if `myDigits` were used correctly, the failure to store the current digit means that the `carryDown` computation is tainted and thus would be incorrect.

(b) 4 out of 5. Student incorrectly updates `high`.

Poor Solution: 3 points ($2^1/_2$, rounds to 3)

3. This question involves reasoning about the code from the Large Integer case study. A copy of the code is provided as part of this examination.

   (a) Write a new `BigInt` member function `Div2`, as started below. `Div2` should change the value of the `BigInt` to be the original value divided by 2 (integer division). Assume the `BigInt` is greater than or equal to 0. One algorithm for implementing `Div2` is:

   1. Initialize a variable carryDown to 0.

   2. For each digit, d, starting with the most significant digit,

      2.1 replace that digit with (d / 2) + carryDown

      2.2 let carryDown be (d % 2) * 5

   3. Normalize the result

   Complete member function `Div2` below.

```
void BigInt::Div2()
// precondition:  BigInt ≥ 0
{
  int carrydown=0;
  int len = numDigits();
  int k;
  int x;
  for(k=len-1; k<=0; k--)
    {
      x = GetDigit(k);
      x = (x/2) + carrydown;
      carrydown = (x%2) * 5;
    }
}
```

(b) Write function DivPos, as started below. DivPos returns the quotient of the integer division of dividend by divisor. Assume that dividend and divisor are both positive values of type BigInt.

For example, assume that bigNum1 and bigNum2 are positive values of type BigInt:

bigNum1	bigNum2	DivPos(bigNum1, bigNum2)
18	9	2
17	2	8
8714	2178	4
9990	999	10

There are many ways to implement division; however, you must use a binary search algorithm to find the quotient of dividend divided by divisor in this problem. You will receive no credit on this part if you do not use a binary search algorithm.

One algorithm for implementing division using binary search is as follows:

1. Initialize low to 0 and high to dividend.
2. For each iteration,

   2.1 compute mid = (low + high + 1)

   2.2 divide mid by 2

   2.3 if mid * divisor is larger than dividend (mid is too large to be the quotient) then set high equal to mid - 1 else set low equal to mid.

3. When low == high the search terminates, and you should return low.

In writing function `DivPos`, you may call function `Div2` specified in part (a). Assume that `Div2` works as specified, regardless of what you wrote in part (a). You will receive no credit on this part if you do not use a binary search algorithm.

Complete function `DivPos` below. Assume that `DivPos` is called only with parameters that satisfy its precondition.

```
BigInt DivPos(const BigInt & dividend, const BigInt & divisor)
// precondition: dividend > 0, divisor > 0
{ int low = 0;
  int high= dividend;
  int mid;
  while (low != high)
     { mid = (low + high + 1)
       if((mid * divisor) > dividend)
          high = mid-1;
          else low = mid;
     }
     return low;
}
```

Commentary:

(a) 1 out of 4. Student earns the init `carryDown` and loop attempt 1/2 points.

(b) 1 1/2 out of 5. Student earns the loop, sum calculation, and return quotient 1/2 points. Since there is no division by two, the student cannot earn the interval test or update bounds points.

Computer Science A: Question 4; AB: Question 1

Overview

This question was substantively different from what would have been asked in Pascal. Students were provided a context for a class, given the declaration of the class, and asked to implement a constructor and two member functions for the class. While there was a lot of reading, the intent was to provide a real-life example in which to embed an interesting programming problem. The amount of reading and the relative position of this question on the A exam resulted in a comparatively low mean of 1.97 (this improves to 3.41 without the 0's and –'s). The question discriminated best at the 4-5 grade cutpoint on the A exam. AB students fared much better, with a mean of 4.60 and a very even distribution of scores, making this an effective discriminator across all grade cutpoints for the AB exam.

In part (a), the student was asked to write a constructor for the Quilt class that required reading data from a file. Many students had no idea how to read data from a file under control of a loop, or how the extraction operator (>>) worked. Students and teachers are strongly encouraged to examine the canonical solution for this question to see how to properly read data from an external file.

In part (b), the student was asked to supply the body of the loop for the `PlaceFlipped()` member function. Most students received full credit for this part, or were off by one in their index calculations. As it was unclear from the problem specification, and not clarified by the example provided, a solution was deemed correct if the student reflected the block about its horizontal axis, or rotated it 180 degrees.

In part (c), the student was asked to alternately place blocks and flipped blocks into a matrix that represented the entire quilt. Again, as it was unclear from the problem specification, and not clarified by the example, full credit was given to both checkerboard alternation and alternation that resulted in alternating vertical stripes if there were an even number of columns. An alternating row pattern was not given full credit. Most students received some credit on this part, with the most common errors being confusion between the block indices and the larger matrix indices, forgetting to declare a local matrix to return, and forgetting to return the matrix once finished.

Scoring Guidelines

Part A: `Quilt::Quilt` (3 points)

+1/2 read row and column dimensions from `inFile`
Note: consistent use of `cin`, or no attempt to extract from `inFile`, loses this 1/2 point **and** both "read from `inFile` into my `Block[r][c]`" 1/2 points (see below)

+1/2 resize matrix appropriately

+1 loop over rows and columns
- **+1/2** attempt
- **+1/2** correct

+1 read from `inFile` into my `Block[r][c]` (consistent stream misuse loses both 1/2 points)
- **+1/2** attempt
- **+1/2** correct

Usage: -1 incorrectly overwrite `myRowsOfBlocks/myColsOfBlocks`

-1/2 `var >> inFile` or `var << inFile`

Notes:

```
>> // skips whitespace
inFile.get( );    // reads and returns next character, including whitespace (can be OK)
inFile.get(ch);   // reads next character into ch, including whitespace (can be OK)
inFile.ignore(num,'\n');            // can be OK
getline(inFile,aString);      // can be OK
getch( );    // NOT OK
getchar( );         // NOT OK
>> endl;     // NOT OK
```

Part B: `Quilt::PlaceFlipped` (1 point)

+1/2 attempt from `myBlock` into `qmat`

+1/2 correct (reflection about X or rotation OK, but reflection about Y axis loses this half)

Part C: `Quilt::QuiltToMat` (5 points)

+2 declaration/sizing/return of local matrix
- **+1/2** attempt
- **+1** correct
- **+1/2** return matrix (matrix must be declared to get this half)

+1 loop over entire structure
- **+1/2** attempt
- **+1/2** correct

+2 `PlaceBlock/PlaceFlipped` alternation
- **+1** attempt at alternation (within loop context)
- **+1** consistent and correct (matrix must be declared to get this point)

Notes: checkerboard OK
stripes that result from flipping every other block OK

Sample Student Responses

Please note that only the portions of the question that contain student responses have been reproduced here. For the entire question, see pages 82-86.

Excellent Solution: 8 points

(a) Write the code for the constructor that initializes a quilt, as started below. The constructor reads the block pattern for the main block from a file represented by the parameter `inFile`. You may assume the file is open and that the file contains the number of rows followed by the number of columns for the block, followed by the characters representing the pattern. For example, the file `pattern`, which contains the pattern for the first block in the quilt shown above, would look like this:

```
4 5
x...x
.x.x.
..x..
..x..
```

The constructor also sets the number of rows and columns of blocks which make up the entire quilt in the initializer list.

