

# Workbook to Accompany



An Introduction to Programming  
and Computer Science

Maria Litvin  
Phillips Academy, Andover, Massachusetts

Gary Litvin  
Skylight Software, Inc.

Skylight Publishing  
Andover, Massachusetts

Skylight Publishing  
14 Lincoln Street  
Andover, MA 01810

web: <http://www.skylit.com>  
e-mail: [sales@skylit.com](mailto:sales@skylit.com)  
[support@skylit.com](mailto:support@skylit.com)

ISBN 978-0-9654853-8-8

**Copyright © 1998 by Maria Litvin, Gary Litvin, and  
Skylight Publishing**

This book is licensed under the Creative Commons  
Attribution-NonCommercial-ShareAlike 4.0 International License



**You are free to:**

Share — copy and redistribute the material in any medium or format

Adapt — remix, transform, and build upon the material

The licensor cannot revoke these freedoms as long as you follow the license terms.

**Under the following terms:**

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

NonCommercial — You may not use the material for commercial purposes.

ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

## Contents


Chapter 1.	Introduction to Hardware and Software .....	5
Chapter 2.	A First Look at a C++ Program.....	11
Chapter 3.	Variables and Constants .....	17
Chapter 4.	Arithmetic Expressions .....	19
Chapter 5.	Arrays, apvector and apmatrix Classes .....	23
	Chapters 1-5 Review.....	25
Chapter 6.	Logical Expressions and if-else Statements .....	29
Chapter 7.	Iterative Statements: while, for, do-while .....	33
Chapter 8.	The switch Statement .....	39
	Chapters 6-8 Review.....	41
Chapter 9.	Algorithms.....	43
Chapter 10.	Monte Carlo Methods.....	47
Chapter 11.	Pointers, References, Dynamic Memory Allocation.....	49
Chapter 12.	Strings.....	53
	Chapters 9-12 Review.....	57
Chapter 13.	Structures.....	61
Chapter 14.	Modularity .....	65
Chapter 15.	Classes .....	69
Chapter 16.	Templates .....	75
	Chapters 13-16 Review.....	79
Chapter 17.	Linked Lists.....	83
Chapter 18.	Stacks.....	87
Chapter 19.	Recursion.....	91
Chapter 20.	Queues.....	97
	Chapters 17-20 Review.....	99
Chapter 21.	Classes: More Advanced Features .....	103
Chapter 22.	Trees .....	105
Chapter 23.	Expression Trees .....	109
Chapter 24.	Heaps .....	111
Chapter 25.	Analysis of Algorithms .....	113
	Chapters 22-25 Review.....	115
Chapter 26.	Searching and Hashing.....	117
Chapter 27.	Sorting.....	121
Chapter 28.	Inheritance .....	123
◆◆	Projects .....	125
	Answers and Solutions .....	133

## How to Use this Workbook

Chapter numbers correspond to the chapters in *C++ for You++*.

An underlined question number indicates that the answer or solution is provided at the end of the workbook.

Multiple-choice questions are marked by an answer box:

The  symbol marks questions that assume implementation on a computer.

The ■ symbol marks questions that require more insight or more detailed work (intermediate).

The ◆ symbol marks questions that are tricky or require a lot of work (advanced).

All projects in the “◆◆ Projects” section are challenge projects that assume computer implementation.

## Chapter 1. Introduction to Hardware and Software

**Are the following entities or devices part of a computer system's hardware (H) or software (S)?**

1. Operating system \_\_\_\_\_
2. CPU \_\_\_\_\_
3. Compiler \_\_\_\_\_
4. Program editor \_\_\_\_\_
5. IDE \_\_\_\_\_
6. GUI (Graphical User Interface) \_\_\_\_\_
7. Modem \_\_\_\_\_
8. Assembler \_\_\_\_\_
9. Bus \_\_\_\_\_
10. RAM \_\_\_\_\_
11. ROM \_\_\_\_\_
12. ■ File \_\_\_\_\_

**13.** What is the maximum number of different codes or numbers that can be represented in 3 bits? \_\_\_\_\_ 8 bits? \_\_\_\_\_

**14.** An experiment consists of tossing a coin 10 times and its outcome is a sequence of heads and tails. How many possible outcomes are there? \_\_\_\_\_

**15.** A 12-bit A/D converter digitizes amplitudes of an analog signal into 12-bit numbers. How many different digitized amplitude values are possible?  
\_\_\_\_\_

**16.** 16-bit addresses can directly address 64KB of memory. How much memory (in MB) is directly addressable with 32-bit addresses? \_\_\_\_\_

**17. ■** If a GIGTEL microprocessor can directly address 1024 GB of memory, what should the width of the address bus be? \_\_\_\_\_

Assume that these 8-bit binary numbers represent unsigned integers in the usual way, with the least significant bit on the right. Write the decimal value and the hex representation of these binary numbers. Example:

	binary	decimal	hex
	00011100	28	1C
18.	00000011	_____	_____
19.	00001111	_____	_____
20.	00101111	_____	_____
21.	10000001	_____	_____
22.	11000001	_____	_____
23.	00001011	_____	_____
24.	11110101	_____	_____

Assume that these 16-bit binary numbers represent unsigned integers in the usual way, with the least significant bit on the right. Write the decimal value and the hex representation of these binary numbers.

Example:

	binary	decimal	hex
	00000011 00001111	$3 \cdot 256 + 15 = 783$	030F
25.▪	00000001 11000011	_____	_____
26.▪	00001111 00001011	_____	_____
27.▪	10000000 00000000	_____	_____
28.▪	11111111 00000000	_____	_____
29.▪	10000001 10000001	_____	_____

- 30.** (a) Using the ASCII table, write the ASCII codes for the following characters in decimal, hex and binary:

	decimal	hex	binary
'A'	_____	_____	_____
'a'	_____	_____	_____
'Q'	_____	_____	_____
'q'	_____	_____	_____

- (b) Note that the binary representations of the ASCII codes for the lower and upper case of the same letter differ only in one bit. Write the binary and hex representations of a byte where that bit is set to 1 and all other bits are set to 0:

binary	hex
_____	_____

**Solve for x, where x is a (decimal) number. A character in single quotes represents the ASCII code for that character:**

- 31.** '5' - x = '0'                      x = \_\_\_\_\_  
**32.** 'G' + x = 'g'                      x = \_\_\_\_\_  
**33.** x + ']' = '['                      x = \_\_\_\_\_

- 34.** Interpret the following hex dump of an ASCII data file used later in one of the labs in the book:

```

45 4E 47 4C 49 53 48 2D-49 54 41 4C 49 41 4E 20 _____
44 49 43 54 49 4F 4E 41-52 59 0D 0A 77 61 6E 74 _____
20 20 20 20 76 6F 6C 65-72 65 0D 0A 61 20 20 20 _____
20 20 20 20 75 6E 0D 0A-49 20 20 20 20 20 20 _____
69 6F 0D 0A 70 72 6F 67-72 61 6D 20 70 72 6F 67 _____
72 61 6D 6D 61 0D 0A 74-68 69 73 20 20 20 20 71 _____
75 65 73 74 6F 0D 0A 74-6F 64 61 79 20 20 20 6F _____
67 67 69 0D 0A 6C 6F 76-65 20 20 20 20 61 6D 61 _____
72 65 0D 0A _____

```

**Name the software development tool that transforms:**

- 35.** Source code into an object module                      \_\_\_\_\_  
**36.** Object module(s) into an executable program                      \_\_\_\_\_

- 37. ■** Come up with a method for representing the state of a tic-tac-toe board in computer memory. Can you fit your representation into three bytes?

- 38.▪** How much memory does it take to hold a 512 by 512 gray-scale image with 16 levels of gray?
- 39.♦** When a printer runs out of paper, the eight-bit printer status register of the parallel interface adapter gets the following settings: bit 7 (leftmost bit), “BUSY,” is set to 1; bit 5, “PE” (“paper end”), is set to 1; and bit 3, “ERROR,” is set to 0. Bit 4 is always 1 when a printer is connected, bit 6 is 0, and bits 0-2 are not used. Write the hex value for the bit mask that has bits 0-2 set to 0 and bits 3-7 set to 1. This mask should “cut out” the important bits from a byte by applying the “and” instruction. Write the hex value equal to the setting of the printer status register when the printer runs out of paper.
- Mask: \_\_\_\_\_
- Status register: \_\_\_\_\_
- 40.♦** Serial interface is a standard hardware interface used for connecting input or output devices—such as modems, mice, digitizing tablets, or printers—to a computer. A serial interface adapter has an eight-bit line control register that specifies communication parameters. Bits 0 and 1 (the least significant bits) specify the number of data bits:

Bit 0	Bit 1	Data bits
0	0	5
0	1	6
1	0	7
1	1	8

Bit 2 specifies the number of stop bits: 0 means 1 stop bit and 1 means 2 stop bits. Bit 3, when set, enables parity checking, and bit 4 selects odd parity (0) or even parity (1). Bits 5-7 are set to 0 (not used). Write the hex value that should be programmed into the line control register for:

- (a) 7 data bits, one stop bit, and even parity \_\_\_\_\_
- (b) 8 data bits, one stop bit, and no parity \_\_\_\_\_

- 41.** ♦ The table below is called a *Greco-Roman square*: each of the three Latin letters occurs exactly once in each row and each column; the same is true for each of the three Greek letters; and each Latin-Greek combination occurs exactly once in the table:

A $\gamma$	B $\alpha$	C $\beta$
B $\beta$	C $\gamma$	A $\alpha$
C $\alpha$	A $\beta$	B $\gamma$

Substitute the digits 0, 1 and 2 for A, B, C and for  $\alpha$ ,  $\beta$ ,  $\gamma$  (in any order). Convert the resulting base-3 numbers into decimal (base-10) numbers. The base-3 system uses only three digits: 0, 1, and 2. The numbers are represented as follows:

Base 3	Decimal
0	0
1	1
2	2
10	3
11	4
12	5
20	6
21	7
22	8
100	9
...	...

Add 1 to each number. You will get a table in which the numbers 1 through 9 are arranged in such a way that the sum of the numbers in each row and column is the same. Explain why you get this result and find a way to substitute the digits 0, 1, and 2 for letters so that the sum of numbers in each of the two diagonals is the same as in the rows and columns. What you get then is called a *magic square*. Using a similar method, build a 5 by 5 magic square.

- 42.♦** In the Nim game, stones are arranged in piles of arbitrary size. Each player in turn takes a few stones from any one pile. Every player must take at least one stone on every turn. The player who takes the last stone wins.

Games of this type often have a winning strategy. This strategy can be established by tagging all possible positions in the game with two tags, “plus” and “minus,” in such a way that any move from a “plus” position always leads to a “minus” position, and from any “minus” position there is always a possible move into some “plus” position. The final winning position must be tagged “plus.” Therefore, if the first player begins in a “minus” position, she can win by moving right away into a “plus” position and returning to a “plus” position on each subsequent move. If, however, the first player begins in a “plus” position, then the second player can win, provided he knows how to play correctly.

In the Nim game, we can convert the number of stones in each pile into a binary number and write these binary numbers in one column (so that the units digits are aligned on the right). We can tag the position “plus” if the number of 1’s in each column is even and “minus” if the count of 1’s in at least one column is odd. Prove that this method of tagging “plus” and “minus” positions defines a winning strategy. Who wins starting with four piles of 1, 3, 5, and 7 stones—the first or the second player? What’s the correct response if the first player takes five stones from the pile of 7?

## Chapter 2. A First Look at a C++ Program

**Questions 1-11 refer to the following program:**

```
#include <iostream.h>
#include "apstring.h"

int main()
{
    apstring firstName;

    cout << "Please enter your first name: ";
    cin >> firstName;
    cout << firstName
         << ', '
         << ', '
         << "congratulations on your first program!"
         << endl;
    return 0;
}
```

1.  Enter, compile and run the program.

**Mark *true* or *false*:**

2. The program has no functions. \_\_\_\_\_
3. `apstring` is not a reserved word. \_\_\_\_\_
4. The program prompts the user to enter his or her name. \_\_\_\_\_
5. `firstName` is a C++ reserved word. \_\_\_\_\_
6. `#include <iostream.h>` is a preprocessor directive. \_\_\_\_\_
7. Each item displayed with the `<<` operator must be placed on a separate line.

8.  \_\_\_\_\_  
`cout` is a reserved word. \_\_\_\_\_

9.  Mark all C++ reserved words in the program.

10.  Find an example in the Dictionary program where `\n` is used instead of `endl`. Rewrite the following statements using only two `<<` operators:

```
cout << firstName
     << ', '
     << ', '
     << "congratulations on your first program!"
     << endl;
```

11. <sup>■</sup> Modify the program so that it supports the following dialog with the user (user input is shown in bold type):

```
Please enter your first name: Sheila
Please enter your last name: Fox
Sheila Fox, congratulations on your second program!
```

Identify the following statements as referring to required C++ syntax or optional C++ style:

12. A program begins with two slashes (`//`). \_\_\_\_\_
13. The names of all functions begin with a capital letter. \_\_\_\_\_
14. Each opening brace has a matching closing brace. \_\_\_\_\_
15. All statements within a pair of matching braces are indented by 4 characters. \_\_\_\_\_
16. A closing brace is placed on a separate line. \_\_\_\_\_
17. A program has a blank line before each function definition. \_\_\_\_\_
18. One of the functions in the program is called “main.” \_\_\_\_\_
19. The word `IF` is not used as a name for a variable. \_\_\_\_\_

Mark *true* or *false*:

20. `CHAR` is not a reserved word. \_\_\_\_\_
21. Each C++ statement must be written on a separate line. \_\_\_\_\_
22. Each preprocessor directive must be written on a separate line. \_\_\_\_\_
23. C++ functions can be called only from `main()`. \_\_\_\_\_
24. Each function must have a return statement. \_\_\_\_\_
25. The compiler automatically corrects syntax errors. \_\_\_\_\_
26. <sup>■</sup> Programming languages usually have less redundancy than natural languages. \_\_\_\_\_

Complete each sentence with the word *always*, *sometimes*, or *never*:

27. Function prototypes are \_\_\_\_\_ placed near the top of the program.
28. Function names \_\_\_\_\_ begin with the `#` symbol.
29. <sup>■</sup> There is \_\_\_\_\_ a semicolon after a closing brace.
30. <sup>■</sup> Program input \_\_\_\_\_ comes from `cin`.
31. <sup>■</sup> Standard library functions' prototypes are \_\_\_\_\_ provided in system header files.

