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 cach 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

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

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.

GO ON TO THE NEXT PAGE

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

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

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.

GO ON TO THE NEXT PAGE

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.

GO ON TO THE NEXT PAGE

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.

GO ON TO THE NEXT PAGE

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.

GO ON TO THE NEXT PAGE

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.

GO ON TO THE NEXT PAGE

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

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 ≤ 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

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

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

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

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 );
```

```
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.

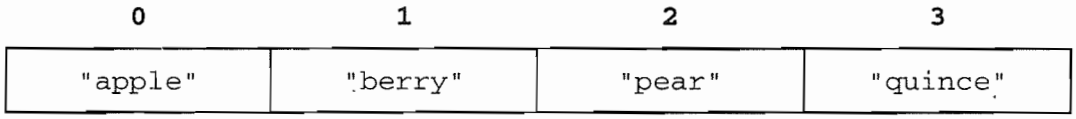
```
void FillSeniorList(const apvector<StudentInfo> & roster,
                    int numStudents, apvector<StudentInfo> & seniors,
                    int & numSeniors)
// precondition: roster contains numStudents records,
//               0 < numStudents ≤ roster.length(),
//               and seniors 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:



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	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
```

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