Complete the constructor below. Assume that the constructor is called only with parameters that satisfy its precondition.

```
Quilt::Quilt(istream & inFile, int rowsOfBlocks, int colsOfBlocks)
   : myBlock(0, 0), myRowsOfBlocks(rowsOfBlocks),
     myColsOfBlocks(colsOfBlocks)
// precondition:  inFile is open, rowsOfBlocks > 0, colsOfBlocks > 0
// postcondition: myRowsOfBlocks and myColsOfBlocks are initialized to
//                the number of rows and columns of blocks that make up
//                the quilt; myBlock has been resized and
//                initialized to the block pattern from the
//                stream inFile.
{ int row, col;
   char temp;
   inFile >> row >> col;
   myBlock.resize(row, col);
   for (int x = 0; x < row; x++)
    { for(int y=0; y< col ; y++)
       {
          infile >> temp;
          myBlock[x][y] = temp;
       }
    }
   myRowsOfBlocks = rowsOfBlocks;
   myColsOfBlocks = colsOfBlocks;
 }
```

(b) Complete the member function `PlaceFlipped` below. Assume that `PlaceFlipped` is called only with parameters that satisfy its precondition.

```
void Quilt::PlaceFlipped(int startRow, int startCol,
                         apmatrix<char> & qmat)
// precondition:  startRow ≥ 0; startCol ≥ 0;
//                startRow + myBlock.numrows() ≤ qmat.numrows();
//                startCol + myBlock.numcols() ≤ qmat.numcols();
// postcondition: a flipped version of myBlock has been copied into the
//                matrix qmat with its upper-left corner at the position
//                startRow, startCol
{
  int r, c;

  for (r = 0; r < myBlock.numrows(); r++)
  {
    for (c = 0; c < myBlock.numcols(); c++)
    {
      qmat[startRow+r][startCol+c] = myBlock[myBlock.numrows()-1-r][c];

    }
  }
}
```

(c) Write the member function `QuiltToMat`, as started below. `QuiltToMat` returns a matrix representing the whole quilt in such a way that the main block alternates with the flipped version of the main block, as shown in the original example. If `Q` represents the example quilt, then the call `Q.QuiltToMat()` would return a matrix of characters with the given block placed starting with the upper-left corner at position 0, 0; the flipped block placed with its upper-left corner at position 0, 5; the given block placed with its upper-left corner at position 0, 10; the flipped block placed with its upper-left corner at position 4, 0, and so on.

In writing `QuiltToMat`, you may call functions `PlaceBlock` and `PlaceFlipped` specified in part (b). Assume that `PlaceBlock` and `PlaceFlipped` work as specified, regardless of what you wrote in part (b).

Complete the member function `QuiltToMat` below.

```
apmatrix<char> Quilt::QuiltToMat()
{
  apmatrix<char> mat(myRowsOfBlocks * myBlock.numrows(), myColsOfBlock * myBlock.numcols(
  int x,y;
  for (x=0; x< mat.numrows()-1; x+= myBlock.numrows())
  { int s = x % 2;
    for(y=0; y< mat.numcols()-1; y+= myBlock.numCols())
    {
      if (s ==0)
      {
        PlaceBlock(x,y, mat); s++;
      }
      else
      {
        PlaceFlipped(x,y, mat); s--;
      }
    }
  }
}
```

Commentary:

Student earns all points except the return matrix and loop correctness 1/2 points on part (c).

Good Solution: 6 points

(a) Write the code for the constructor that initializes a quilt, as started below. The constructor reads the block pattern for the main block from a file represented by the parameter `inFile`. You may assume the file is open and that the file contains the number of rows followed by the number of columns for the block, followed by the characters representing the pattern. For example, the file `pattern`, which contains the pattern for the first block in the quilt shown above, would look like this:

```
4 5
x...x
.x.x.
..x..
..x..
```

The constructor also sets the number of rows and columns of blocks which make up the entire quilt in the initializer list.

Complete the constructor below. Assume that the constructor is called only with parameters that satisfy its precondition.

```
Quilt::Quilt(istream & inFile, int rowsOfBlocks, int colsOfBlocks)
  : myBlock(0, 0), myRowsOfBlocks(rowsOfBlocks),
    myColsOfBlocks(colsOfBlocks)
// precondition:  inFile is open, rowsOfBlocks > 0, colsOfBlocks > 0
// postcondition: myRowsOfBlocks and myColsOfBlocks are initialized to
//                the number of rows and columns of blocks that make up
//                the quilt; myBlock has been resized and
//                initialized to the block pattern from the
//                stream inFile.
```

```
{int r,c,i,j;
  apstring Line;
  inFile >>r;
  inFile >>c;
  getline (inFile, Line); // to get the end of Line marker for the 1st Li
  myBlock.resize(r,c);
  for (i=0; i<r; i++)
   {for(j=0;j<c;j++)
        inFile >> myBlock[i][j];
    getline(inFile, Line);// end of Line marker.
   }//for

  myRowsOfBlocks =r;
  myColsOfBlocks =c;
}
```

(b) Complete the member function `PlaceFlipped` below. Assume that `PlaceFlipped` is called only with parameters that satisfy its precondition.

```
void Quilt::PlaceFlipped(int startRow, int startCol,
                          apmatrix<char> & qmat)
// precondition:  startRow ≥ 0; startCol ≥ 0;
//                startRow + myBlock.numrows() ≤ qmat.numrows();
//                startCol + myBlock.numcols() ≤ qmat.numcols();
// postcondition: a flipped version of myBlock has been copied into the
//                matrix qmat with its upper-left corner at the position
//                startRow, startCol
{
  int r, c;

  for (r = 0; r < myBlock.numrows(); r++)
  {
    for (c = 0; c < myBlock.numcols(); c++)
    {
     qmat[startRow+r][startCol+c] =
    }
                    myBlock[myRowsOfBlocks-r-1][c];
  }
}
```

(c) Write the member function `QuiltToMat`, as started below. `QuiltToMat` returns a matrix representing the whole quilt in such a way that the main block alternates with the flipped version of the main block, as shown in the original example. If `Q` represents the example quilt, then the call `Q.QuiltToMat()` would return a matrix of characters with the given block placed starting with the upper-left corner at position 0, 0; the flipped block placed with its upper-left corner at position 0, 5; the given block placed with its upper-left corner at position 0, 10; the flipped block placed with its upper-left corner at position 4, 0, and so on.

In writing `QuiltToMat`, you may call functions `PlaceBlock` and `PlaceFlipped` specified in part (b). Assume that `PlaceBlock` and `PlaceFlipped` work as specified, regardless of what you wrote in part (b).

Complete the member function `QuiltToMat` below.

```
apmatrix<char> Quilt::QuiltToMat()
{ apmatrix<char> qui(12,15);
  int r, c;
  bool orig=true;
  for (r=0; r<12; r+=4)
    for(c=0; c<15; c+=5)
      {if(orig)
          PlaceBlock(r,c,qui);
       else
          PlaceFlipped(r,c,qui);
       orig=!orig;
      } //for
  return qui;
}
```

Commentary:

(a) 2 out of 3. Student earns all points, but loses 1 usage point by overwriting `myRowsOfBlocks` and `myColsOfBlocks`.

(b) $^1/_2$ out of 1.

(c) $3^1/_2$ out of 5. By using particular values instead of performing computations with the appropriate variables, the student loses the resize point and the loop correctness $^1/_2$ point.

Poor Solution: 4 points ($3^1/_2$, rounds up)

(a) Write the code for the constructor that initializes a quilt, as started below. The constructor reads the block pattern for the main block from a file represented by the parameter `inFile`. You may assume the file is open and that the file contains the number of rows followed by the number of columns for the block, followed by the characters representing the pattern. For example, the file `pattern`, which contains the pattern for the first block in the quilt shown above, would look like this:

```
4 5
x...x
.x.x.
..x..
..x..
```

The constructor also sets the number of rows and columns of blocks which make up the entire quilt in the initializer list.