**Choose the right word to fill the blank:**

- 32.** `#ifdef-#else-#endif` directives are used for conditional \_\_\_\_\_ (compilation/execution) of a program fragment.
- 33.** The `dict.dat` file is \_\_\_\_\_ (compiled with/linked with/read by) the Dictionary program.
- 34.** The format of the `dict.dat` file in the Dictionary program must be compatible with \_\_\_\_\_ (C++ syntax/the operating system/the `LoadDictionary(...)` function).

**Questions 35-39 refer to the following program:**

```
#include <iostream.h>

double Average (double x, double y);

int main()
{
    double a, b;

    cout << "Enter two numbers: ";
    cin >> a >> b;
    cout << "The average of " << a << " and " << b << " is "
         << Average(a,b) << endl;
    return 0;
}

double Average (double x, double y)
{
    return (x + y) / 2.;
}
```

- 35.** How many functions does this program have? \_\_\_\_\_
- 36.** Enter, compile and run the program.
- 37.** Comment out the line that contains the function prototype. Compile the program and observe the compiler message.
- 38.** Delete the prototype and move the function definition above `main` to make the program work.
- 39.** Modify the program so that it prompts the user for three numbers and displays their average.

Circle all names below that can be used as valid names in a C++ program without generating a syntax error (even though they may be stylistically awful):

- 40. 7seas
- 41.   \_get\_3
- 42. cin.get
- 43.   Cout
- 44. long^
- 45.   INT
- 46. cout\_
- 47.   \_0

Mark *true* or *false*:

- 48. cin is called the “stream extraction operator.” \_\_\_\_\_
- 49. In the statement:

```
ifstream inFile("DICT.DAT");
```

ifstream is the standard name of the file output class and inFile is a name chosen by the programmer. \_\_\_\_\_

- 50. The #include <iostream.h> directive must be included in any program that uses cout or cin. \_\_\_\_\_
- 51.▪ Every program sends its output to cout. \_\_\_\_\_

- 52.▪ What is the output of the following code?

```
cout << "Fun"  
      << "day\n";
```

- A) Syntax error
- B) Fun day\n
- C) Fun  
day
- D) Funday
- E) Depends

53. ■ Answer Question 52 above for the following code:

```
cout << "Fun" << "day" << "\n";
```

54. ■ Answer Question 52 for the following code:

```
cout << Fun
      << day << "\n";
```

and

```
cout << Fun
      << day << \n;
```

55. ■ Write statements that produce the following dialog with the user (the user's input is in bold):

```
No errors.
Enter another file name ==> TEST.DAT
Loading TEST.DAT. Please wait...
```



56. ♦ Without going too deeply into the meaning of the following code, try to find one syntax error (a mistake that would be caught by the compiler) and one bug (a logical mistake not detected by the compiler):

```
int Minimum(int a[], int n)

// Returns the position of the smallest element in the array "a".
// "n" is the number of elements in the array.

{
    int i, iMin = 0;    // iMin is the position of the
                       // smallest element.

    for (i = 1; i < n; i++) {
        if (a[i] < a[iMin])
            iMin = i
    }
    return i;
}
```

57.◆<sup>□</sup> The following program converts temperatures from Celsius to Fahrenheit:

```
int main()
{
    double degC, degF;

    cout << "Please enter degrees Celsius: ";
    cin >> degC;
    degF = 32. + 9./5. * degC;
    cout << degC << "C is " << degF << "F\n";

    return 0;
}
```

Restructure the program so that the main program prompts the user and displays the result but the actual conversion is performed in a separate function:

```
double CelToFahr(double x);
```

## Chapter 3. Variables and Constants

1. Assuming that the `unsigned short` data type is implemented as a two-byte binary value, what is the largest possible value that an `unsigned short` variable can hold? Give your answer as a decimal integer.
2. What is the output of the following program?

```
#include <iostream.h>

int main()
{
    double amt = 100.00;
    cout << sizeof(amt) - sizeof(double) << endl;

    return 0;
}
```

### Which of the following lines are syntactically valid declarations?

3. `short years, long hours;`
  4. `unsigned, short positive;`
  5. `double short;`
  6. `double dollars_and_cents;`
  7. `char mi; int age;`
- 
8. <sup>□</sup> Determine how many bytes a `long double` variable occupies on your system.
  9. If character constants in single quotes are represented using ASCII code, what is the decimal value of `'1'+'1'`?

### Which of the following lines are syntactically valid declarations?

10. `int count = 10; count2 = count; neg_count = -count;`
11. `double x = 0, y = 1, r = x*x + y*y;`
12. `char A = 'A', C = 67, H = 72;`
13. `char CR = \n;`
14. `char exclamation_point = "!";`
15. `const double pi = 3.14159; double r=3., area = pi*r*r/2.;`

Mark *true* or *false*:

16. The use of global variables is a sign of good program design. \_\_\_\_\_
17. If a local and a global variable have the same name, the compiler reports a syntax error. \_\_\_\_\_
18. The `typedef` statement is used to rename variables. \_\_\_\_\_
19. The scope of a variable is the largest range of its values. \_\_\_\_\_
- 20.▣ Local variables in different functions may have the same name. \_\_\_\_\_
- 21.▣ Local variables in a function may have the same names as the function arguments in its definition. \_\_\_\_\_

22. In the following declaration:

```
enum COIN {penny=1, nickel, dime, quarter, half_dollar};
```

what is the value of `quarter`? \_\_\_\_\_

- 23.▣ If you take any two positive integers  $m$  and  $n$  ( $m > n$ ), then the numbers

$$a = m^2 - n^2; \quad b = 2mn; \quad c = m^2 + n^2$$

form a Pythagorean triple:

$$a^2 + b^2 = c^2$$

You can use algebra to prove that this is always true.

Write a program that prompts the user for two positive integers,  $m$  and  $n$ , and prints out the corresponding Pythagorean triple  $(a, b, c)$ .

- 24.♦ Find a bug (a logical error or an error in usage not caught by the compiler) in the following code fragment:

```
double a, b;  
int temp;  
  
cout << "Enter two real numbers: ";  
cin >> a >> b;  
...  
// Swap the numbers:  
temp = a;  
a = b;  
b = temp;  
...
```

## Chapter 4. Arithmetic Expressions

### What is the output of the following statements?

1. `cout << 5. / 10 << endl;` \_\_\_\_\_
2. `cout << 5 / 10 << endl;` \_\_\_\_\_
3. `cout << 1 / 2 * 10 << endl;` \_\_\_\_\_
4. `cout << 1. / 2 * 10 << endl;` \_\_\_\_\_
5. `cout << 1 / 2. * 10 << endl;` \_\_\_\_\_
6. `cout << 5 % 10 << endl;` \_\_\_\_\_
7. `cout << -5 % 10 << endl;` \_\_\_\_\_

### Fix the bugs in the following statements:

- 8.
- ```
const short days = 365, hours = 24, mins = 60, secs = 60;
cout << "Seconds in a year = "
    << hours * mins * secs * double(days) << endl;
```

- 9.
- ```
const double g = 16.;
double t;
cout << " Enter time in secs: ";
cin >> t;
cout << "The travel distance is " << 1 / 2 * (g * t * t)
    << endl;
```

- 10.
- ```
short x, y;

cout << "Enter two integers x and y: ";
cin >> x >> y;
x *= double(x);
y *= double(y);
double sq_radius = x + y;
if (sq_radius > 250000.)
    cout << "(x,y) is outside the circle.\n";
```

11. ■ Write a function:

```
double BodyMassIndex(int inches, int lbs);
```

that takes a person's height in inches and weight in pounds as arguments and returns the body mass index (BMI). BMI is defined as the weight, expressed in kilograms, divided by the square of the height expressed in meters. One inch is 0.0254 meters and one pound is 0.454 kilograms. Write a program that prompts the user for his weight and height, calls `BodyMassIndex (...)`, and displays the BMI.

12. ■ Write a function

```
int DogsHumanAge(int years);
```

that converts a dog's age to the corresponding human age. Assume that a dog's first year corresponds to a human age of 13. After that every three years in a dog's life correspond to sixteen years in human life. The function returns the corresponding human age, truncated to the nearest integer. Write a program that prompts the user for the age of his dog in years (a positive integer), calls the `DogsHumanAge (...)` function and displays the result.

13. What is the output of the following statements?

```
int c = 3;
cout << c++;
cout << c;
cout << ++c;
cout << c << endl;
```

---

Mark *true* or *false*:

14. The result of the statement

```
int c = a + b++;
```

is always the same as the result of

```
int c = a + b + 1;
```

---

15. ■ After executing the statements

```
w /= 2;
w *= 2;
```

the value of the `int` variable `w` always remains unchanged. \_\_\_\_\_

16. ■ If the value of `x` is negative, the value of `x%3` can be negative. \_\_\_\_\_  
 17. ■ The value of `(a+1)%7` is always the same as the value of `a%7+1`. \_\_\_\_\_

18. ■ If `double x` has a positive value, come up with an expression that rounds `x` to the nearest integer.  
 \_\_\_\_\_


19. ■ Write a code fragment that prompts the user to enter a positive two-digit integer, reads a number into an `int` variable, computes a new number in which the two digits are reversed, and displays the result.

20. ■ Display the value of the `INT_MAX` constant defined in the `limits.h` header file provided with your C++ compiler.

21. Fix the cast operator in the following statements so that it works for all integer values of `n`:

```
int n;
long n2;
...
n2 = long(n*n);    // set n2 to n squared
```

22. ■ Write a program that supports the following dialog with the user (the user's input is shown in bold):

|                                                                                                                                                                                                                   |                                                                                       |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| <pre>Enter deposit amount (dollars) ==&gt; <b>900</b> Enter annual interest rate (%) ==&gt; <b>6</b>  Balance after 1 year:      954.00 Balance after 2 years:    1011.24 Balance after 3 years:    1071.91</pre> |  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|

Use a variable of an integral type to hold the initial balance entered by the user; allow amounts between 1 and \$1,000,000. Use the simple interest formula: the balance at the end of the year is equal to the balance at the beginning of the year times  $(1 + \text{rate})$ , where the rate is expressed as a decimal.

23. ■ Write a program that prompts the user for his travel distance, the car's gas mileage (mpg) and the price of gas, and displays the estimated cost of gas for the trip.
24. ■ Fill in the blanks in the following program that converts a number of minutes into hours:minutes.

```
#include <iostream.h>
#include <iomanip.h>

int main()
{
    int mins;

    cout << "Enter the number of minutes: ";
    cin >> mins;

    cout << _____ << ':'
         << setfill('0') // set output fill character to '0'

         << setw(2) << _____ << endl
         << setfill(' '); // restore the default fill char

    return 0;
}
```

## Chapter 5. Arrays, `apvector` and `apmatrix` Classes

Mark *true* or *false*:

1. The elements of the array  
`char a[5];`  
are properly referred to as  
`a[1], a[2], a[3], a[4]` and `a[5]`. \_\_\_\_\_
2. The following array has 101 elements:  
`int x[100];` \_\_\_\_\_
3. The following array occupies 120 consecutive bytes in memory:  
`short codes[60];` \_\_\_\_\_
4. `apvector` is a reserved word. \_\_\_\_\_
5. The `apvector` class reports an error when a subscript value is out of bounds.  
\_\_\_\_\_

Which of the following are valid declarations:

6. `apvector x(100);` \_\_\_\_\_
7. `apvector<int> count;` \_\_\_\_\_
8. `apvector<double> coord[100];` \_\_\_\_\_
9. `apvector<int> odd(5) = {1, 2, 3, 5, 7};` \_\_\_\_\_
10. `apvector<bool> flags(5, false);` \_\_\_\_\_
11. Declare an `apvector` `q` of 10 chars, all set to `'?'`.
12. Declare an `apmatrix` of `int` with 10 rows and 4 columns called `chart`.

13. ■ Enter the following declarations and assignments into a test program, compile, and find out which of them cause problems:

```
#include "apvector.h"
...
apvector<int> a;           _____
apvector<int> b(100);     _____
apvector<double> x(100); _____
a = b;                   _____
x = b;                   _____
x = double(b);          _____
```

Try to explain why.

14. ■ The following statements should double the values of the elements of the array *scores*. Find the bug.

```
int i;
apvector<int> scores(16);
...
i = 0;
// Repeat as long as i does not exceed 16:
while (i <= 16) {
    scores[i] *= 2;
    i++;
}
```

## Chapters 1-5 Review

Complete each sentence with the word *always*, *sometimes*, or *never*:

1. C++ compilers are \_\_\_\_\_ implemented in hardware.
  2. Data files \_\_\_\_\_ use ASCII code.
  3. The computer screen \_\_\_\_\_ shows characters typed on the keyboard.
  4. Output devices are \_\_\_\_\_ stream devices.
  5. The 8-bit binary number 11111101 \_\_\_\_\_ represents -3.
- 
6. If Byte-a-Bit's pizza can be ordered medium or large with any combination of five toppings, can the designer of Byte-a-Bit's POS system fit a pizza description into one byte?

Mark *true* or *false*:

7. Each C++ program begins with a comment. \_\_\_\_\_
8. Each C++ program has at least one function. \_\_\_\_\_
9. Each function in the source code must also have a prototype near the top of the program. \_\_\_\_\_
10. `prompt` is a C++ reserved word. \_\_\_\_\_
11. User data is often read into the program by using the stream extraction operator. \_\_\_\_\_

Which of the following lines are syntactically valid declarations of variables, constants, or arrays?

12. `double xout = .99;`
13. `const int hundred = "100";`
14. `char alarm = \a;`
15. `const double int secs_in_day = 86400;`
16. `apvector<int> price(3) = 69, 79, 99;`
17. `double c-out = 2;`
18. `unsigned double freq = 166.;`
19. `enum PRIMCOLOR {Red = 1, Green, Blue = 4};`

- 20.** What is the output of the following statements if the user enters 3?

```
int dist;
cout << "Enter travel distance in miles: ";
cin >> dist;
dist *= 1.6;
cout << dist << " mi = " << double(dist) << " km\n";
```

**What is printed by the following statements?**

- 21.**

```
const int boilingPoint = 100;
cout << "Boiling point Fahrenheit = "
    << int(9. / 5 * boilingPoint + .5) + 32 << endl;
```

---

- 22.**

```
int c = 3;
c += 4; c /= 4; c -= 1; c *= 4;
cout << "c = " << c << endl;
```

---

- 23.■** The following code may fail when the variable `hours` gets a large value:

```
...
int hours;
cin >> hours;
cout << hours << " hours is " << 3600 * hours << " seconds\n";
```

Change the declaration of `hours` into a larger integral type and declare and use a symbolic constant of the appropriate integral type instead of 3600.

- 24.** Write a function

```
double Distance(int x1, int y1, int x2, int y2);
```

that returns the distance between the points  $(x_1, y_1)$  and  $(x_2, y_2)$ . The distance formula is

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

and the function should work for all integer values of coordinates. Use the library function `double sqrt(double s)`, which is declared in `math.h`.