Complete the constructor below. Assume that the constructor is called only with parameters that satisfy its precondition.

```
Quilt::Quilt(istream & inFile, int rowsOfBlocks, int colsOfBlocks)
  : myBlock(0, 0), myRowsOfBlocks(rowsOfBlocks),
    myColsOfBlocks(colsOfBlocks)
// precondition:  inFile is open, rowsOfBlocks > 0, colsOfBlocks > 0
// postcondition: myRowsOfBlocks and myColsOfBlocks are initialized to
//                the number of rows and columns of blocks that make up
//                the quilt; myBlock has been resized and
//                initialized to the block pattern from the
//                stream inFile.
```

```
{
inFile >> myRowsOfBlocks;
inFile >> myColsOfBlocks;
For(int x= 0; x< myRowsOfBlocks; x++)
   {
   for(int y=0; y <myColsOfBlocks ; y++)
     {
     inFile >> myBlock [x][y];
     }
   }
}.
```

(b) Complete the member function `PlaceFlipped` below. Assume that `PlaceFlipped` is called only with parameters that satisfy its precondition.

```
void Quilt::PlaceFlipped(int startRow, int startCol,
                         apmatrix<char> & qmat)
// precondition:  startRow ≥ 0; startCol ≥ 0;
//                startRow + myBlock.numrows() ≤ qmat.numrows();
//                startCol + myBlock.numcols() ≤ qmat.numcols();
// postcondition: a flipped version of myBlock has been copied into the
//                matrix qmat with its upper-left corner at the position
//                startRow, startCol
{
  int r, c;

  for (r = 0; r < myBlock.numrows(); r++)
  {
    for (c = 0; c < myBlock.numcols(); c++)
    {
      qmat[(start row + myBlock.numrows()) - r][(startcol + myBlock.numcols()) - c] =
                                                                      myBlock[r][c];
    }
  }
}
```

(c) Write the member function `QuiltToMat`, as started below. `QuiltToMat` returns a matrix representing the whole quilt in such a way that the main block alternates with the flipped version of the main block, as shown in the original example. If `Q` represents the example quilt, then the call `Q.QuiltToMat()` would return a matrix of characters with the given block placed starting with the upper-left corner at position 0, 0; the flipped block placed with its upper-left corner at position 0, 5; the given block placed with its upper-left corner at position 0, 10; the flipped block placed with its upper-left corner at position 4, 0, and so on.

In writing `QuiltToMat`, you may call functions `PlaceBlock` and `PlaceFlipped` specified in part (b). Assume that `PlaceBlock` and `PlaceFlipped` work as specified, regardless of what you wrote in part (b).

Complete the member function `QuiltToMat` below.

```
apmatrix<char> Quilt::QuiltToMat()
{
int r,c;
For(r=0; r < myRowsOfBlocks; r++)
  {
  For (c=0; c < myColsOfBlocks ; c++)
    {
    if ( r==(myRowsOfBlocks -1))
      {
      r++; c=0;
      PlaceBlock(r, c);
      }
    else
      {
      PlaceBlock(r, c);
      }
    if ((r+1)= (myRowsOfBlocks -1))
      {
      r++; c=0;
      PlaceFlipped(r+1,c);
      }
    else
      {
      PlaceFlipped(r+1, c);
      }
  }
```

Commentary:

(a) 1½ out of 3. Student fails to resize and overwrites `myRowsOfBlocks` and `myColsOfBlocks`.
(b) ½ out of 1.
(c) 1½ out of 5. Student fails to declare a local matrix (loses 3 points). Student also loses loop correctness ½ point.

Computer Science AB: Question 3

This question examined the student's facility with a basic dynamically allocated data structure, the linked list, and was very similar to, and somewhat easier than, linked list questions asked in the past. It required knowledge of simple pointer mechanics and list traversal combined with an array traversal in part (c). The mean score was 5.63, and the question was an effective discriminator at the lower grade cutpoints.

The first part tests basic linked list mechanics by asking the student to insert a new member into an existing list (that may have no members). The student needed to create a new node, assign to the data fields of the new node, and correctly insert the new node into the (potentially) existing list. Most students did very well on this question. Common errors included failure to distinguish between declaring a pointer and allocating new memory, and not understanding how to use the constructor that was provided to accomplish the task of initializing the data fields.

In part (b), the student was asked to traverse the entire list, counting the number of nodes that matched a particular criterion. Most students performed quite well on this part, with the most common errors being running off the end of the list, and forgetting to declare a temporary pointer to traverse the list.

In part (c), the student was asked to reason about traversing an array of linked lists to find a maximum, which caused some cognitive overload as students appeared to have trouble "switching gears" from one data structure (the linked list) to another (the array). The most common errors were incorrectly initializing or updating the maximum value, and printing out the incorrect information (either the wrong data or clubs that were not maximal).

Scoring Guidelines

Part A: `InsertMember` (2 points)

- **+1/2** new (can use non-existent default constructor)
- **+1/2** attempted assignment to both data fields and/or next
- **+1/2** correct assignment of all data members
- **+1/2** correct insertion of new node (watch for empty list failure)

Part B: `CountLevel` (3 points)

- **+1/2** temp pointer to traverse
- **+1** list traversal
  - **+1/2** attempt
  - **+1/2** correct (missing/incorrect init of temp pointer loses this half)
- **+1** count members
  - **+1/2** test
  - **+1/2** increment count (in context of a test)
- **+1/2** return correct accumulated value (missing init of count loses this half)

Part C: `PrintClubsWithMostInLevel` (4 points)

- **+1** loop over all clubs (no OBOB)
- **+1** compare and set `max` as appropriate
  - **+1/2** init (watch for unguarded init to 0th element)
  - **+1/2** correct (`CountLevel` must have correct parameters)
- **+1** identify appropriate clubs to print
- **+1** correctly print only appropriate clubs (only earns 1/2 if missing line break)

Usage: -1/2 memory leak

Note: confusion of -> vs. **.** results in failure to earn associated points rather than usage deduction (*except* in "attempt" and "identify" points)

Sample Student Responses

Please note that only the portions of the question that contain student responses have been reproduced here. For the entire question, see pages 156-160.

Excellent Solution: 9 points

(a) Write function `InsertMember`, as started below. `InsertMember` adds a student with the given name and respective level in high school to the given club.

For example, after the call `InsertMember("Taylor", 9, Club1)`, variable `Club1` might be as shown below. The diagram shows that the new member `"Taylor"` was inserted at the beginning of the list of members, but the student could have been inserted anywhere in the list.

```
  Club1
clubName:  German

memberList → [Taylor|9|] → [Roberts|10|] → [Schwab|11|] → [Shaw|9|] → [Hunt|10|] → [Rodger|10|/]
```

Complete function `InsertMember` below. Assume that `InsertMember` is called only with parameters that satisfy its precondition.

```
void InsertMember(const apstring & name, int level, Club & anyClub)
// precondition:  anyClub contains zero or more members, name does not
//                appear in anyClub, and level is 9, 10, 11, or 12
// postcondition: a new member with the given name and respective level
//                has been added to anyClub
{
anyClub.memberList = new Member(name, level, anyClub.memberList);
}
```

(b) Write function `CountLevel`, as started below. `CountLevel` counts and returns the number of club members of the specified level in `anyClub`.

For example, the call `CountLevel(Club1, 10)` returns 3, since there are 3 tenth graders in the German club. The call `CountLevel(Club2, 10)` returns 0 since there are no tenth graders in the Computer club.

Complete function `CountLevel` below. Assume that `CountLevel` is called only with parameters that satisfy its precondition.

```
int CountLevel(const Club & anyClub, int level)
// precondition:  level is 9, 10, 11, or 12
// postcondition: returns the number of members in anyClub
//                of that level
{
    int counter = 0;
    Member* ptr = anyClub.memberList;
    while (ptr)
    {
        if (ptr -> level == level)
            counter ++;
        ptr = ptr -> next;
    }
    return (counter);
}
```

(c) In writing `PrintClubsWithMostInLevel`, you may call function `CountLevel` specified in part (b). Assume `CountLevel` works as specified, regardless of what you wrote in part (b).

Complete function `PrintClubsWithMostInLevel` below. Assume that `PrintClubsWithMostInLevel` is called only with parameters that satisfy its precondition.

```
void PrintClubsWithMostInLevel(const apvector<Club> & clubsArray,
                               int level)
// precondition:  clubsArray contains clubsArray.length() clubs
// postcondition: prints the name of the club or clubs in clubsArray
//                that contain the largest number of members in a given
//                level in high school (9 - 12), one club per line.
```

```
{
    int max = 0;
    for (int i=0; i< clubsArray.length(); i++)
        if (CountLevel(clubsArray[i], level) > max)
            max = CountLevel(clubsArray[i], level)
    for(int i=0; i< clubsArray.length(); i++)
        if((CountLevel(clubsArray[i], level)) == max )
            cout << clubsArray[i].clubName << endl;
}
```

Commentary:

Student earns all 9 points with a standard solution.

Good Solution: 6 points ($5^1/_2$, rounds up)

(a) Write function `InsertMember`, as started below. `InsertMember` adds a student with the given name and respective level in high school to the given club.

For example, after the call `InsertMember("Taylor", 9, Club1)`, variable `Club1` might be as shown below. The diagram shows that the new member `"Taylor"` was inserted at the beginning of the list of members, but the student could have been inserted anywhere in the list.

```
Club1
clubName:   German

memberList --> [Taylor | 9] --> [Roberts | 10] --> [Schwab | 11] --> [Shaw | 9] --> [Hunt | 10] --> [Rodger | 10 /]
```

Complete function `InsertMember` below. Assume that `InsertMember` is called only with parameters that satisfy its precondition.

```
void InsertMember(const apstring & name, int level, Club & anyClub)
// precondition:  anyClub contains zero or more members, name does not
//                appear in anyClub, and level is 9, 10, 11, or 12
// postcondition: a new member with the given name and respective level
//                has been added to anyClub
{
    Member *temp = new member;
    temp.name = name;
    temp.level = level;
    temp->next = anyClub->memberlist;
    anyclub->memberlist = temp;
}
```

(b) Write function `CountLevel`, as started below. `CountLevel` counts and returns the number of club members of the specified level in `anyClub`.

For example, the call `CountLevel(Club1, 10)` returns 3, since there are 3 tenth graders in the German club. The call `CountLevel(Club2, 10)` returns 0 since there are no tenth graders in the Computer club.

Complete function `CountLevel` below. Assume that `CountLevel` is called only with parameters that satisfy its precondition.

```
int CountLevel(const Club & anyClub, int level)
// precondition:  level is 9, 10, 11, or 12
// postcondition: returns the number of members in anyClub
//                of that level
```

```
{ int sum;
  member*temp = anyClub->memberlist;
  while (temp != 0)
  {
     if(temp.level = level)
        sum++;
     temp = temp->next;
  }
}
```

(c) In writing `PrintClubsWithMostInLevel`, you may call function `CountLevel` specified in part (b). Assume `CountLevel` works as specified, regardless of what you wrote in part (b).

Complete function `PrintClubsWithMostInLevel` below. Assume that `PrintClubsWithMostInLevel` is called only with parameters that satisfy its precondition.

```
void PrintClubsWithMostInLevel(const apvector<Club> & clubsArray,
                               int level)
// precondition:  clubsArray contains clubsArray.length() clubs
// postcondition: prints the name of the club or clubs in clubsArray
//                that contain the largest number of members in a given
//                level in high school (9 - 12), one club per line.
```

```
{ int x, most=0;
  for(x=0; x<clubsArray.length; x++)
  {
     if(CountLevel(clubsArray[x], level()) > most;
        most = CountLevel(clubsArray[x], level());
  }
  for(x=0; x<clubsArray.length; x++)
  {
    if(CountLevel(clubsArray[x], level() = most
       cout<< clubsArray[x].clubname;
  }
```

Commentary:

(a) 1 out of 2. While technically there is no default constructor for `Member` (as a constructor with parameters was defined), students were not penalized for using the "non-existent" default constructor. Thus, the statement `Member * temp = new Member;` did not cause a usage deduction. However, the student did not correctly assign the data fields to temp and did not insert correctly.

(b) 1 out of 3. Student incorrectly initializes the temp pointer (also causes the loss of the loop correctness 1/2 point). The test is incorrect (but an attempted test is present, so the student earns the count 1/2 point) and the student fails to return the accumulated value.

(c) 3 1/2 out of 4. Because of a failure to insert a line break after the printing of each club name, the student loses 1/2 point.

Poor Solution: 3 points

(a) Write function `InsertMember`, as started below. `InsertMember` adds a student with the given name and respective level in high school to the given club.

For example, after the call `InsertMember("Taylor", 9, Club1)`, variable `Club1` might be as shown below. The diagram shows that the new member `"Taylor"` was inserted at the beginning of the list of members, but the student could have been inserted anywhere in the list.

```
Club1
clubName:   German

memberList --> [Taylor|9]-->[Roberts|10]-->[Schwab|11]-->[Shaw|9]-->[Hunt|10]-->[Rodger|10/]
```

Complete function `InsertMember` below. Assume that `InsertMember` is called only with parameters that satisfy its precondition.

```
void InsertMember(const apstring & name, int level, Club & anyClub)
// precondition:  anyClub contains zero or more members, name does not
//                appear in anyClub, and level is 9, 10, 11, or 12
// postcondition: a new member with the given name and respective level
// int num=0;     has been added to anyClub
  if ( m == NULL)
  {
     return num;
  }
  while( memberlist != NULL)
  {
     if (level = memberlist ->level)
        num++
     memberlist = memberlist ->next;
  }
  return num;

new Member temp;
temp-> name = name;
temp-> level = level;
temp-> next = NULL;
if ( memberlist == NULL)
   memberlist = temp;
else
{  temp-> next = member ->next;
   member -> next = temp;
}
```

(b) Write function `CountLevel`, as started below. `CountLevel` counts and returns the number of club members of the specified level in `anyClub`.

For example, the call `CountLevel(Club1, 10)` returns 3, since there are 3 tenth graders in the German club. The call `CountLevel(Club2, 10)` returns 0 since there are no tenth graders in the Computer club.

Complete function `CountLevel` below. Assume that `CountLevel` is called only with parameters that satisfy its precondition.

```
int CountLevel(const Club & anyClub, int level)
// precondition:   level is 9, 10, 11, or 12
// postcondition: returns the number of members in anyClub
//                of that level
```

```
int num=0;
if m==NULL
    return 0;
while (memberlist != NULL)
{   if (level = memberlist ->Level)
        num ++;
    memberlist = memberlist->next;
}
return num;
```

(c) In writing `PrintClubsWithMostInLevel`, you may call function `CountLevel` specified in part (b). Assume `CountLevel` works as specified, regardless of what you wrote in part (b).