- 25.** Find the index (subscript) of the middle element in an array:

```
apvector<char> test(17);
```

\_\_\_\_\_

- 26.** Fill in the blanks in the following function:

```
double AddEveryOther (const apvector<double> &a)
// Returns a[1] + a[3] + ...
// (the sum of all the elements in odd places).
{
    int i = 1, n = _____;
    double sum = 0.;

    // Repeat statements within braces as long as i
    // remains less than n:
    while (i < n) {
        _____ += _____;
        _____ += _____;
    }
    return _____;
}
```



## Chapter 6. Logical Expressions and if-else Statements

1. Write a function

```
double Max(double x, double y);
```

that returns the larger of the values  $x$  and  $y$  (or either, if they are equal).

**Fill in the blanks in the following functions:**

- 2.

```
bool isdigit(char d)
// Returns true if d is a digit (in ASCII code), false otherwise.
{
    return _____
}
```

- 3.

```
bool isalpha(char c)
// Returns true if c is a letter (in ASCII code),
// false otherwise.
{
    return _____
}
```

4. Find and fix the bug in the following code:

```
int mins;
bool halfHour;
...
// Set halfHour to true when mins is 30 mins after the hour:
halfHour = (mins % 30 == 0);
if (halfHour)
    cout << '\a';
```

**Simplify:****5.**

```
if (!(x == 7) && !(x > 7))  
    ...
```

---

**6.**

```
bool inside = !((x < left) || (x > right) || (y < top)  
              || y > bottom);
```

---

**7.**

```
bool no = (ch[0] == 'N' && ch[1] == 'O') ||  
          (ch[0] == 'n' && ch[1] == 'o') ||  
          (ch[0] == 'N' && ch[1] == 'o') ||  
          (ch[0] == 'n' && ch[1] == 'O');
```

---

**8.** Simplify the if statement in the following code:

```
enum COLOR {WHITE, BLACK};  
const int ROWS=480, COLS=640;  
  
apmatrix<COLOR> pixels(ROWS, COLS);  
...  
    int row, col;  
    ...  
    if (pixels[row][col] == WHITE && pixels[row][col+1] == BLACK ||  
        pixels[row][col] == BLACK && pixels[row][col+1] == WHITE)  
        count++;
```

**9.** Remove as many parentheses as possible without changing the meaning of the conditional expression in the if statement:

```
if (((x + 2) > a) || ((x - 2) < b)) && (y >= 0)  
    ...
```

---

10. ■ Fill in the blanks in the following function:

```
int HexToDec(char hexdigit)

// Returns the decimal value of the corresponding hexadecimal
// digit, where "hexdigit" is a character '0'-'9' or 'A'-'F'
// (in ASCII code).
// Returns -1, if "hexdigit" is not a valid hex digit.
// Examples: HexToDec('3') returns 3;
//           HexToDec('B') returns 11;
//           HexToDec('X') returns -1;

{
    int x = -1;

    if ( _____ )
        _____;

    else if ( _____ )
        _____;

    return x;
}
```

11. ◆ Priority mail costs \$3.00 for the first two pounds and \$1.00 for each additional pound or fraction. First class mail costs 32 cents for the first ounce and 23 cents for each additional ounce or fraction, up to 11 ounces, after which priority rates apply. Fill in the blanks in the following function:

```
enum MAIL {FIRSTCLASS, PRIORITY};

double MailMeter(double ounces, MAIL type)

// Returns the required postage for a letter or package.
// "ounces" is the weight in ounces.
// "type" is the type of service: FIRSTCLASS or PRIORITY.

{
    ...
}
```

12. Using the rules of short-circuit evaluation, correct the following statement to prevent accessing elements of the array with subscripts out of bounds:

```
const int ROWS = 32, COLS = 32;
apmatrix<char> grid(ROWS, COLS);
...
if (grid[row-1][col+1] == 'x' && row >= 1 && col < COLS-1)
    ...
```

## 13. ■ Write a function

```
bool Later(int month1, int day1, int year1, int month2,
           int day2, int year2);
```

that returns `true` if the second date is later than the first and `false` otherwise.

14. ■ Circle all the expressions below that are equivalent (i.e. have the same value for all values of the boolean variables `a` and `b`) to `!(a && b)`:

- (a) `!a && !b`
- (b) `!a && b`
- (c) `a || b`
- (d) `!a || !b`

## 15. ■ U.S. taxpayers use federal income tax schedules like the one below to calculate their income tax:

**Schedule X** —Use if your filing status is **Single**

| If the amount on Form 1040, line 37, is:<br><i>Over—</i> | <i>But not over—</i> | Enter on Form 1040, line 38 | <i>of the amount over—</i> |
|----------------------------------------------------------|----------------------|-----------------------------|----------------------------|
| \$0                                                      | \$23,350             | .....15%                    | \$0                        |
| 23,350                                                   | 56,550               | \$3,502.50 + 28%            | 23,350                     |
| 56,550                                                   | 117,950              | 12,798.50 + 31%             | 56,550                     |
| 117,950                                                  | 256,500              | 31,832.50 + 36%             | 117,950                    |
| 256,500                                                  |                      | 81,710.50 + 39.6%           | 256,500                    |

Taxpayers who are married and file taxes jointly use a similar schedule, Schedule Y-1, where the rates are the same but the income cutoffs are \$0; 39,000; 94,250; 143,600; and 256,500 respectively. Since not all taxpayers have a degree in accounting, many turn to paid preparers or computer programs.

Write the code for the following function:

```
enum FILING {SINGLE, MARRIED};

long IncomeTax(long taxableIncome, FILING status)

// Returns the income tax from schedules,
// rounded to the nearest dollar, given taxable income
// in dollars and filing status.

{
    ...
}
```

## Chapter 7. Iterative Statements: while, for, do-while

1. How many times is the function `sin(x)` called in the following code fragment?

```
#include <math.h>
...
double sum1 = 0., sum2 = 0.;
int n = 100;
while (n > 0) {
    if (sin(M_PI/n) > 0)
        sum1 += sin(M_PI/n);
    else
        sum2 += sin(M_PI/n);
    n--;
}
```

2. What is the output of the following program?

```
#include <iostream.h>

int main()
{
    int count, max = 6;
    for (count = 1; count < max; count++)
        cout << ++count;
    cout << endl;
    return 0;
}
```

---

3. Describe the behavior of the following program:

```
#include <iostream.h>

int main()
{
    int count = 0;

    while (count <= 1) count += 0.1;
    cout << count << endl;
    return 0;
}
```

- A) Syntax errors—will not compile
- B) Displays 1.1
- C) Displays 1
- D) Goes into an infinite loop
- E) Displays 0



## 4. ■ Write a program

$$1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots - \frac{1}{n} + \dots$$

converges to  $\ln 2$  — the natural log of 2. Write a program that prompts the user for a positive integer  $n$ , estimates  $\ln 2$  by adding the first  $n$  terms of the sequence, displays the estimate together with the value of `log(2.)` returned by the library function `double log(double x)`, declared in `math.h`. Note that in C++, `log(...)` returns the natural log and `log10(...)` returns the logarithm base 10.

## 5. ■ Write a function

```
double pow(double x, int n);
```

that returns the value of  $x^n$ . Assume that  $x \neq 0$ ,  $n \geq 0$ .

6. ■ Given a positive number  $a$ , the sequence of values

$$x_0 = \frac{a}{2}$$

$$x_{n+1} = \frac{1}{2} \left( x_n + \frac{a}{x_n} \right) \quad (n \geq 0)$$

converges to  $\sqrt{a}$ . Fill in the blanks in the following function that uses iterations to estimate the square root of a number:

```
double SqrtEst(double a)
// a is a positive number.
// Returns an estimate r of the square root of a, such that
// |r^2 - a| < 0.01 .
{
    double r = a/2.;
    double diff;

    do {
        _____;
        _____;
    } while (diff > .01 || diff < -.01);
    return r;
}
```

- 7.** Fill in the blanks in the following function that sets the elements of an array to the integers 1 through  $n$ , where  $n$  is the length of the array:

```
void SetSequence (apvector<int> &v)
// Sets the elements of the vector v to
// consecutive integers, starting from 1.
{
    int i, n = _____;
    for ( _____ )
        _____;
}
```

- 8.** Find and fix the bug in the following code:

```
apvector<char> hello(6);
int i;

hello[0] = ' '; hello[1] = 'h'; hello[2] = 'e';
hello[3] = 'l'; hello[4] = 'l'; hello[5] = 'o';

// Shift to the left and append '!':

i = 0;
while (i < 6) {
    hello[i-1] = hello[i];
    i++;
}
hello[5] = '!';
```

- 9.** What output is produced by the following statements?

```
const int SIZE = 4;
int i;
apvector<int> a(2*SIZE, 1);

for (i = 0; i < SIZE; i++)
    a[2*i] = i + 1;
for (i = SIZE; i < 2*SIZE; i++)
    cout << a[i] << ' ';
cout << endl;
```

**10. ■** Write a function

```
double Polynomial(const apvector<double> &a, double x)

// Returns the value of the n-th degree polynomial
// P(x) = a[0] + a[1]*x + a[2] * x^2 + ... + a[n] * x^n.
// Assumes that the length of a is n+1.
{
    _____
    _____
    _____
    _____
    _____
}

```

**11. ■** Fix the bug in the following function (but keep the while loop):

```
int CountOdds (const apvector<int> &a)

// Returns the number of odd values in the array.

{
    int count = 0, i = 0, n = a.length();
    while (i < n) {
        if (a[i] % 2 == 0) continue;
        count++;
        i++;
    }
    return count;
}

```

**12.** What does the `MysteryCount (...)` function count?

```
int MysteryCount(const apvector<int> &v)

// Returns ...

{
    int i, n = v.length(), count = 0;

    for (i = 0; i < n; i++) {
        if (v[i] != 0) break;
        count++;
    }
    return count;
}

```



15. Fill in the blanks in the following function:

```
int AlternateSum(const apvector<int> &v)
// Returns
//   v[0] - v[1] + v[2] - v[3] + ... - v[n-1],
//   where n is the number of elements. Assumes that n is even.
{
    int i, n = v.length(), sum = 0;
    for ( _____ ) {
        _____
        _____
    }
    return sum;
}
```

16. ♦<sup>▣</sup> A non-negative “big integer” is represented as an array of  $N$  digits. The value of each digit is an integer between 0 and 9. The most significant digits are at the beginning of the array, and zero values indicate leading zeroes. Fill in the blanks in the following function that calculates the sum of two “big integers,”  $a$  and  $b$ :

```
const int N = 100;

void Add(const apvector<int> &a, const apvector<int> &b,
         apvector<int> &sum)

// Calculates the sum of two "big integers" a and b
// represented as arrays of digits. Places the result in the
// array sum.
// Assumes that the result fits into N digits: the overflow
// condition is ignored.
{
    int i, d, carry = 0;
    sum.resize(N);
    for ( _____ ) {
        d = a[i] + b[i] + carry;
        sum[i] = _____;
        carry = _____;
    }
}
```

Write a test program that defines and initializes (or lets the user enter) two arrays and displays their “long” sum.

## Chapter 8. The switch Statement

1. <sup>□</sup> A credit card account number is represented as an array of 16 digits. Visa account numbers start with 4, Mastercard with 5, American Express with 37 and Discover with 6011. Write a function

```
char CreditCardType(const apvector<char> &account);
```

that identifies the credit card type and verifies the second, third and fourth digits, where necessary. The function returns the type of the card: V for Visa, M for Mastercard, A for American Express, D for Discover, and X for all other types or invalid numbers. Implement your function using a switch on the first digit.

2. Rewrite the following code with no `switch` or `if` statements:

```
int month, days;
...
switch (month) {
    case 1: days = 31; break;
    case 2: days = 28; break;
    case 3: days = 31; break;
    ...
    case 11: days = 30; break;
    case 12: days = 31; break;
}
```

3. If a program declares an enumerated type

```
enum COLOR {RED, BLUE, YELLOW, ORANGE, GREEN};
```

and the program includes a switch statement

```
switch (item) {
    ...
}
```

then `item` cannot be of which of the following types?

- A) char    B) short    C) int    D) double    E) COLOR

**4. ■** Write a function

```
void Translate(apvector<char> &str);
```

that examines a character string `str` and replaces all tab and newline characters with spaces; `'\"'` with `'/'`; `'~'` with `'!'`; and double quotes with single quotes. Use a switch statement.

**5. ◆** Write a program that plays Rock-Paper-Scissors with the user:

```
Rock... Paper... Scissors... Shoot!
Make your move (r, p, s) or q (to quit): s

I said Rock
Ha! You are zapped -- 1:0

Rock... Paper... Scissors... Shoot!
Make your move (r, p, s) or q (to quit): p

I said Paper
Paper-aper! Tie -- 1:0

Rock... Paper... Scissors... Shoot!
Make your move (r, p, s) or q (to quit): q

Sorry, you lost! 1:0
Thanks for playing.
```

Use nested switch statements. Try to invent a simple but not immediately obvious strategy for the computer, or use your compiler's on-line help to find a library function that generates random numbers.

## Chapters 6-8 Review

1. Circle all the expressions below that are equivalent (i.e., have the same value for all values of the variables a and b) to `!(a || !b)`:

- (a) `!a || !b`
- (b) `!a || b`
- (c) `!a && b`
- (d) `!(a-b)`

2. Write a function

```
char Grade(int score);
```

that computes the letter grade for a given cumulative score as follows:

| Score        | Grade |
|--------------|-------|
| 93 or higher | A     |
| 85-92        | B     |
| 72-84        | C     |
| 60-71        | D     |
| Less than 60 | F     |

3. Write a program that supports the following dialog with the user:

```
Enter quantity: 75
You have ordered 75 floppies -- $19.50

Next customer (y/n): y

Enter quantity: 97
Floppies can be ordered only in packs of 25.

Next customer (y/n): n

Thank you for using Floppy Systems.
```

(Define the unit price of a floppy disk as a constant equal to 26 cents.)