Complete function `PrintClubsWithMostInLevel` below. Assume that `PrintClubsWithMostInLevel` is called only with parameters that satisfy its precondition.

```
void PrintClubsWithMostInLevel(const apvector<Club> & clubsArray,
                               int level)
// precondition:  clubsArray contains clubsArray.length() clubs
// postcondition: prints the name of the club or clubs in clubsArray
//                that contain the largest number of members in a given
//                level in high school (9 - 12), one club per line.
```

```
int max_count;
int m;
for(k=0; k<club.length(); k++)
{  if (clubArray[k].Countlevel >max-count)
   {  cout << club->name<<endl;
   }
elseif(club Array [k].coun-Level ==max-count)
   {  cout << club->name<<endl;
   }
```

Commentary:

(a) 1/2 out of 2. Student earns the attempt assignment 1/2 point.

(b) 1 1/2 out of 3. Student earns the list traversal, increment count, and return 1/2 points.

(c) 1 out of 4. Student earns the loop point.

Computer Science AB: Question 4

This question examined the student's facility with binary trees, the other common dynamically allocated data structure. This was a reasonably difficult tree question and the AB students performed well, although not as well as they did on Question AB3. The mean was 4.57 and this question discriminated well across all grade levels but was most useful at the 4-5 grade cutpoint.

Part (a) was the more difficult of the two parts, as it required the student to correctly keep track of their state as they recursively traversed the tree. Most students know to use recursion to traverse a binary tree and thus received at least some points. The most common errors included failing to properly guard the NULL case, and returning incorrect values when the name was not in the subtree.

Part (b) was much simpler than part (a) and almost all students received full credit. The most common error was a failure to correctly guard and handle the NULL case.

Scoring Guidelines

Part A: `PathLength` (7 points)

- +1 null test and return 0 (must have attempt at else)
  - **+1/2** attempt
  - **+1/2** correct
- +2 recursive calls on left/right
  - +1 attempt (must have both calls, "procedures" OK, need 3 parameters of appropriate type)
  - +1 correct (watch for not incrementing `level;` `level++` is incorrect)
- +2 compute and return result when name in subtree (note: presence in root is immaterial)
  - +1 attempt (must attempt to verify that name is in subtree)
  - +1 correct
- +2 compute and return result when name is NOT in the subtree
  - +1 attempt
    - – compare `T->name` to `someName`
    - – must attempt to verify that name is NOT in the subtree
  - +1 correct

Part B: `RootPath` (2 points)

- +1 null test and return 0 (must have attempt at else)
  - **+1/2** attempt
  - **+1/2** correct
- +1 general case
  - **+1/2** attempt
    - – "procedures" OK
    - – need 3 parameters of appropriate type
  - **+1/2** correct

Sample Student Response

Please note that only the portions of the question that contain student responses have been reproduced here. For the entire question, see pages 161-163.

Excellent Solution: 9 points

(a) Write function `PathLength`, as started below. If person `P` is in tree `T`, then `PathLength(T, P, 1)` should return the length of the longest path from the root of `T` to a node containing `P`; if person `P` does not appear in tree `T`, then `PathLength(T, P, 1)` should return 0. Note that parameter `level` can be used to keep track of the current level of the tree.

For the tree given above, the following are examples of calls to `PathLength`.

Function Call	Value Returned
`PathLength(T, "Susan", 1)`	4
`PathLength(T, "Ken", 1)`	3
`PathLength(T, "Chris", 1)`	6
`PathLength(T, "David", 1)`	0
`PathLength(T->left, "Theresa", 1)`	1
`PathLength(T->right->left, "Don", 1)`	3

In writing `PathLength`, you may call function `Max` as specified in the beginning of this question. Assume that `Max` works as specified.

Complete function `PathLength` below.

```
int PathLength(TreeNode * T, const apstring & someName, int level)
```

```
if (T==NULL)
   return 0;
else
  {
   if (T->name == SomeName)
     return max(level, max(PathLength(T->left, SomeName, level + 1),
                           PathLength(T->right, someName, level + 1)));
   else
     return max(PathLength(T->left, someName, level + 1),
                PathLength(T->right, SomeName, level + 1));
  }
```

(b) Write function RootPath, as started below. RootPath should return the length of the longest path from the root of the tree to a node containing the same name as the root; if no node other than the root contains that name, then RootPath returns 1; if the tree is empty, RootPath should return 0. For the tree given above, the following are examples of calls to RootPath.

Function Call	Value Returned
RootPath(T)	4
RootPath(T->left)	1
RootPath(T->right)	5
RootPath(T->left->left->left)	0

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

Complete function RootPath below.

```
int RootPath(TreeNode * T)

if (T= NULL)
   return 0;
else
   return PathLength(T, T→name, 1);
```

Commentary:

Student earns all 9 points with a very compact solution.

Good Solution: 6 points

(a) Write function `PathLength`, as started below. If person `P` is in tree `T`, then `PathLength(T, P, 1)` should return the length of the longest path from the root of `T` to a node containing `P`; if person `P` does not appear in tree `T`, then `PathLength(T, P, 1)` should return 0. Note that parameter `level` can be used to keep track of the current level of the tree.

For the tree given above, the following are examples of calls to `PathLength`.

Function Call	Value Returned
`PathLength(T, "Susan", 1)`	4
`PathLength(T, "Ken", 1)`	3
`PathLength(T, "Chris", 1)`	6
`PathLength(T, "David", 1)`	0
`PathLength(T->left, "Theresa", 1)`	1
`PathLength(T->right->left, "Don", 1)`	3

In writing `PathLength`, you may call function `Max` as specified in the beginning of this question. Assume that `Max` works as specified.

Complete function `PathLength` below.

```
                                // level is not used
int PathLength(TreeNode * T, const apstring & someName, int level)
{
  int i=0; j=0, k=0;
  if(T!=NULL)
    {
     i = PathLength (T->left, someName, level);
     j = Path Length (T-> right, someName, level);
     if((i==0||j==0)&&(T->name==someName)
     // if the name isn't below current node but this node contains it
        k++;  // start counting the path
     else if(i>0 || j>0)
      // if this node is part of the path
        k = Max (i,j);
      // find the longest path so far;
    }
  return k;
```

(b) Write function `RootPath`, as started below. `RootPath` should return the length of the longest path from the root of the tree to a node containing the same name as the root; if no node other than the root contains that name, then `RootPath` returns 1; if the tree is empty, `RootPath` should return 0. For the tree given above, the following are examples of calls to `RootPath`.

Function Call	Value Returned
`RootPath(T)`	4
`RootPath(T->left)`	1
`RootPath(T->right)`	5
`RootPath(T->left->left->left)`	0

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

Complete function `RootPath` below.

```
int RootPath(TreeNode * T)
{
  return (PathLength (T, T-> name, 0);
}
```

Commentary:

(a) 5 out of 7. Student earns all points except the correctness points for returning the correct result if the name is and is not in the tree.

(b) ½ out of 2. Student neglects to test the NULL case, and doesn't return the correct value in the general case.

Poor Solution: 4 points

(a) Write function `PathLength`, as started below. If person `P` is in tree `T`, then `PathLength(T, P, 1)` should return the length of the longest path from the root of `T` to a node containing `P`; if person `P` does not appear in tree `T`, then `PathLength(T, P, 1)` should return 0. Note that parameter `level` can be used to keep track of the current level of the tree.

For the tree given above, the following are examples of calls to `PathLength`.

Function Call	Value Returned
`PathLength(T, "Susan", 1)`	4
`PathLength(T, "Ken", 1)`	3
`PathLength(T, "Chris", 1)`	6
`PathLength(T, "David", 1)`	0
`PathLength(T->left, "Theresa", 1)`	1
`PathLength(T->right->left, "Don", 1)`	3

In writing `PathLength`, you may call function `Max` as specified in the beginning of this question. Assume that `Max` works as specified.

Complete function `PathLength` below.

```
int PathLength(TreeNode * T, const apstring & someName, int level)
{   if (T== NULL)return 0;

    else
      if (T-> name == someName)
            return level;

      else
       {PathLength(T-> left, someName, level ++);
        PathLength(T-> right, someName, level ++);
       }

}
```

(b) Write function RootPath, as started below. RootPath should return the length of the longest path from the root of the tree to a node containing the same name as the root; if no node other than the root contains that name, then RootPath returns 1; if the tree is empty, RootPath should return 0. For the tree given above, the following are examples of calls to RootPath.

Function Call	Value Returned
RootPath(T)	4
RootPath(T->left)	1
RootPath(T->right)	5
RootPath(T->left->left->left)	0

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

Complete function RootPath below.

```
int RootPath(TreeNode * T)
{ if (T== NULL)
      return 0;
  else
      return PathLength(T, T->name, 1);
```

Commentary:

(a) 2 out of 7. Student earns the point for handling the NULL case, as well as the left/right attempt point.
(b) 2 out of 2. Student earns all points for this part in straightforward fashion.

Chapter IV Statistical Information

- Table 4.1 — Section II Scores
- How AP Grades Are Determined
- Table 4.2 — Scoring Worksheets
- Table 4.3 — Grade Distributions
- Table 4.4 — Section I Scores and AP Grades
- College Comparability Studies
- Reminders for all Grade Report Recipients
- Reporting AP Grades
- Purpose of AP Grades

Table 4.1 — Section II Scores

The following tables show the score distributions for AP candidates on each free-response question from the 1999 Computer Science exams.

Computer Science A

	Question 1		Question 2		Question 3		Question 4	
Score	Number of Students	% At Score	Number of Students	% At Score	Number of Students	% At Score	Number of Students	% At Score
9	2,239	18.6	586	4.9	419	3.5	312	2.6
8	2,226	18.4	927	7.7	648	5.4	332	2.8
7	1,535	12.7	1,044	8.7	870	7.2	304	2.5
6	1,019	8.4	1,236	10.2	914	7.6	461	3.8
5	750	6.2	1,329	11.0	1,060	8.8	557	4.6
4	650	5.4	1,275	10.6	1,064	8.8	821	6.8
3	751	6.2	1,076	8.9	740	6.1	1,005	8.3
2	893	7.4	719	6.0	721	6.0	1,091	9.0
1	1,174	9.7	955	7.9	924	7.7	2,059	17.1
0	521	4.3	1,426	11.8	1,911	15.8	1,634	13.5
No Response	309	2.6	1,494	12.4	2,796	23.2	3,491	28.9
Total Candidates	12,067		12,067		12,067		12,067	
Mean	5.50		3.71		2.87		1.96	
Standard Deviation	3.04		2.93		2.95		2.48	
Mean as % of Maximum Score	61		41		32		22	

Computer Science AB

	Question 1		Question 2		Question 3		Question 4	
Score	Number of Students	% At Score	Number of Students	% At Score	Number of Students	% At Score	Number of Students	% At Score
9	809	12.3	43	0.7	1,037	15.7	781	11.8
8	700	10.6	565	8.6	857	13.0	557	8.5
7	574	8.7	702	10.7	971	14.7	674	10.2
6	638	9.7	768	11.7	1,004	15.2	686	10.4
5	649	9.8	655	9.9	764	11.6	564	8.6
4	622	9.4	753	11.4	562	8.5	527	8.0
3	661	10.0	693	10.5	421	6.4	494	7.5
2	587	8.9	675	10.2	272	4.1	558	8.5
1	691	10.5	571	8.7	220	3.3	458	6.9
0	350	5.3	520	7.9	239	3.6	539	8.2
No Response	310	4.7	646	9.8	244	3.7	753	11.4
Total Candidates	6,591		6,591		6,591		6,591	
Mean	4.60		3.75		5.63		4.29	
Standard Deviation	2.95		2.67		2.67		3.13	
Mean as % of Maximum Score	51		42		63		48	

How AP Grades Are Determined

Students could have received 0 to 40 points in Section I and 0 to 36 points in Section II of either of the Computer Science exams. However, these scores are not released to the student, school, or college. Instead, these raw scores are converted to grades on an AP 5-point scale, and it is these grades that are reported. This conversion involves a number of steps, which are detailed on the Scoring Worksheet on the facing page:

1. **The multiple-choice score is calculated.** To adjust for random guessing, a fraction of the number of wrong answers is subtracted from the number of right answers. This fraction is 1/4 for five-choice questions (as on the Computer Science exams), so that the expected score from random guessing will be zero.
2. **The free-response score is calculated.** When the free-response section includes two or more parts, those parts are weighted according to the value assigned to them by the Development Committee. This allows the committee to place more importance on certain skills to correspond to their emphasis in the corresponding college curriculum.
3. **A composite score is calculated.** Weighting also comes into play when looking at the multiple-choice section in comparison to the free-response section. In consultation with experts from the College Board and ETS, the AP Computer Science Development Committee decided that for the both exams, Section I should contribute 50% to the total score, and Section II, 50%. The maximum composite score was 80 for Computer Science A, and 100 for Computer Science AB. The Scoring Worksheets on the next two pages detail the process of converting section scores to composite scores for each exam.
4. **AP grades are calculated.** The Chief Faculty Consultant sets the four cut points that divide the composite scores into groups. A variety of information is available to help the CFC determine the score ranges into which the exam grades should fall:
   - Distributions of scores on each portion of the multiple-choice and free-response sections of the exam, along with totals for each section and the composite score total, are provided.
   - With these tables and special statistical tables presenting grade distributions from previous years, the CFC can compare the exam at hand to results of other years.
   - For each composite score, a roster summarizes student performance on all sections of the exam.
   - Finally, on the basis of professional judgment regarding the quality of performance represented by the achieved scores, the CFC determines the candidates' final AP grades.

   See Table 4.3 for the grade distributions for the 1999 AP Computer Science Exams.

If you're interested in more detailed information about this process, please see the "Technical Corner" of our website: www.collegeboard.org/ap. There you'll also find information about how the AP Exams are developed, how validity and reliability studies are conducted, and other nuts-and-bolts data on all AP subjects.

Table 4.2 — Scoring Worksheet — AP Computer Science A

Section I: Multiple Choice

[________ − (1/4 × ________)] × 1.000 = ________ = ________

Number correct (out of 40) — Number wrong — Multiple-Choice Score (If less than zero, enter zero.) — Weighted Section I Score

Section II: Free Response

Question 1 ________ (out of 9) × 1.1111 = ________ (Do not round)

Question 2 ________ (out of 9) × 1.1111 = ________ (Do not round)

Question 3 ________ (out of 9) × 1.1111 = ________ (Do not round)

Question 4 ________ (out of 9) × 1.1111 = ________ (Do not round)

Sum = ________

Weighted Section II Score (Do not round)

Composite Score

________ + ________ = ________

Weighted Section I Score + Weighted Section II Score = Composite Score (Round to nearest whole number.)