4. Write a program that produces the following output (where the user may enter any positive integer under 20):


```
Enter a positive integer under 20: 6
1 + 2 + 3 + 4 + 5 + 6 = 21
```

5. Write a function that determines whether a given number is a median for values stored in an array:

```
bool IsMedian(const apvector<int> &sample, double m)
// Returns true if m is a median for values in the array sample,
// false otherwise.
// (Here we call m a median if the number of elements that are
// greater than m is the same as the number of elements that are
// less than m.)
```


6. Fill in the blanks in the following function:

```
void Wedge(apvector<int> &w, int n)
// Sets the elements of the array w to values
// 1, 2, ..., n-1, n, n-1, ..., 2, 1.
// Resizes the array as necessary.
{
    int i = 1;
    _____;
    while (i <= n) {
        w[ _____ ] = i;
        w[ _____ ] = i;
        i++;
    }
}
```

7.  Finish the poem:  
*One, two, buckle your shoe;*  
*Three, four, shut the door;*

...

and write a program that displays the appropriate line of your poem:

```
Enter a number 1-10 (or 0 to quit): 1 
Buckle your shoe

Enter a number 1-10 (or 0 to quit): 2
Buckle your shoe

Enter a number 1-10 (or 0 to quit): 0
Bye
```

Use a switch statement.

## Chapter 9. Algorithms

1. Write a function

```
bool Scramble(apvector<char> &word);
```

that takes an array of letters and moves them around so that any two elements that were neighbors become separated. (The values of the characters do not matter: you can envision them all as different letters.) The function returns `true` if successful, `false` otherwise. Do not use any temporary local arrays.

2. A non-negative “big integer” is represented as an array of  $N$  digits. The value of each digit is an integer between 0 and 9. The most significant digits are at the beginning of the array, and zero values indicate leading zeroes. Write a function

```
void PrintBigInt(const apvector<int> &a);
```

that displays the number after skipping the leading zeroes.

3. Write a program that converts military time into the a.m./p.m. form:

```
Enter military time (e.g., 23:59) ==> 19:11  
7:11 p.m.
```



4. Assume for a moment that the C++ binary operators `+` and `-` work only for non-negative numbers and that the result of subtraction must be non-negative, too; however, the unary negation operator and the relational operators work as usual. Write a function

```
int Difference(int a, int b);
```

that returns  $a - b$ .

5. Fill in the blanks in the following function that returns the average of the two largest elements of an array:

```
double AverageTopTwo (const apvector<int> &scores)

// Finds the two largest elements in scores and returns their
// average.
// Assume that the size of the array is >= 2.

{
    int i, size = scores.length();
    int imax1 = 0;           // index of the largest element
    int imax2 = 1;           // index of the second largest element.

    // if scores[imax2] is bigger than scores[imax1] --
    // swap imax1 and imax2
    if (scores[imax2] > scores[imax1]) {
        i = imax1;

        _____
        _____
    }

    for (i = 2; i < size; i++) {
        if (scores[i] > scores[imax1]) {

            _____
            _____

        }
        else if ( _____ )

            _____
    }
    return _____
}
```

6. Recall that  $1 + 3 + \dots + (2p - 1) = p^2$  for any integer  $p \geq 1$ . Write a “simple” function which finds out whether a given number is a perfect square. A “simple” function cannot use arrays, nested loops, math functions, or arithmetic operations except addition.

```
bool IsPerfectSquare(int n)

// n is any positive integer.
// Returns true if n is a perfect square and false otherwise.
{
    ...
}
```

7. ■ An ISBN (International Standard Book Number) has ten digits. The first nine digits may have values between '0' and '9'; they identify the country in which the book was printed, the publisher, and the individual book. The tenth digit is a check digit assigned in such a way that the number  $d_1d_2d_3d_4d_5d_6d_7d_8d_9d_{10}$  has the property:

$$(10d_1 + 9d_2 + 8d_3 + 7d_4 + 6d_5 + 5d_6 + 4d_7 + 3d_8 + 2d_9 + d_{10}) \bmod 11 = 0$$

“mod” stands for modulo division. If  $d_{10}$  needs the value 10 to balance the check digit equation, then the character 'X' is used. For example, 096548534X is a valid ISBN.

Note that if we simply took the sum of all the digits, the check digit would remain valid for any permutation of the digits. Different coefficients make the number invalid when any two digits are swapped, catching a common typo.

Write a function

```
bool IsValidISBN(const apvector<char> &isbn);
```

that returns true if isbn is a valid number, false otherwise.

8. ■ The following function rearranges the elements in an array in such a way that it ends up partially ordered: all the negative numbers appear to the left of all the non-negative numbers. Fill in the blanks and test your function.

```
void SwapPosNeg(apvector<double> &v)
{
    int size = v.length(), i = 0, j = size - 1;
    double temp;

    while (i < j) {
        if (v[i] < 0) // Skip it
            _____
        else if ( _____ ) // Skip it
            _____
        else { // if both out of place--swap them
            _____
            _____
            _____
        }
    }
}
```

9. ■ Write a function that rotates an array of a given size  $n$  by a given number of steps  $d$ :

```
Rotate (apvector<int> &v, int d);
```

A positive  $d$  rotates the array forward; a negative  $d$ , backward. For example, if  $v$  holds elements 1, 4, 9, 16, 25, after `Rotate(v, -2)` the values in  $v$  are 9, 16, 25, 1, 4.

10. ♦ Write and test a function

```
void SpiralPrint (const apmatrix<char> &puzzle);
```


that prints out all the elements of a 2-D table of symbols in “spiral” order, clockwise. For example, the table

```
H e l l  
l d ! o  
r o W ,
```

should produce:

```
Hello,World!
```


## Chapter 10. Monte Carlo Methods

1. Write a function that returns a randomly selected letter between 'A' and 'Z'.
2. Write a function that simulates a throw of two dice and returns the sum of the points.
3.  Use the Monte Carlo method to estimate  $\ln 2$ , the natural logarithm of 2.  $\ln 2$  is equal to the area under the hyperbola  $y = 1/x$  between  $x = 1$  and  $x = 2$ , and your estimate should be based on that fact. Compare your result with the value returned by the library function `double log(double x)` (declared in `math.h`). Note that in C++, `log(...)` returns the natural logarithm and `log10(...)` returns the logarithm base 10.

4.  Write and test a function

```
void Shuffle(apvector<int> &v);
```

that rearranges the elements of a given array in random order, with equal probability for each outcome.

5.  Using a Monte Carlo simulation, estimate the probability that at least two people in a random group of 20 people have birthdays on the same day. Disregard leap years. Hint: zero out an array of 365 counters, then generate a random set of 20 integers between 0 and 364 and increment counters with the corresponding subscripts. Check whether any counter is greater than 1. Repeat many times to estimate the probability of coinciding birthdays.



## Chapter 11. Pointers, References, Dynamic Memory Allocation

Mark *true* or *false*:

1. Both pointers and references hold addresses of memory locations. \_\_\_\_\_
2. When a reference variable is declared, it must always be initialized. \_\_\_\_\_
3. The `new` operator returns a reference. \_\_\_\_\_
4. References may be used to pass arguments to functions “by reference.” \_\_\_\_\_
5. A function may return a pointer but not a reference. \_\_\_\_\_
6. The same variable may appear in the program as both an lvalue and an rvalue.  
\_\_\_\_\_

Which of the following lines are syntactically valid declarations?

7. `double amt, *pAmt = 0;`
8. `char *str, ch = &str;`
9. `int n, r = &n;`
10. `char ch, &rch = ch, *pch = &ch;`
11. `double z, &z1 = z, &z2 = z;`
12. `int k, *pk, **ppk;`

What is the output of the following statements?

13. 

```
char ch1 = '*', ch2 = '+', *s = &ch1;
ch2 = *s;
cout << ch1 << ch2;
```

 \_\_\_\_\_

14. 


```
int x = 3, y = 9, &r = y, *p;
p = &y;
*p = 0;
cout << x << r;
```

 \_\_\_\_\_

15. 

```
double u = 1.1, v = 1.2, *max = &u;
if (v > u) max = &v;
cout << *max;
```

 \_\_\_\_\_

16. Write a function `bool Reciprocal(...)` that takes one argument  $x$  of the type `double`, passed by reference. If  $x$  is not 0, the function sets  $x$  to  $1/x$  and returns `true`; otherwise it returns `false`.
17.  Fill in the blanks and test the following function:

```
bool GetRange(const apvector<int> &v, int &vMin, int &vMax)

// Finds the minimum and the maximum value in the
// array v and places them into vMin
// and vMax, respectively. Assumes that the array is not empty.
// Returns true if not all the elements of the array
// have the same value, false otherwise.

{
    int i, n = _____;

    vMin = _____;

    vMax = _____;

    for (_____ ) {
        if (_____ )
            _____;

        if (_____ )
            _____;
    }
    return (_____);
}
```

18.  Write and test a function

```
void SampleStats(const apvector<double> &x,
                 double &mean, double &stddev);
```

that computes the mean and the standard deviation of a statistical sample (presented as an array of doubles). The mean of a sample  $x_1, \dots, x_n$  is defined as:

$$m = \frac{x_1 + \dots + x_n}{n}$$

and the standard deviation is defined as:

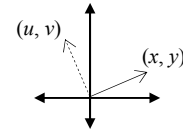
$$s = \sqrt{\frac{(x_1 - m)^2 + \dots + (x_n - m)^2}{n}}$$

- 19.** Recall that if  $a$  and  $b$  are the real and imaginary parts of a complex number, then that number squared has the real and imaginary parts  $a^2 - b^2$  and  $2ab$ , respectively. Find and fix the bug in the following code:

```
void SquareComplex(double &a, double &b)
// Squares a complex number a + bi
{
    a = a*a - b*b;
    b = 2*a*b;
}
```

- 20.■** Consider the following function:

```
void RotateVector(double &x, double &y, double &u, double &v)
// For coordinates x, y of a vector on the plane,
// calculates the coordinates u, v of the vector rotated
// 90 degrees counterclockwise.
{
    u = -y;
    v = x;
}
```



The arguments  $u$  and  $v$  are properly passed to the function by reference. It works fine when  $u$  and  $v$  are different from  $x$  and  $y$ . For example:

```
double x = 0., y = 1., u, v;
...
Rotate(x, y, u, v);
...
```

But what happens if  $u$  and  $v$  are the same as  $x$  and  $y$  in the function call? For example:

```
double x = 0., y = 1.;
...
Rotate(x, y, x, y);
...
```

This is a common problem called *aliasing*: different names  $x$  and  $u$  in the function argument list mask the fact that they may actually refer to the same variable. Fix the code of the function `Rotate(...)` to avoid the aliasing bug.



## Chapter 12. Strings

Mark *true* if the two declarations in a pair can be used interchangeably in all programs:

1.    `char msg[] = "Yes";`  
      `char msg[4] = "Yes";`    \_\_\_\_\_

2.    `char *msg = "No";`  
      `char msg[3] = "No";`    \_\_\_\_\_

3.    `apstring msg = "Maybe";`  
      `char msg[4] = "Maybe";`    \_\_\_\_\_

4.    Find a bug in the following declaration:

```
apstring fileName = "c:\dicts\english.txt";
...
```

5.    If the `stricmp(...)` function compares two null-terminated strings similarly to the `strcmp(...)` function but disregards upper- and lowercase letters, determine whether

```
if (stricmp(s1, s2) != 0)
```

works exactly the same way as

```
if (strcmp(strlwr(s1), strlwr(s2)) != 0)
```

**Given the declaration:**

```
apstring stars = "*****";
```

**which of the following are syntactically correct statements?**

6.    `stars = char stars[10];`

7.    `stars = "char stars[10]";`

8.    `apstring banner = stars;`

9.    `char *banner = stars;`

10.    `apstring banner = stars + "-|-|-|-|";`

- 11.▪** `c_str()`, a member function in the `apstring` class, returns a pointer to the actual null-terminated string buffer associated with the string. For example, in the following code:

```
apstring name;  
cin >> name;  
int len = strlen(name.c_str());
```

`len` will get the same value as in

```
int len = name.length();
```

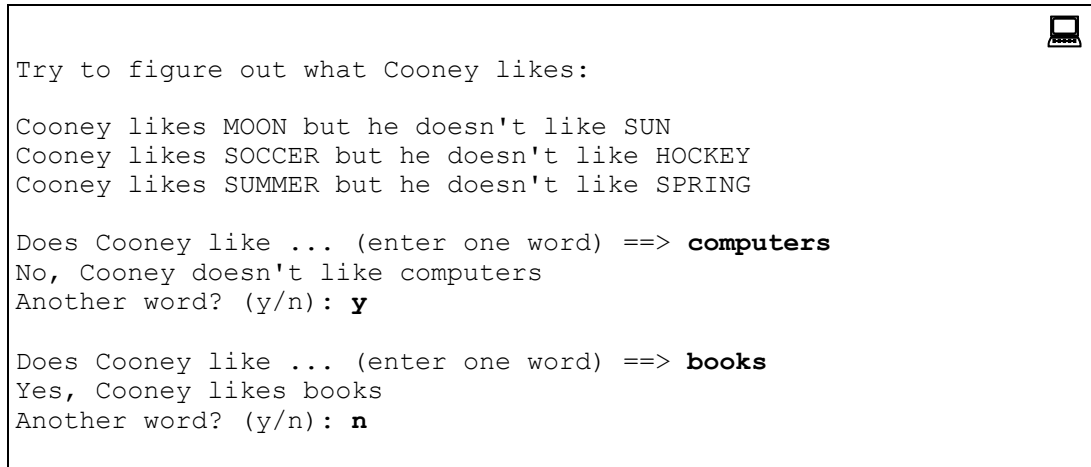
Which of the two versions is more efficient?

- 12.▪** What is wrong with the following function definition?

```
bool AllLetters(const apstring &str, int len)  
// Returns true if str contains only letters.  
// len is the length of the string.  
{  
    len = str.length();  
    ...  
}
```

Eliminate the redundancy.

13. ■ The Cooney game produces the following dialog with the user:



```

Try to figure out what Cooney likes:

Cooney likes MOON but he doesn't like SUN
Cooney likes SOCCER but he doesn't like HOCKEY
Cooney likes SUMMER but he doesn't like SPRING

Does Cooney like ... (enter one word) ==> computers
No, Cooney doesn't like computers
Another word? (y/n): y

Does Cooney like ... (enter one word) ==> books
Yes, Cooney likes books
Another word? (y/n): n

```

Guess the rule for words that Cooney likes. Write a function

```
bool CooneyLikes(const apstring &word);
```

and use it in a program that plays the Cooney game. The program automatically stops after five consecutive correct guesses. Use the `apstring` class for all strings.

14. ■ Write a program that reads a text file and displays it with a line number at the beginning of each line.
15. ◆ In MS DOS/Windows systems, a file name consists of up to eight characters (excluding '.', ':', back slash, '?', and '\*'), followed by an optional '.' and extension. The extension may contain between zero and three characters. For example: `1stfile.txt` is a valid file name. File names are case-blind. Write and test a function

```
apstring EnterFileName();
```

that prompts the user to enter a file name, validates the input, displays appropriate error messages if the name is invalid, appends the default extension ".txt" if no extension is given (if no "." appears in the input string), converts the filename to the upper case, and returns the resulting string to the calling program.