AP Grade Conversion Chart Computer Science A

Composite Score Range*	AP Grade
60-80	5
45-59	4
33-44	3
25-32	2
0-24	1

*The candidates' scores are weighted according to formulas determined in advance each year by the Development Committee to yield raw composite scores; the Chief Faculty Consultant is responsible for converting composite scores to the 5-point AP scale.

Table 4.2 — Scoring Worksheet — AP Computer Science AB

Section I: Multiple Choice

[________ − (1/4 × ________)] × 1.2500 = ________ = ________

Number correct (out of 40) — Number wrong — Multiple-Choice Score (If less than zero, enter zero.) — Weighted Section I Score

Section II: Free Response

Question 1	________ (out of 9)	× 1.3889 =	________ (Do not round)
Question 2	________ (out of 9)	× 1.3889 =	________ (Do not round)
Question 3	________ (out of 9)	× 1.3889 =	________ (Do not round)
Question 4	________ (out of 9)	× 1.3889 =	________ (Do not round)

Sum = ________

Weighted Section II Score (Do not round)

Composite Score

________ + ________ = ________

Weighted Section I Score + Weighted Section II Score = Composite Score (Round to nearest whole number.)

AP Grade Conversion Chart Computer Science AB

Composite Score Range*	AP Grade
70-100	5
60-69	4
41-59	3
31-40	2
0-30	1

*The candidates' scores are weighted according to formulas determined in advance each year by the Development Committee to yield raw composite scores; the Chief Faculty Consultant is responsible for converting composite scores to the 5-point AP scale.

Table 4.3 — Grade Distributions

Computer Science A

Nearly 60 percent of the AP students who took this exam earned a qualifying grade of 3 or above.

	Examination Grade	Number of Students	Percent at Grade
Extremely well qualified	5	1,921	15.9
Well qualified	4	2,899	24.0
Qualified	3	2,254	18.7
Possibly qualified	2	1,356	11.2
No recommendation	1	3,637	30.1
Total Number of Students		12,067	
Mean Grade		2.84	
Standard Deviation		1.47	

Computer Science AB

More than two thirds of the AP students who took this exam earned a qualifying grade of 3 or above.

	Examination Grade	Number of Students	Percent at Grade
Extremely well qualified	5	2,047	31.1
Well qualified	4	957	14.5
Qualified	3	1,717	26.1
Possibly qualified	2	686	10.4
No recommendation	1	1,184	18.0
Total Number of Students		6,591	
Mean Grade		3.30	
Standard Deviation		1.46	

Table 4.4 — Section I Scores and AP Grades

The following tables give the probabilities that a student would receive a particular grade ~~on~~ each of the 1999 AP Computer Science Exams given that student's score on the multiple-choice section of that exam.

Computer Science A

Multiple-Choice Score	AP Grade					Total
	1	2	3	4	5	
34 to 40	0.0%	0.0%	0.4%	16.9%	82.7%	14.6%
27 to 33	0.0%	0.7%	12.2%	70.3%	16.8%	22.6%
21 to 26	1.5%	13.7%	57.1%	27.6%	0.2%	19.8%
17 to 20	20.1%	44.3%	33.8%	1.7%	0.0%	11.6%
0 to 16	87.7%	10.3%	2.0%	0.0%	0.0%	31.4%
Total	30.1%	11.2%	18.7%	24.0%	15.9%	100.0%

Computer Science AB

Multiple-Choice Score	AP Grade					Total
	1	2	3	4	5	
30 to 40	0.0%	0.1%	3.1%	12.1%	84.7%	29.3%
27 to 29	0.0%	0.5%	22.9%	38.8%	37.8%	12.5%
20 to 26	0.8%	8.8%	62.9%	22.0%	5.5%	27.0%
15 to 19	17.8%	43.2%	37.8%	1.3%	0.0%	12.9%
0 to 14	84.7%	13.1%	2.2%	0.0%	0.0%	18.3%
Total	18.0%	10.4%	26.1%	14.5%	31.1%	100.0%

College Comparability Studies

The Advanced Placement Program has conducted college grade comparability studies in all AP subjects. These studies have compared the performance of AP Exam candidates with that of college students in related courses who have taken the AP Exam at the end of their course. In general, AP cutpoints are selected so that the lowest AP 5 is equivalent to the average A student in college, the lowest AP 4 equivalent to the average B student, and the lowest AP 3 equivalent to the average C student (see figure below).

AP Grade	Average College Grade
5	
	A
4	
	B
3	
	C
2	
	D
1	

Research studies conducted by colleges and universities and by the AP Program indicate that AP students generally receive higher grades in advanced courses than do the students who have taken the regular freshman-level courses at the institution. Summaries of several studies can be located at www.collegeboard.org/ap/techman/chap5. Each college is encouraged to undertake such studies in order to establish appropriate policy for the acceptance of AP grades.

Reminders for All Grade Report Recipients

AP Examinations are designed to provide accurate assessments of achievement. However, any examination has limitations, especially when used for purposes other than those intended. Presented here are some suggestions for teachers to aid in the use and interpretation of AP grades.

- AP Examinations in different subjects are developed and evaluated independently of each other. They are linked only by common purpose, format, and method of reporting results. Therefore, comparisons should not be made between grades on different AP Examinations. An AP grade in one subject may not have the same meaning as the same AP grade in another subject, just as national and college standards vary from one discipline to another.
- AP grades are not exactly comparable to college course grades. However, the AP Program conducts research studies every few years in each AP subject to ensure that the AP grading standards are comparable to those used in colleges with similar courses.
- The confidentiality of candidate grade reports should be recognized and maintained. All individuals who have access to AP grades should be aware of the confidential nature of the grades and agree to maintain their security. In addition, school districts and states should not release data about high school performance without the school's permission.
- AP Examinations are not designed as instruments for teacher or school evaluation. A large number of factors influence AP Exam performance in a particular course or school in any given year. As a result, differences in AP Exam performance should be carefully studied before being attributed to the teacher or school.
- Where evaluation of AP students, teachers, or courses is desired, local evaluation models should be developed. An important aspect of any evaluation model is the use of an appropriate method of comparison or frame of reference to account for yearly changes in student composition and ability, as well as local differences in resources, educational methods, and socioeconomic factors.
- The "Report to AP Teachers," sent to schools automatically when five or more students take a particular AP Exam, can be a useful diagnostic tool in reviewing course results. This report identifies areas of strength and weakness for the students in each AP course. The information may also provide teachers with guidance for course emphasis and student evaluation.

- Many factors can influence course results. AP Exam performance may be due to the degree of agreement between your course and the course defined in the relevant AP Course Description, use of different instructional methods, differences in emphasis or preparation on particular parts of the examination, differences in pre-AP curriculum, or differences in student background and preparation in comparison with the national group.

Reporting AP Grades

The results of AP Examinations are disseminated in several ways to candidates, their secondary schools, and the colleges they select.

- College and candidate grade reports contain a cumulative record of all grades earned by the candidate on AP Exams during the current or previous years. These reports are sent in early July. (School grade reports are sent shortly thereafter.)
- Group results for AP Examinations are available to AP teachers whenever five or more candidates at a school have taken a particular AP Exam. This "Report to AP Teachers" provides useful information comparing local candidate performance with that of the total group of candidates taking an exam, as well as details on different subsections of the exam.

Several other reports produced by the AP Program provide summary information on AP Examinations.

- State and National Reports show the distribution of grades obtained on each AP Exam for all candidates and for subsets of candidates broken down by sex and by ethnic group.
- The Program also produces a one-page summary of AP grade distributions for all exams in a given year.