16.♦<sup>□</sup> Write a program that accepts user input in the form

*int1 op int2*

parses (analyses) the input, and displays the result of the arithmetic operation. Let *int1* and *int2* be integers and let *op* be +, -, \* or /.

## Chapters 9-12 Review

**1. Write a function**

```
int CountRed(const apvector<COLOR> &rgb);
```

that returns the number of RED values in an `rgb` array. `COLOR` is defined as:

```
enum COLOR {RED=1, GREEN, BLUE=4};
```

**2. Write a function**

```
int Max5(const apvector<int> &scores);
```

that returns the largest sum of five consecutive elements in an array `scores`. Assume that `scores` has at least five elements.

**3. Write a function**

```
void InsertMiddle(apvector<double> &v, double x);
```

that inserts a value `x` in the middle of an array `v`. Assume that the current length of the array is even and resize it as necessary.

**4. Fill in the blanks in the following function:**

```
enum COLOR {WHITE, BLACK};

int CountRuns(const apvector<COLOR> &pixels)

// Returns the number of "runs" in the pixels array.
// A "run" is a segment of all (one or more) consecutive pixels
// of the same color.

{
    int _____;
    for ( _____ )
        if ( _____ )
            _____;
    return count;
}
```

5. <sup>▣</sup> (a) Write a program that prints out all positive integers under 100 that are equal to the double product of their digits. For example:  $36 = 2(3 \cdot 6)$ .

(b) <sup>▣</sup> Do the same for all positive integers under 1000 that are equal to four times the product of their digits.

6. <sup>▣</sup> Write and test a function

```
int CountDifferent (const apvector<int> &a);
```

that returns the number of different values stored in an array of integers.

7. <sup>▣</sup> Write a function

```
void RandomTime(int &hours, int &mins, char &a_pm);
```

that sets `hours`, `mins` and `a_pm` to a random time of the day. The function sets `a_pm` to either 'a' or 'p'. Test the randomness of your function by verifying that the fraction of generated times that fall between 9 a.m. and 5 p.m. is close to  $1/3$ .

**What is the output of the following code?**

**8.**

```
char *yes = "yes";  
if (*yes == 'y') yes = "no";  
cout << yes << endl;
```

\_\_\_\_\_

**9.**

```
double z = 2000, &century = z;  
cout << century++;
```

\_\_\_\_\_

**10.**

```
int a = -7, b = 3, *p = &a;  
cout << *p;  
p = &b;  
cout << *p << endl;
```

\_\_\_\_\_

**Write and test the following functions:****11.** 


```
bool HasPunctuation(const apstring &str)
// Returns true if str includes a period, a comma,
//   a semicolon, or a colon; false otherwise.
```

**12.** 

```
bool HasWhiteSpace(const apstring &str)
// Returns true if str includes a character classified
//   as whitespace by the isspace() function (i.e., space, tab,
//   vert. tab, newline, formfeed, or carriage return),
//   false otherwise.
```


**13.** 

```
bool HasDuplicates(const apstring &str)
// Returns true if str has at least two identical characters,
//   false otherwise.
```

**14.**  Using the `apstring` class functions and operators, write and test a function

```
apstring SwapFirstLast(const apstring &name);
```

The function's argument `name` contains a person's first and last name (two words, separated by one space). The function builds and returns a new string in which the last name is placed first, followed by a comma, a space, and the first name.

**15.**  A playing card can be described by its rank (`int` 1 through 13) and suit:

```
enum SUIT {SPADE, HEART, DIAMOND, CLUB};
```

(a) Write a function

```
bool FullHouse(const apvector<int> &cardRank);
```

that returns true if a given poker hand of five cards is a Full House (three cards of the same rank plus a pair, also matching in rank).

(b) Write a function

```
bool StraightFlush(const apvector<int> &cardRank,
                   const apvector<SUIT> &cardSuit);
```

that returns true if a given poker hand of five cards is a Straight Flush (five consecutive cards of the same suit).

16. ♦<sup>□</sup> Using a Monte Carlo simulation, estimate the probability of getting a Full House (three cards of the same rank plus a pair, also matching in rank) on a random deal out of a deck of 52 cards (four suits, 13 cards of different ranks in each suit). Compare your estimate with the theoretical result,  $6/4165$ .

## Chapter 13. Structures

### Mark *true* or *false*:

1. After the `struct Somename { ... }` definition, `Somename` becomes a new, user-defined data type. \_\_\_\_\_
2. The members of a structure may occupy arbitrary locations in memory. \_\_\_\_\_
3. The assignment operator, when applied to structures, by default means member by member assignment. \_\_\_\_\_
4. Members of a structure may be accessed only through a pointer to the structure. \_\_\_\_\_
5. Structures may be passed as arguments to functions both by value and by reference. \_\_\_\_\_

### Questions 6-11 use the following definitions:

```
const int MAXCHAPTERS = 30;

struct CHAPTER {
    apstring title;
    int nPages;
    int firstPageNo;
};

struct BOOK {
    apstring title;
    int nChapters;
    apvector<CHAPTER> chapters;
    int nPages;
};

BOOK bk;
```

6. Write the expression for the number of chapters in `bk`.  
\_\_\_\_\_
7. Write the expression for the title of the third chapter in `bk`.  
\_\_\_\_\_
8. Write the expressions for the number of pages and the first page number in the last chapter in `bk`.  
\_\_\_\_\_  
\_\_\_\_\_

**9. ■** Write a function

```
void TOC(const BOOK &book);
```

that prints the table of contents (the title and first page number of each chapter in the book). In the main program, define and initialize a `BOOK` variable and test your function.

**10. ■** Write and test a function

```
void SetPagination(BOOK &book);
```

that sets the `firstPageNo` field in each of the chapters, assuming that the `nPages` fields are properly initialized. The first chapter must start on page 1, and each consecutive chapter should start on the next odd page following the end of the previous chapter. The function should also set the `nPages` member in the `BOOK` structure equal to the total number of pages in the book, which must be an even number.

**11. ■** Write and test an overloaded `<<` operator for the `BOOK` structure that displays the book title, the number of chapters, and the total number of pages in the book. For example:

```
cout << bk << endl;
```

```
The Birds of New England: 9 chapters, 160 pp.
```

**12. ■** Design the constants, structures, and data types necessary to represent a computer data entry form. A form may contain up to 12 fields. Each field is described by its name, position, width, whether it is alphabetic, numeric, or alphanumeric, and its current value (a character string). A form is described by its name, the number of fields it has, and the array of fields. For example, this is a form with three fields:

```
INVENTORY
```


```
Part No.: 53124      Descr: Machine screw 1 1/2 x 3/16_____
```




```
Price: $____.06
```



**Questions 13-16 refer to a file of credit card transaction records that contains fixed-length records of 77 bytes in the following format:**

| 0                                                                   | 1        | 2     | 3    | 4           | 5                   | 6         | 7  |
|---------------------------------------------------------------------|----------|-------|------|-------------|---------------------|-----------|----|
| 0123456789012345678901234567890123456789012345678901234567890123456 |          |       |      |             |                     |           |    |
| Date                                                                | TranCode | Amt   | SIC  | MerchAcct   | Merchant_Name       | City      | ST |
| 06-29-93                                                            | 253      | 41.50 | 7996 | 17018103250 | SEA WORLD OF CALIFO | SAN DIEGO | CA |
| ...                                                                 |          |       |      |             |                     |           |    |

**(The transaction code, an integer, indicates cash, merchandise, or another type of transaction. SIC is the standard four-digit industry classification code for the establishment.) A sample file, TRANSACT.DAT,  has a few records in the specified format.**

13. Define a TRANSACTION structure that would represent all the information from a transaction record, and at the same time would be convenient for sorting the records by date and for calculating the total amount of several transactions.
14.  Write an overloaded << operator for the TRANSACTION structure that outputs a transaction in the format of the transaction file record.
15.  Write and test an overloaded >> operator that reads a transaction file record into a TRANSACTION structure.
16.  Write a program that takes two transaction files, sorted by date, and merges them into one sorted file.



## Chapter 14. Modularity

1. Describe the behavior of the following code:

```
#define sells shells
#ifdef sells
    cout << "She sells sea"
    cout <<
#include "sells"
    << endl;
    cout << "by the seashore" << endl;
#endif
```

where the file `sells` contains just one line:

```
"shells"
```

- A) Won't compile: syntax error
- B) No output
- C) Displays:

```
She sells seasells
by the seashore
```

- D) Displays:

```
She sells seashells
by the seashore
```

- E) None of the above.

2. Name at least five important advantages of modular programs:

(a) \_\_\_\_\_

(b) \_\_\_\_\_

(c) \_\_\_\_\_

(d) \_\_\_\_\_

(e) \_\_\_\_\_

- 3.** A project consists of two source modules, `test.cpp` and `module2.cpp`, and a header file `somefun.h`. `test.cpp` contains `main()`, which calls `SomeFun()` twice. `module2.cpp` contains `OtherFun()`, which calls `SomeFun()` once. `somefun.h` contains the definition of `SomeFun()`:

```
static void SomeFun() {cout << "Oh, what fun...\n";}
```

Both `test.cpp` and `module2.cpp` include `iostream.h` and `somefun.h` at the top.

Describe the program's behavior:

- A) Runs OK. The executable program contains only one copy of `SomeFun`'s code.  
 B) Runs OK. The executable contains 2 identical copies of `SomeFun`'s code.  
 C) Runs OK. The executable contains 3 identical copies of `SomeFun`'s code.  
 D) Compile error on one or both of the modules.  
 E) Link error.
- 4.** Answer Question 3 above where the keyword `static` is replaced by `inline`.
- 5.** Answer Question 3 where the keyword `static` is removed.
- 6.** A function that is called only internally within the module that defines it should be declared `static` for the following reasons:  
 A) To document the fact  
 B) To save space in the table of globals  
 C) To avoid name clashes with functions in other modules  
 D) All of the above  
 E) None of the above

- 7.** Explain the reasons for putting the text of a header file between directives:

```
#ifndef SOMENAME
#define SOMENAME
...
#endif
```

Mark *true* or *false*:

8. The compiler builds a table of all unresolved externals for each module. \_\_\_\_\_
9. If a static function is placed in a header file, its compiled code will be duplicated in every module that includes that header file. \_\_\_\_\_
10. Inline functions save space in the object code. \_\_\_\_\_
11. The term *locality* describes reduced interdependence between modules. \_\_\_\_\_
12. Static functions in different modules of the same project may have the same name. \_\_\_\_\_
- 
13. What is wrong with a project in which functions from module A call a function from module B and another function from module B calls functions from module A?
- A) The program goes into an infinite loop
  - B) It will be impossible to create the header file(s)
  - C) The linker won't be able to resolve externals
  - D) Code will be duplicated in object modules
  - E) Inelegant design



## Chapter 15. Classes

Mark *true* or *false*:

1. Classes and structures differ only in the default access property classification of the first unmarked group of members: private or public. \_\_\_\_\_
2. Only data members of a class can be private. \_\_\_\_\_
3. A class's source code can be compiled separately. \_\_\_\_\_
4. Functions defined inside a class definition are automatically treated as inline functions. \_\_\_\_\_
5. Public members of the class can be accessed without a prefix anywhere in the program. \_\_\_\_\_
6. Inline member functions cannot be overloaded. \_\_\_\_\_

Complete each sentence with the word *always*, *sometimes*, or *never*:

7. Constructors \_\_\_\_\_ take arguments.
8. Destructors \_\_\_\_\_ take arguments.
9. A constructor \_\_\_\_\_ has the same name as its class.
10. Inline member functions in a class are \_\_\_\_\_ declared in the class's header file.
11. Constructors \_\_\_\_\_ return a value.
12. Constructors \_\_\_\_\_ dynamically allocate memory by using the `new` operator.

Choose the right word to fill the blank:

13. Data members of a class are made private to achieve \_\_\_\_\_ (*encapsulation / reusability / simpler user interface to a class*).
14. A constructor is called \_\_\_\_\_ (*automatically / only / explicitly*) when a variable of the class type is declared.
15. A destructor is called automatically when a variable of the class type \_\_\_\_\_ (*is set to 0 / goes out of scope / is passed to a function*).
16. "Accessors" are member functions that provide access to the \_\_\_\_\_ (*public / private / static*) data members of the class.

- 17.** “Modifiers” are used to \_\_\_\_\_ (*validate arguments for / change / overload*) private members of the class.

**Determine whether the following statements refer to required C++ syntax or optional C++ style:**

- 18.** Names of member functions begin with a capital letter. \_\_\_\_\_  
**19.** Local variables in member functions do not have the same names as class members. \_\_\_\_\_  
**20.** Variables of the class type have different names from class members. \_\_\_\_\_  
**21.** A class definition is placed in the header file. \_\_\_\_\_  
**22.** All of a class's member functions are placed in the same source file. \_\_\_\_\_

- 23.** In the following code,

```
class MYCLASS {  
    int mNumber;  
    ...  
};
```

mNumber is:

- A) private    B) static    C) inline    D) global    E) public

**24.▪** Describe the behavior of the following program:

```
#include <iostream.h>

class POINT {

public:
    double GetX() {return x;}

private:
    double x, y;
};

int main()

{
    POINT point;
    double x, y;

    x = 200.;
    cout << GetX() << endl;
    return 0;
}
```

- A) Compile error    B) Link error    C) Displays 0    D) Displays 200  
E) Output may vary from run to run



**25.▪** Find a bug in the POINT::MoveTo(...) member function:

```
// POINT.H
const double WIDTH = 1024.;
const double HEIGHT = 768.;

class POINT {
private:
    double x, y;

public:
    // Modifier
    void MoveTo(double ax, double ay);
};

// POINT.CPP
#include "point.h"

void POINT::MoveTo(double ax, double ay)
{
    double x, y;

    if (ax >= 0. && ax <= WIDTH && ay >= 0. && ay <= HEIGHT) {
        x = ax;
        y = ay;
    }
}
```

**26.** The apvector class's destructor contains the statement:

```
delete [] mList;
```

Explain why delete is used with brackets, rather than simply

```
delete mList;
```

**27.** How many times is the constructor `MYCLASS ()` called in the following code?

```
int main()
{
    MYCLASS x, y;

    for (int i = 0; i < 3; i++) {
        MYCLASS z;
        ...
    }
}
```

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

**Mark *true* or *false*:**

- 28.** A class may have another class as a member. \_\_\_\_\_  
**29.** A structure may have a function as a member. \_\_\_\_\_  
**30.** A constructor may return a value. \_\_\_\_\_  
**31.** An “accessor” member function cannot take arguments. \_\_\_\_\_  
**32.** A structure and a class may have the same name in the same module. \_\_\_\_\_

**Complete each sentence with the word *always*, *sometimes*, or *never*:**

- 33.** Data members of a class are \_\_\_\_\_ private.  
**34.** Function members of a class are \_\_\_\_\_ private.  
**35.** A class's destructor \_\_\_\_\_ de-allocates memory.  
**36.** A class \_\_\_\_\_ has more than one constructor.  
**37.** A class \_\_\_\_\_ has a data member.
- 38.** Explain *encapsulation*.



## Chapter 16. Templates

Mark *true* or *false*:

1. Templated functions and classes are used to implement the same functionality for different data types. \_\_\_\_\_
2. A template function may have more than one parameterized type. \_\_\_\_\_
3. The compiler generates code for a template function or class only when it sees how that function or class is used. \_\_\_\_\_
4. Templated classes may use only a built-in type as a parameter. \_\_\_\_\_
5. Templates enhance the type checking discipline of C++. \_\_\_\_\_

Write and test a templated function that returns:

6.  The absolute value of a number.
7.  The average of two numbers, returned as a double.
8.  The maximum of two values.

Consider the templated function `OrderPair(...)`:

```
template <class SOMETYPE>
void OrderPair(SOMETYPE &x, SOMETYPE &y)

// Swaps the variables x, y if necessary to put them
// in ascending order.

{
    if (x > y) {
        SOMETYPE temp = x;
        x = y;
        y = temp;
    }
}
```

Which of the following pairs of variables can be arguments to this function for it to work as expected?

9. `int x = 2, y = 1;`
10. `double x = 2., y = 1.;`
11.  `long x = 2; int y = 1;`
12.  `apvector<int> x(2, 1), y(2, -1);`
13.  `apstring x = "Boston", y = "Atlanta";`
14.  `char *x = "Boston", *y = "Atlanta";`

15. Write a templated function that sorts an array of values using the selection sort algorithm. Test it on integers and `apstrings`.
16. ▣ The following class implements a point on a 2-D plane:

```
// POINT.H
class POINT {

private:
    double x, y;

public:
    POINT() {x = 0; y = 0;}
    POINT (double ax, double ay) {x = ax; y = ay;}
    double GetX() {return x;}
    double GetY() {return y;}
    void MoveTo(double ax, double ay) {x = ax; y = ay;}
};
```

Convert it into a templated class so that it works with `x-y` coordinates of any numeric type. Verify that your class works with integers as well as doubles.

17. ▣ The templated class `SEGMENT` represents a line segment on a plane by its two endpoints. The member function `Stretch3()` stretches the segment by a factor of three from its midpoint:

```
template <class COORD>
class SEGMENT {

private:
    COORD x1, y1;
    COORD x2, y2;

public:
    ...
    void Stretch3();
};

template <class COORD>
SEGMENT::Stretch3()

{
    COORD xMid = (x1 + x2) / 2;
    x1 = xMid + 3*(x1 - xMid);
    x2 = xMid + 3*(x2 - xMid);
    COORD yMid = (y1 + y2) / 2;
    y1 = yMid + 3*(y1 - yMid);
    y2 = yMid + 3*(y2 - yMid);
}
```

The class is intended to work with `int` and `double` data types, but it does not. Fix the bug.

## Chapters 13-16 Review

**Given the following declarations:**

```
struct BigInt {
    char sign;
    apvector<int> digits;
};

BigInt x, *p;
apvector<BigInt> v(5);
```

**which of these lines are syntactically correct?**

1. int x = x.digits[0];
2. int y = x.digits[3];
3. int digits = v[3].digits[3];
4. char sign = p->sign;
5. BigInt z = v[0];
6. int n = v[3].digits.length();

**Questions 7-8 refer to the structure STUDENT, defined as follows:**


```
struct STUDENT {
    apstring name;
    double GPA;
};
```

7. Write an overloaded << operator for the structure STUDENT.
8. Write an overloaded < operator for the structure STUDENT that compares students according to their GPAs.
  
9. What are the advantages of using a header file over explicitly including its text into each module that needs it?
  - I. Saves space in source modules
  - II. Saves space in object modules
  - III. Simplifies software maintenance


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

10. How do you protect a header file from being included in the same module twice?

- 11.** Write a function that prints

```
Please send $25.00 to have the data on your hard drive restored 
```

when the identifier HITDISK is #define'd, and

```
Thank you for using X-SELL by Ransom Software 
```

otherwise.

- 12.** What happens when the following declarations are placed in a header file?

```
extern double pi;  
double pi = 3.14;
```

- A) Always works fine
- B) Works only for a one-module project, but is a bad design decision
- C) Always produces linker error “Multiple definitions of pi”
- D) Always produces compiler error “Undefined variable pi”
- E) None of the above



**Complete each sentence with the word *always*, *sometimes*, or *never*:**

- 13.** A constructor is \_\_\_\_\_ called explicitly.
- 14.** A templated class \_\_\_\_\_ works with integers and doubles.
- 15.** A structure \_\_\_\_\_ has a destructor.
- 16.** A templated function \_\_\_\_\_ returns a value of the parameterized data type.
- 17.** An inline function is \_\_\_\_\_ defined as a templated function.

**18.** <sup>□</sup> Given the definitions:

```
// LADDER.H
class LADDER {
    public:
        LADDER(int size);
        ~LADDER();
        int Add();

    private:
        int *mList;
        int mSize;
};

// LADDER.CPP
LADDER::LADDER(int size)
{
    int i;
    mList = new int[size];
    mSize = size;
    for (i = 0; i < size; i++)
        mList[i] = i+1;
}

LADDER::~~LADDER()
{
    delete [] mList;
}

int LADDER::Add()
{
    int i, sum = 0;
    for (i = 0; i < mSize; i++)
        sum += mList[i];
    return sum;
}
```

fill in the blanks in `main()` to declare an instance of the class `LADDER` and use it to print the sum  $1 + 2 + 3 + \dots + 30$ .

Continued 

```
#include "ladder.h"

int main()
{
    _____;

    cout << _____;

    return 0;
}
```

## Chapter 17. Linked Lists

Mark *true* or *false*:

1. The term *Abstract Data Type* refers to a data type that is not a C++ built-in data type. \_\_\_\_\_
2. One of the advantages of linked lists over arrays is that we can find an element quickly in an ordered list. \_\_\_\_\_
3. A linked list takes more memory than an array to store the same number of data elements of the same type. \_\_\_\_\_
4. It usually takes less computer time to concatenate two arrays than two linked lists. \_\_\_\_\_
5. It usually takes less computer time to concatenate two linked lists with a tail than two regular singly linked lists. \_\_\_\_\_

6. Given the declarations

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

NODE
node3 = {13, _____ },
node2 = {11, _____ },
node1 = {7, _____ },
*head = &node1;
```

fill in the blanks in the initialization of `node3`, `node2` and `node1`, so that `node1`, `node2`, and `node3` form a linked list pointed to by `head`. Use the “address of” operator.

7. Given the following declaration:

```
struct NODE {
    SOMETYPE info;
    NODE *next;
};
```

write a function

```
void Rotate (NODE* &head);
```

that splits off the first node of the linked list pointed to by `head` and appends it at the end of the list. The function should accomplish this by rearranging pointers: do not allocate new nodes or move data items between nodes.

**8. Write a function**

```
NODE *Copy(NODE *head);
```

that creates a copy of a linked list and returns a pointer to its head.

**9. ■ Given the declarations**

```
struct POINT {  
    double x;  
    double y;  
};  
  
struct NODE {  
    POINT vertex;  
    NODE *next;  
};
```

write a function

```
void RemoveClosestVertex(const POINT &p, NODE* &polygon);
```

where `polygon` is the head of a linked list, passed by reference. The function finds the vertex closest to `p` and removes it from the list. Use the distance formula:

$$d^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2$$

**10. ■** Given the declarations

```
enum COLOR (RED = 1, GREEN, BLUE = 4);

struct POINT {
    double x;
    double y
};

struct RECT {
    double left;           // left < right;
    double right;
    double top;           // top < bottom;
    double bottom;
    COLOR color;
};

struct NODE {
    RECT rect;
    NODE *next;
};
```

write a function

```
bool IsInRed (const POINT &p, NODE *sketch);
```

where `sketch` is a linked list of rectangles of various colors. The function returns `true` if `p` is inside any red rectangle on the list, `false` otherwise.

**11. ■** Given the declarations


```
struct NODE {
    apstring data;
    NODE *prev;
    NODE *next;
};

struct LIST {
    NODE *head;
    NODE *tail;
};
```

write a function

```
void Append(LIST &list1, LIST &list2);
```

where `list1` and `list2` are doubly linked lists. The function concatenates `list2` to `list1`. Do not allocate or delete any nodes, just concatenate the lists and update the `list1` structure.

**12.**  In a *circular* linked list:

```
struct NODE {  
    int data;  
    NODE *next;  
};  
  
NODE *head;
```

the `next` pointer in the last node is set not to null, but to the first node of the list (in other words `lastnode->next == head`). Write a function

```
int Sum(NODE *head);
```

that returns the sum of all data elements in the list.

## Chapter 18. Stacks

1. What is the output of the following code:

```
apstack<char> stack;
char ch;

stack.push('A');
stack.push('B');
stack.push('C');
while (!stack.isEmpty()) {
    stack.pop(ch);
    cout << ch;
}
cout << endl;
```

- A) None    B) ABC    C) CCC    D) CBA    E) C

2. If `apstack<int> stack` contains

(top) -1 3 7 -2 4 -6

what is its content after the following code is executed?

```
...
int x;
apstack<int> stackPos, stackNeg;

while (!stack.isEmpty()) {
    stack.pop(x);
    if (x >= 0)
        stackPos.push(x);
    else
        stackNeg.push(x);
}

while (!stackPos.isEmpty()) {
    stackPos.pop(x);
    stack.push(x);
}

while (!stackNeg.isEmpty()) {
    stackNeg.pop(x);
    stack.push(x);
}
...
```

---

**3.** The following code uses an array of 5 stacks of integers:

```
#include "apstack.h"
...
apvector<apstack<int> > stacks(5);
int i, count, k;

// Set up the first stack:
for (k = 1; k <= 5; k++)
    stacks[0].push(k);

// Pop from the previous stack and push on the next stack:
for (i = 1; i < 5; i++) {
    for (count = 0; count < 5 - i; count++) {
        stacks[i-1].pop(k);
        stacks[i].push(k);
    }
}

// Print all stacks:
for (i = 0; i < 5; i++) {
    while (!stacks[i].isEmpty()) {
        stacks[i].pop(k);
        cout << k;
    }
}
```

What is the output?

- A) 12345    B) 54321    C) 15243    D) 12435    E) No output



**4.** In the following code, a stack of integers is used. Find and fix a bug in the code.

```
#include "apstack.h"

struct POINT {
    int x;
    int y;
};
...
POINT cursor;
apstack<int> stack;
...
stack.push(cursor.x);           // Save cursor position
stack.push(cursor.y);
OpenWindow(newApplet);         // Open a new window
...
stack.pop(cursor.x);           // Restore cursor position
stack.pop(cursor.y);
DrawCursor(cursor);
```

5. ■ The following program reads a binary number (a string of binary digits) from `cin` and displays the number as a decimal. Explain why the use of a stack here is overkill; rewrite without the stack.

```
#include <iostream.h>
#include "apstring.h"
#include "apstack.h"

int main()
{
    int i;
    apstring binnum;
    char ch;
    apstack<int> stack;
    int dig, val = 0, power2 = 1;

    cin >> binnum;
    for (i = 0; i < binnum.length(); i++) {
        ch = binnum[i];
        if (ch != '0' && ch != '1') break;
        stack.push(ch - '0');    // Push the int value: 0 or 1
    }

    while (!stack.isEmpty()) {
        stack.pop(dig);
        val += dig * power2;
        power2 *= 2;
    }
    cout << val << endl;
    return 0;
}
```

- 6.♦ A stack of characters can be implemented using the `apstring` class discussed in Chapter 12. Does the following code correctly implement the push and pop functions?

```
apstring stack;          // Declare an empty stack of chars;

void push (apstring &stack, char ch)
{
    if (stack.length() < MAXLENGTH)
        stack += ch;
}

void pop (apstring &stack, char &ch)
{
    int len = stack.length() - 1;
    if (len >= 0) {
        ch = stack[len];
        stack = stack.substr(0, len);
    }
}
```

Review the code for the `+=` operator and the `substr()` function in `apstring.cpp` and discuss the merits of the above implementation of stack.

## Chapter 19. Recursion

- 1.** A mystery function is defined as follows:

```
void MysteryFunction(int n)
{
    if (n < 0) return;
    cout << n;
    MysteryFunction(n-1);
    cout << n;
}
```

What output is produced by this call?

```
...
MysteryFunction(3);
...
```

- A) 33 B) 32103 C) 30123 D) 32100123 E) The program hangs

- 2.** What is the output when `Enigma(n)` is called with the argument 3?

```
void Enigma (int n)
// Prints something...
{
    int i;
    for (i = 0; i < n; i++)
        Enigma(i);
    cout << n;
}
```

- A) 0123 B) 3210 C) 00123 D) 00100123  
E) None of the above

3. What happens when a function `fun1 (...)` calls `fun2 (...)` and `fun2 (...)` calls `fun1 (...)` (assuming proper function prototypes and definitions)?

- A) Compiler error
- B) Link error
- C) The program always aborts with the “stack overflow” error
- D) The program always goes into an infinite loop
- E) Depends: may work fine in certain recursive programs



4. A picture consists of graphic elements and embedded pictures. Let us represent a picture as a linked list with nodes:

```
struct NODE {
    GRAPHOBJECT elmt;
    NODE *subpicture; // "subpicture" itself is a list of elements.
    NODE *next;
};
```

Assuming that

```
void Draw(const GRAPHOBJECT &elmt);
```

draws a graphic element, the function that draws the whole picture may look as follows:

```
void DrawPicture (NODE *picture)

// Draws a picture and all its subpictures. A picture is
// represented as a linked list pointed to by "picture".