For information on any of the above, please call AP Services at (609) 771-7300 or contact them via e-mail at apexams@ets.org.

Purpose of AP Grades

AP grades are intended to allow participating colleges and universities to award college credit, advanced placement, or both to qualified students. In general, an AP grade of 3 or higher indicates sufficient mastery of course content to allow placement in the succeeding college course, or credit for and exemption from a college course comparable to the AP course. Credit and placement policies are determined by each college or university, however, and students should be urged to contact their colleges directly to ask for specific advanced placement policies in writing.

Appendix AP Publications and 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 sort out those publications that may be of particular use to you, refer to the following key:

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

You can order many items online through the AP Aisle of the College Board Online store at http://cbweb4p.collegeboard.org/tcb/store.html/. The most current AP Order Form, which contains information about all available items, can be downloaded from the AP Library (www.collegeboard.org/ap/library). Alternatively, call AP Order Services at (609) 771-7243. American Express, MasterCard, and VISA are accepted for payment.

If you are mailing your order using the AP Order Form, send it to the Advanced Placement Program, Dept. E-05, P.O. Box 6670, Princeton, NJ 08541-6670. 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 fourth-class book rate postage (or its equivalent) on all prepaid orders; you should allow between two and three weeks for delivery. Postage will be charged on all orders requiring billing and/or requesting a faster method of shipment.

Publications may be returned within 15 days of receipt if postage is prepaid and publications are in resalable condition and still in print. Unless otherwise specified, orders will be filled with the currently available edition; prices are subject to change without notice.

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 and award options available to students, and includes the upcoming exam schedule.

Student Guides (available for Calculus, English, and U.S. History): $12 SP

These are course and exam preparation manuals designed for high school students who are thinking about or taking a specific AP course. Each guide answers questions about the AP course and exam, suggests helpful study resources and test-taking strategies, provides sample test questions with answers, and discusses how the free-response questions are scored.

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: $12 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 set of Course Descriptions is available for $100. Course Descriptions are also available for downloading free of charge from the AP Library on College Board Online.

Five-Year Set of Free-Response Questions (1995-99): $5 T

This is our no-frills publication. Each booklet contains copies of all the 1995-99 free-response questions in its subject; nothing more, nothing less. Collectively, the questions represent a comprehensive sampling of the concepts assessed on the exam in recent years and will give teachers plenty of materials to use for essay-writing or problem-solving practice during the year.

Interpreting and Using AP Grades: Free A, C, T

A booklet containing information on the development of scoring standards, the AP Reading, grade-setting procedures, and suggestions on how to interpret AP grades.

Guide to the Advanced Placement Program: Free A

Written for both administrators and AP Coordinators, this guide is divided into two sections. The first section provides general information about AP, such as how to organize an AP program at your high school, the kind of training and support that is available for AP teachers, and a look at the AP Exams and grades. The second section contains more specific details about testing procedures and policies and is intended for AP Coordinators.

Released Exams: $20
($30 for "double" subjects: Calculus, Computer Science, Latin, Physics) T

About every four years, on a staggered schedule, the AP Program releases a complete copy (multiple-choice and free-response sections) 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: $30. For each subject with a released exam, you can purchase a packet of 10 copies of that year's exams ($30) for use in your classroom (e.g., to simulate an AP Exam administration).

Secondary School Guide to the AP Program: $10 A, T

This guide is a comprehensive consideration of the AP Program. It covers topics such as: developing or expanding an AP program; gaining faculty, administration, and community support; AP grade reports, their use and interpretation; AP Scholar Awards; receiving college credit for AP; AP teacher training resources; descriptions of successful AP programs in nine schools around the country; and "Voices of Experience," a collection of ideas and tips from AP teachers and administrators.

Teacher's Guides: $12 T

Whether you're about to teach an AP course for the first time, or you've done it for years but would like to get some fresh ideas for your classroom, the Teacher's Guide can be your adviser. It contains syllabi developed by high school teachers currently teaching the AP course and college faculty who teach the equivalent course at their institution. 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. In order to help teachers and administrators who are interested in establishing an APVT at their school, the College Board has published three guides: *AP Vertical Teams in Science, Social Studies, Foreign Language, Studio Art, and Music Theory: An Introduction* ($12); *A Guide for Advanced Placement English Vertical Teams* ($10); and *Advanced Placement Program Mathematics Vertical Teams Toolkit* ($35). A discussion of the English Vertical Teams guide, and the APVT concept, is also available on a 15-minute VHS videotape ($10).

EssayPrep™ SP, T

EssayPrep is available through the AP subject pages of College Board Online (www.collegeboard.org/ap). Students can select an essay topic, type a response, and get an evaluation from an experienced reader. The service is offered for the free-response portions of the AP Biology, English Language and Composition, English Literature and Composition, and U.S. History exams. The fee is $15 per response for each evaluation. SAT II: Writing topics are also offered for a fee of $10. Multiple evaluations can be purchased at a 10-20% discount.

The College Handbook with College Explorer® CD-ROM: $25.95 SP, T, A, C

Includes brief outlines of AP placement and credit policies at two- and four-year colleges across the country. Notes number of freshmen granted placement and/or credit for AP in the prior year.

APCDs™: $49 (home version), $450 (multi-network site license) SP, T

These CD-ROMs are currently 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 for students to schedule and organize their study time.

Videoconference Tapes: $15 SP, A, C, T

Each year, AP conducts live, interactive videoconferences for various subjects, enabling AP teachers and students to talk directly with the Development Committees that design the AP Exams. Tapes of these events are available in VHS format and are approximately 90 minutes long.

AP Pathway to Success (video available in English and Spanish): $15 SP, T, A, C

This 25-minute-long video takes a look at the AP Program through the eyes of people who know AP: students, parents, teachers, and college admissions staff. They answer such questions as "Why Do It?", "Who teaches AP Courses?", and "Is AP For You?". College students discuss the advantages they gained through taking AP, such as academic self-confidence, writing skills, and course credit. AP teachers explain what the challenge of teaching AP courses means to them and their school, and admissions staff explain how they view students who have stretched themselves by taking AP Exams. There is also a discussion of the impact that an AP program has on an entire school and its community, and a look at resources available to help AP teachers, such as regional workshops, teacher conferences, and summer institutes.

What's in a Grade? (video): $15 T, C

AP Exams are composed of multiple-choice questions (scored by computer), and free-response questions that are scored by qualified professors and teachers. This video presents a behind-the-scenes look at the scoring process featuring footage shot on location at the 1992 AP Reading at Clemson University and other Reading sites. Using the AP European History Exam as a basis, the video documents the scoring process. It shows AP faculty consultants in action as they engage in scholarly debate to define precise scoring standards, then train others to recognize and apply those standards. Footage of other subjects, interviews with AP faculty consultants, and explanatory graphics round out the video.

AP® Computer Science

1998-99 Development Committee

Susan Rodger
Duke University
Durham, North Carolina, *Chair*

Alyce Brady
Kalamazoo College, Michigan

Cathy Key
University of Texas
San Antonio

Joseph Kmoch
Washington High School
Milwaukee, Wisconsin

Kathleen Larson
Kingston High School
New York

Mark Weiss
Florida International University
Miami

Chief Faculty Consultant: Mark Stehlik
Carnegie Mellon University, Pittsburgh, Pennsylvania

Chief Faculty Consultant Designate: Chris Nevison
Colgate University, Hamilton, New York

ETS Consultants: Fran Hunt, Esther Tesar

College Board Consultant: Gail Chapman

18060-004830 • S60M20 • Printed in U.S.A.

I.N. 255167