{
    for (NODE *node = picture; node; node = node->next) {
        Draw(node->elmt); // Draw the element
                        // Draw the subpicture
    }
};
```

Explain why this function does not seem to have an explicit base case and fill in the blank for the recursive call.

**5.** Predict the output of `MysterySum(10)`, where

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

Justify your answer by using mathematical induction. Explain why this is an inappropriate use of recursion.

**6.** Fill in the blanks in the following function:

```
long CountPaths(int x, int y)
// Returns the number of all possible paths from the point (0,0)
// to the point(x,y), where x and y are any non-negative
// integers. From any point the path may extend only
// up or to the right by one unit (i.e., one of the current
// coordinates x or y can be incremented by one).
{
    if (x <= 0 || y <= 0)
        return _____;
    else
        return _____
        _____;
}
```

- 7.♦ On Beavis Island the alphabet consists of three letters — A, B, and H — but no word may have two A's in a row. Fill in the blanks in the following recursive function, `AllWords(...)` that prints out all Beavis Island words of a given length:

```
void AllWords(apstring &wordBuf, int count)

// wordBuf is a string of a given length. It contains
// the initial sequence of letters in a word
// that is being built.
// count is the number of letters currently in wordBuf.
//
// Call from main (or from another function) as follows:
// apstring wordBuf = "*****"; // A string of a given length;
// AllWords(wordBuf, 0);

{
    if (count == wordBuf.length()) {
        // Base case:
        _____; // Display the string
    }
    else { // Recursive case:
        if (count == 0 || wordBuf[count-1] != 'A') {
            // Append 'A' only if last
            // letter is not an 'A'
            _____;
            _____;
        }
        _____; // Append 'B'
        _____;
        _____; // Append 'H'
        _____;
    }
}
```

- 8.♦ Suppose we have a set of positive integers. We want to choose several of them so that their sum is as large as possible but does not exceed a given limit. This type of problem is called a Knapsack Problem. For example, we may want to choose several watermelons at the market so that their total weight is as large as possible but does not exceed the airline limit for one bag.

Write a recursive function that solves a simplified Knapsack Problem: it only calculates the optimal sum but does not report the selected items:

```
int KnapsackSum(const apvector<int> &w, int n, int limit)

// w contains positive integers.
// "n" is the number of elements in the array.
// Returns the sum of some elements of the array,
// so that it has the largest possible value that
// does not exceed "limit".
```

Use mathematical induction to prove that your code is correct. Can you think of an algorithm that uses neither recursion nor a stack?



## Chapter 20. Queues

1. The most economical structure for the linked-list implementation of a queue is
- A) A singly linked list    B) A doubly linked list  
C) A linked list with a pointer to the tail    D) A circular list  
E) None of the above

2. The following class implements a ring-buffer queue of characters:

```
const int BUFSIZE = 256;
class RBQUEUE {
    ...
private:
    char buf[BUFSIZE];
    int front;
    int rear;
};
```

`front` is the index of the first element and `rear` is the index of the first vacant slot. Write a member function

```
int RBQUEUE::length();
```

that returns the number of elements in the queue.

3. The IBM PC BIOS uses a keyboard ring buffer of 16 bytes, starting at the address 40:1E. The two-byte locations 40:1A and 40:1C represent the front and the rear of the keyboard queue respectively. These are integer addresses (offsets from 40:0) stored with the least significant byte first. Each pressed keyboard key adds two bytes to the keyboard buffer: the ASCII code of the character followed by the so-called *scan code* that represents the location of the key on the keyboard. Examine the following hex memory dump and determine the current contents of the keyboard queue and the last eight characters typed.

\_0 \_1 \_2 \_3 \_4 \_5 \_6 \_7 \_8 \_9 \_A \_B \_C \_D \_E \_F

|           |                                           |
|-----------|-------------------------------------------|
| 0040:0010 | 28 00 28 00 30 0B                         |
| 0040:0020 | 3A 27 31 02 61 1E 0D 1C-64 20 20 39 34 05 |

4. The following code uses two queues of integers:

```
#include "apqueue.h"
...
apqueue<int> q1, q2;
int k1, k2, sum;

for (k1 = 1; k1 <= 4; k1++)
    q1.enqueue(k1);
q2.enqueue(1);
q2.enqueue(1);
while (!q1.isEmpty() && !q2.isEmpty()) {
    q1.dequeue(k1);
    q2.dequeue(k2);
    sum = k1 + k2;
    cout << sum << ' ';
    q2.enqueue(sum);
}
```

What is the output?

- A) 2 3 5 8      B) 2 4 6 5      C) 2 4 7 11      D) 2 3 4 5  
E) None of the above

5. ♦ ■ The Total Image beauty salon offers its customers a car wash while they are getting a haircut. In its first month, The Total Image was flooded with customers, forcing the owner to install a computer system to keep things under control. The software uses four queues: arriving customers and their cars are assigned to the “hairdresser” and the “car wash” queues; when processed, they are moved to the “hair ready” and “car ready” queues. These four events are entered into the system by a human operator. The “asynchronous” handling is required because the duration of haircuts and car washes can vary. Write a program that properly processes the four operator commands corresponding to the four events and notifies a customer when he or she is “all set.”

## Chapters 17-20 Review

**1.** Write a function

```
NODE *MiddleNode(NODE *head);
```

that returns a pointer to the middle node of a linked list. Try to write your function using only one loop.

**2.▪** Write a function

```
void SplitOddEven(NODE *head,  
                 NODE* &head1, NODE* &head2);
```

that takes a linked list pointed to by `head` and splits it into two new linked lists pointed to by `head1` and `head2`. The function should place the first and every odd node into the first list and the second and every even node into the second list. Your function must not allocate new nodes or move the data elements between nodes.

**3.** A doubly linked list structure is defined as follows:

```
struct NODE {  
    apstring info;  
    NODE *prev;  
    NODE *next;  
};  
  
struct LIST {  
    NODE *head;  
    NODE *tail;  
};
```

Write a function



```
bool SingleNode(const LIST &list);
```

that returns true if the list has exactly one node, false otherwise.

4. Explain why an implementation of stack as a linked list with a tail, where the push function appends an item at the end of the list, is not very efficient:

- A) Needs twice as much memory
- B) Takes a long time to find the place to attach the new node in push
- C) Takes a long time to find the top stack item in pop
- D) Takes a long time to adjust head or tail in push
- E) Takes a long time to adjust head or tail in pop



5.   A book's index contains entries and sub-entries nested to several levels. Sub-entries are indicated by deeper indentation. All sub-entries of a given entry are preceded by the same number of spaces; that number is greater than the indentation at the previous level. For example:

```
class
  accessors
  constructors and destructors
    overloaded
    with arguments
  defined
  polymorphism
function
  inline
  static
stack class
  for handling nested structures
  member functions
    push
    pop
```

Write a program that reads a specified index file and verifies that all the entries and subentries are in alphabetical order. To keep things simpler, assume that all entries are in lowercase letters. Use the `apstring` class and relational operators to read the index entry lines from a file and compare strings. Define an `ENTRY` structure that can hold the current indentation offset and the index entry text. Use a stack of index entries.

6. ♦ □ Write a program, in which Cookie Monster finds the optimal path from the upper left corner (0,0) to the lower right corner (SIZE-1, SIZE-1) in a cookie grid (a 2-D array). The elements of the grid contain cookies (a non-negative number) or barrels (-1). On each step Cookie Monster can only go down or to the right. He is not allowed to step on barrels. The optimal path contains the largest number of cookies.

The program prompts the user for a file name, reads the cookie grid from the file, and reports the number of cookies on the optimal path.

*Hints:* If there is only one way to proceed from the current position, then go there and update the total accumulated number of cookies. If there are two ways to proceed, save one of the possible two points (and its total) on stack and proceed to the other point. If you have reached the lower right corner, update the maximum. If there is nowhere to go, examine the stack: pop a saved point, if any, and resume from there.

*Note:* A similar method can be used to traverse any *directed graph* (a diagram of nodes connected with arrows) that does not have circular paths. For example, one application of such a method is to find the shortest route from point A to point B on a road map. From any point we can take one route and leave all other possible routes on stack for future processing.

7. ♦ □ Rewrite the Cookie Monster program above using recursion. For recursive handling, it often helps to restate the question in more general terms. Here we need to refer to the optimal path from (0,0) to any position (*row*, *col*). So our `OptimalPath(...)` function should now take two arguments: `row` and `col`. Note that the maximum number of cookies accumulated at a position (*row*, *col*) is related to the previous positions as follows:

```
OptimalPath(row, col) = cookies[row][col] +
the larger of the two:
    {OptimalPath(row-1, col), OptimalPath(row, col-1)}
```

The only problem is invalid positions: either out of bounds or “barrels.” How can we define `OptimalPath(row, col)` for an invalid position (*row*, *col*), so that the above formula still works? Identify the base case(s) and recursive case(s).

8. ■ □ Rewrite the GCF function, which finds the greatest common factor of two positive integers (as described in Chapter 9), recursively.

9.♦<sup>▣</sup> A 6 by 6 gameboard contains arbitrarily arranged black and white squares. A path across this board may use only black squares and may move only down or to the right. Write and test a program that reads a board configuration from a file (stored by rows) and finds and prints out all paths leading from the upper left corner (0,0) to the lower right corner (5,5). Use queues to hold the points on the incomplete paths.

10.▣ Given the definition:

```
struct STUDENT {
    apstring name;
    double GPA;
};
```

write and test a function

```
void CutAtGPA(apqueue<STUDENT> &studentList,
             double minGPA, apqueue<STUDENT> &honorsList);
```

that removes student records one by one from the `studentList` queue, and adds those students whose GPA is not less than `minGPA` to the `honorsList` queue.

11. (a) Add a constructor to the `apqueue` class:

```
apqueue<itemType>::apqueue(int size);
```

that builds a queue with a buffer capable of holding `size` items and makes it initially empty.

(b) Add a member function:

```
bool apqueue<itemType>::isFull()
```

that returns `true` if the queue buffer is full and no new items may be placed into it without reallocating the buffer of a larger size.

(c) Modify the `enqueue(...)` function so that it does not adjust the buffer size automatically when the queue buffer is full.

12.▣<sup>▣</sup> Re-implement the `apqueue` class using a linked list with a tail and test the new class.

## Chapter 21. Classes: More Advanced Features

1. Name three major canonical features of a class:

---

---

---

Name a mechanism in the implementation of a class `SOMECLASS` that supports the following usage:

2.

```
SOMECLASS a;  
...  
SOMECLASS b = a; _____
```

3.

```
SOMECLASS a, b;  
...  
b = a; _____
```

4.

```
SOMECLASS a;  
...  
cout << a; _____
```

5.

```
OTHERTYPE a;  
...  
SOMECLASS b = a; _____
```

Mark *true* or *false*:

6. `friend` declarations may appear anywhere within a class definition. \_\_\_\_\_
7. If class A is a friend of class B, then B is always a friend of A. \_\_\_\_\_
8. An iterator is a companion class that supports traversals for a list class. \_\_\_\_\_
9. An iterator class is usually declared to be a `friend` of the class on which it iterates. \_\_\_\_\_

10. ■ Take the `apqueue` class and define a friend iterator class `qiterator`. Use the `qiterator` class to write a function

```
void PrintHiPriority(const apqueue<MESSAGE> &q, int threshold);
```

that scans through a queue of messages `q` and prints the text of all messages whose priority is equal to or above `threshold`. `MESSAGE` is defined as:

```
struct MESSAGE {
    apstring mText;
    int mPriority;
};
```

Mark *true* or *false*:

11. A static member of a class occupies the same memory location for all instances of a class. \_\_\_\_\_
12. A static member function can access only public members of its class. \_\_\_\_\_
13. A static member function can be called through any instance of its class. \_\_\_\_\_
14. A static data member can be initialized within the definition of the class. \_\_\_\_\_

Given the class definition

```
class MyClass {
private:
    static int offset;
    ...
public:
    static int GetOffset() {return offset;}
    ...
};
```

which of the following are allowed usages?

15. `int x = offset;`
16. `int x = MyClass::offset;`
17. `int x = GetOffset();`
18. `int x = MyClass::GetOffset();`
19. `MyClass mc; int x = mc.GetOffset();`

## Chapter 22. Trees

1. Define the following “tree” terms:

*root* \_\_\_\_\_

*child* \_\_\_\_\_

*leaf* \_\_\_\_\_

*parent* \_\_\_\_\_

*ancestor* \_\_\_\_\_

*depth* \_\_\_\_\_

2. What is the smallest number of levels required to store 100,000 nodes in a binary tree?
3. What is the smallest and the largest possible number of leaves in a binary tree containing exactly six non-leaf nodes?
4. Prove using mathematical induction that a binary tree of depth  $h$  cannot have more than  $2^h - 1$  nodes.
5. Prove using mathematical induction that in a binary tree with  $N$  nodes
- $$L \leq \frac{N+1}{2}$$
- where  $L$  is the number of leaves.

**Questions 6-12 use the following structure for a node of a binary tree:**

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

6. Write a function

```
bool IsLeaf(NODE *node);
```

that returns `true` if `node` is a leaf.

**7.** Write a function

```
int SumTree(NODE *root);
```

that returns the sum of the values stored in the tree pointed to by `root`.

**8.** What does the following function count?

```
long NumberOfSomething(NODE *root)
{
    long count;

    if (!root) return 0;
    if (!root->left && !root->right)
        count = 1;
    else
        count = NumberOfSomething(root->left) +
                 NumberOfSomething(root->right);
    return count;
}
```

**9.** Write a function

```
NODE *Copy(NODE *root);
```

that creates a copy of a given binary tree and returns a pointer to its root. Assume that there is enough memory to allocate all the nodes in the new tree.

**10.** Write a function

```
int Depth(NODE *root);
```

that returns the depth of the binary tree.

**11.▪** Write a function

```
NODE *MirrorImage(NODE *root);
```

that creates a mirror image of a given binary tree and returns a pointer to its root. Assume that there is enough memory to allocate all the new nodes.

**12.▪** Write a function

```
long CountPaths(NODE *root);
```

that returns the total number of paths that lead from the root to any other node of the binary tree.

13. ♦<sup>□</sup> (a) Write a function

```
NODE *BuildFull(int depth);
```

that builds a binary tree of the given depth with all levels completely filled with nodes. The function should set the values in all the nodes to zero and return a pointer to the root of the new tree.

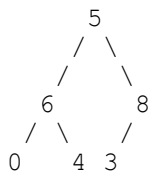
- (b) Write and test a function

```
void FillTree(NODE *root);
```

that appends new nodes (with 0 values) to the tree until all existing levels in the tree are completely filled.

Mark *true* or *false*:

- 14.** The smallest element in a binary search tree is always a leaf. \_\_\_\_\_
- 15.** If a binary search tree holds integers and the root holds 0, then all the nodes of the left subtree hold negative numbers. \_\_\_\_\_
- 16.** If a binary search tree is traversed inorder, all the nodes will be listed in ascending order. \_\_\_\_\_
- 17.** The iterative version of the `Find(...)` function that searches a binary search tree may be as short as the recursive version. \_\_\_\_\_
- 18.** The number of comparisons necessary to find a target node in a binary search tree never exceeds  $\log_2 n + 1$ , where  $n$  is the number of nodes. \_\_\_\_\_
- 19.** Swap two nodes in the tree below to make it a binary search tree:



- 20.** Suppose we start with an empty binary search tree and add elements

475, 474, 749, 623, 292, 557, 681

(in that order). Draw the resulting tree. How can we arrange the same numbers in a balanced binary search tree?

**21.** Write a non-recursive function that returns the value of the largest element in a non-empty binary search tree.

**22.** Draw the binary search tree created by inserting the letters

L O G A R I T H M

(in that order) into an empty tree. List the nodes of this tree when it is traversed:

inorder \_\_\_\_\_

preorder \_\_\_\_\_

postorder \_\_\_\_\_

## Chapter 23. Expression Trees

**Draw expression trees for the following expressions. For unary operators, represent the operand as the left child of the node that contains the operator.**

1.  $100a + 10b + c$
2.  $(a + bi)(a - bi)$
3.  $2 / (1/x + 1/y)$
4. `!((whitespace || digit) && lowercase)`
5. `yr % 4 == 0 && (yr % 100 != 0 || yr % 400 == 0)`

**Choose the appropriate word:**

6. The process of converting a textual representation of an expression into a structured form (e.g., an expression tree) is called \_\_\_\_\_ (*evaluation / parsing / compilation*).
7. In an expression tree, operands are represented by (*names / parents / leaves*).
8. \_\_\_\_\_ (*Stack / Recursion / Traversal*) is the easiest way to evaluate an expression represented by an expression tree.
9. Conventional algebraic notation is called \_\_\_\_\_ (*prefix / infix / postfix*) notation.
10. RPN is another name for \_\_\_\_\_ (*prefix / infix / postfix*) notation.

**Mark true or false:**

11. The order of operands in prefix, postfix, and infix notations of the same expression is the same. \_\_\_\_\_
12. The last token in the postfix notation of an expression that contains binary operations must be an operation sign. \_\_\_\_\_
13. To convert an expression from postfix to prefix notation, you have to write the operands in the same order and the operation signs in reverse order. \_\_\_\_\_
14. Inorder traversal of an expression tree generates the expression in infix notation. \_\_\_\_\_
15. \_\_\_\_\_ expressions are convenient because they do not need any parentheses and can be evaluated with no regard to the precedence of operators. \_\_\_\_\_

**Evaluate the following expressions in prefix, postfix and infix notations:**

16.  $7 * (1 - 9) + (3 - 4)$  \_\_\_\_\_

17.  $2\ 5\ 3 - * 3 +$  \_\_\_\_\_

18.  $+++++1\ 1\ 2\ 3\ 5\ 8$  \_\_\_\_\_

19.  $!(true \ || \ (false \ \&\& \ !true))$  \_\_\_\_\_

20. Write and test a function

```
void PrintExpression(NODE *root);
```

that prints a parenthesized expression from the binary expression tree pointed to by `root`. Assume that operands and operators are represented by `apstring` tokens:

```
struct NODE {
    apstring token;
    NODE *left;
    NODE *right;
};
```

For unary operators, the operand is in the left child and there is no right child.

## Chapter 24. Heaps

Mark *true* or *false*:

1. A heap structure can help implement a priority queue. \_\_\_\_\_
2. A heap is a kind of binary search tree. \_\_\_\_\_
3. The most economical implementation of a heap is a linked binary tree with pointers from each node to its parent. \_\_\_\_\_
4. A full binary tree of depth  $h$  has  $2^h - 1$  nodes. \_\_\_\_\_

**Questions 5-9 refer to a heap implemented as an array  $x$ , where  $x[1]$  corresponds to the root of the heap. The heap contains  $N$  nodes.**

5. What is the subscript for the parent of  $x[i]$ ? \_\_\_\_\_
6. What are the subscripts for the left and right children of  $x[i]$ ?

\_\_\_\_\_

7. Write a condition for  $i$  that determines whether  $x[i]$  is a leaf.

\_\_\_\_\_

8. Write an expression for the depth of the heap.

\_\_\_\_\_

9. Write and test a function

```
void TraverseInOrder(const apvector<int> &x, int N);
```

that would traverse in order a heap of integers with  $N$  nodes and with the root in  $x[1]$ . Hint: use a recursive helper function with a third argument.



## Chapter 25. Analysis of Algorithms

Compare the order of growth of  $f(n)$  and  $g(n)$ . Write “ $f = g$ ”, “ $f < g$ ” or “ $f > g$ ”:

1.  $f(n) = n$        $g(n) = \frac{n^2 + 1}{n}$       \_\_\_\_\_

2.  $f(n) = 10^n$        $g(n) = 10^{2n}$       \_\_\_\_\_

3.  $f(n) = \frac{n(n+1)(n+2)}{6}$        $g(n) = n^2$       \_\_\_\_\_

4.  $f(n) = n!$        $g(n) = 2^n$       \_\_\_\_\_

5.  $f(n) = \sqrt{n^2 + 1}$        $g(n) = n$       \_\_\_\_\_

Mark true or false:

6. In comparing the order of growth,  $f = O(g)$  is like “ $f \leq g$ ” \_\_\_\_\_

7. We often informally say  $f = O(g)$  when we actually mean that the order of growth of  $f$  and  $g$  is the same. \_\_\_\_\_

8.  $\log_2 n = O(\log_{10} n)$ . \_\_\_\_\_

9.  $1 + 2 + \dots + n = O(n^2)$ . \_\_\_\_\_

10.  $(\log_2 n)^2 = O(n)$ . \_\_\_\_\_

11.  $1^k + 2^k + \dots + n^k = O(n^{k+1})$  for any positive integer  $k$ . \_\_\_\_\_

**What is the big-O of the number of operations required to perform the following tasks?**

12. Transposing a square matrix of size  $n$ : \_\_\_\_\_
13. Reversing an array of  $n$  elements: \_\_\_\_\_
14. Removing the first element from a queue with  $n$  nodes: \_\_\_\_\_
15. Removing the top element from a heap with  $n$  nodes: \_\_\_\_\_
16. Finding the number of pairs of consecutive double letters in a character string of length  $n$ : \_\_\_\_\_
17. Traversing inorder a binary search tree with  $n$  nodes: \_\_\_\_\_
18. Finding a node with a given value in a binary search tree with  $n$  nodes: \_\_\_\_\_
19.  $\blacksquare$  \_\_\_\_\_  
Generating all the Beavis Island words (see Question 7 for Chapter 0) of length  $n$ :  
\_\_\_\_\_

**20.** What is Big-O of the order of operations, in terms of  $n$ , in the following function:

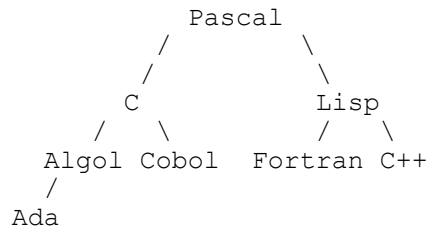
```
void LongToBin(unsigned long n, apvector<char> &binDigit)
// Converts unsigned long into an array of binary digits
{
    int i = 99;

    binDigit.resize(100);
    while (i >= 0 && n > 0) {
        if (n % 2) binDigit[i] = '1';
        else binDigit[i] = '0';
        i--;
        n /= 2;
    }
    while (i >= 0) {
        binDigit[i] = '0';
        i--;
    }
}
```

21.  $\blacklozenge$   $\blacksquare$  Look up the `ftime(...)` library function in the interactive help for your IDE. Write a program that uses `ftime(...)` to obtain benchmarks for the sequential and binary search algorithms. Take an array of  $n$  elements and initialize its values to 0, 1, ...,  $n-1$ . Run each type of search multiple times (e.g., 100,000 times) with a target randomly chosen from the set of values. Call `ftime` before and after, calculate the total elapsed time and the average search time, and tabulate your benchmarks for a few values of  $n$  (e.g.,  $n = 200$ ,  $n = 100$ ,  $n = 50$ ,  $n = 25$ ,  $n = 10$ ). Plot your benchmarks (average search time vs.  $n$ ) for sequential and binary searches. For which  $n$  does binary search outperform sequential search?

## Chapters 22-25 Review

Questions 1-6 refer to the following tree  $T$ :



1. What is the number of leaves in  $T$ ? \_\_\_\_\_
2. What is the depth of  $T$ ? \_\_\_\_\_
3. List the result of inorder traversal of  $T$ .

---

4. List the result of preorder traversal of  $T$ .

---

5. Swap two nodes in  $T$  to obtain a binary search tree (use alphabetical order, reading C++ as “Cplusplus”).
6. Swap two nodes in  $T$  to obtain a heap.

7.  Write and test a function

```
NODE *BST(const apvector<int> &x);
```

that takes elements from an array  $x$ , pre-sorted in ascending order, builds a binary search tree as close to being balanced as possible, and returns the pointer to the root of the new tree. Can you come up with an algorithm that works in  $O(n)$  time, where  $n$  is the size of the array?

Draw an expression tree that represents the following expressions, and convert each of them into RPN (use the  $\sim$  symbol for the unary – sign).

8.  $(x+y) * (x-y)$

---

9.  $(d + 31) \% 7$

---

10.  $(-b + \text{sqrt}(b*b - 4*a*c)) / (2*a)$

---

11.  $!p \ || \ !(q1 \ \&\& \ !q2)$

---

12.  $u \ >= \ v - 1 \ \&\& \ u \ <= \ v + 1$

---

13. In the usual representation of a complete binary tree in an array, does the array

```
pq[8] = {0, 13, 8, 5, 3, 2, 1, 1};
```

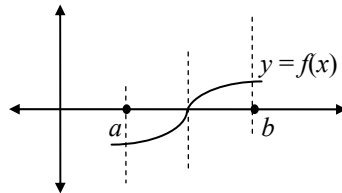
represent a heap?

14. A binary tree has five nodes containing 2, 3, 5, 7, and 11. In how many different shapes can such a tree be a heap and a binary search tree at the same time?

15. Define a `NODE` structure and a `LINKHEAP` class that represent a heap as a linked tree. Write and test the `Insert` and `Remove` functions. What is the Big-O performance for these functions in terms of the number of nodes  $n$  in the heap? (Hint: use recursive helper functions for the destructor and for `Insert` and `Remove`.)

## Chapter 26. Searching and Hashing

1. Describe the difference between searching and pattern recognition.
2. Describe a situation where the performance of a sequential search is better than  $O(n)$ .
3.  $\square$  A divide-and-conquer method can be used to find a zero of a function. Suppose a function  $f(x)$  is a continuous function on the interval  $[a, b]$ . Suppose  $f(a) < 0$  and  $f(b) > 0$ :



The graph of the function must cross the  $x$ -axis at some point. We can split the segment into two halves and continue our search for a zero in the left or the right half, depending on the value of  $f(x)$  in the middle point  $x = (a+b)/2$ .

Write a program that finds  $x$  (to the nearest .001), such that  $x = \cos(x)$ . (*Hint*: consider the function  $f(x) = x - \cos(x)$  on the interval  $[0, \pi/2]$ .)

4. In Scrabble, different letters are assigned different numbers of points:

|       |       |       |       |        |       |        |
|-------|-------|-------|-------|--------|-------|--------|
| a - 1 | e - 1 | i - 1 | m - 3 | q - 10 | u - 1 | x - 8  |
| b - 3 | f - 4 | j - 8 | n - 1 | r - 1  | v - 4 | y - 4  |
| c - 3 | g - 2 | k - 5 | o - 1 | s - 1  | w - 4 | z - 10 |
| d - 2 | h - 4 | l - 1 | p - 3 | t - 1  |       |        |

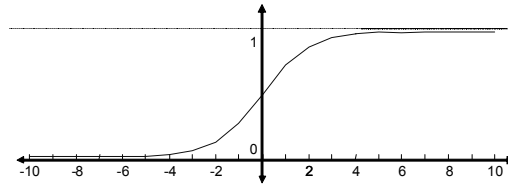
Write a function

```
int Score(const apstring &word);
```

that returns the score for a word without using either `if` or `switch` statements.

5. Neural networks are non-linear statistical models that learn from experience; they are used in pattern recognition systems. Neural network algorithms need to calculate the “sigmoid” function over and over again:

$$y = \frac{1}{1 + e^{-x}}$$



Write a function

```
void TabulateSigmoid(apvector<double> &s);
```

that tabulates the values of sigmoid for  $-10 \leq x \leq 10$  (with steps of 0.01) and places the values into the array *s*. Your function may call the library function `exp(x)` declared in `math.h`.

Write and test a function

```
double Sigmoid (double x, const apvector<double> &s);
```

that fetches and returns the sigmoid value for  $-10 \leq x \leq 10$  from the table. The function returns 0 if  $x < -10$ , and 1 if  $x > 10$ .

6. Define:

*hashing* \_\_\_\_\_

*hash function* \_\_\_\_\_

*collisions* \_\_\_\_\_

*chaining* \_\_\_\_\_

*bucket* \_\_\_\_\_

*probing* \_\_\_\_\_

*clustering* \_\_\_\_\_

Mark *true* or *false*:

7. Access time in a lookup table of length  $n$  is  $O(n)$ . \_\_\_\_\_
8. Probing is feasible only when the population of a hash table is relatively sparse. \_\_\_\_\_
9. Access time in a hash table is never greater than  $O(n)$ , where  $n$  is the size of the table. \_\_\_\_\_
10. It is easy to traverse a hash table in ascending order of keys. \_\_\_\_\_
11. The advantage of hash tables over binary search trees is the faster access time for adding and removing data elements. \_\_\_\_\_
  
12. ■ A hash table has sixty entries. Devise and test a hash function for English words such that all the different words from this paragraph are hashed into the table with no more than four collisions.



## Chapter 27. Sorting

Mark *true* or *false*:

1. If the original array was already sorted, 190 comparisons would be performed in a selection sort of an array containing 20 elements. \_\_\_\_\_
2. When insertion sort is applied to a singly linked list, the best case (that requires the smallest number of comparisons) is when the list is already sorted. \_\_\_\_\_
3. The mergesort procedure is more suitable for use with linked lists than with arrays. \_\_\_\_\_
4. Quicksort is sensitive to data; the performance is  $O(n \log n)$  only if most splits divide the array into two halves that are approximately equal in size. \_\_\_\_\_
5. Quicksort requires an auxiliary array that is as large as the original array. \_\_\_\_\_

Complete each sentence with the word *always*, *sometimes*, or *never*:

6. Selection sort in an array of  $n$  elements \_\_\_\_\_ works in  $O(n^2)$  time.
7. Bubble sort in an array of  $n$  elements \_\_\_\_\_ works in  $O(n)$  time.
8. Insertion sort \_\_\_\_\_ works faster than quicksort.
9. The mergesort algorithm is \_\_\_\_\_ implemented in C++ as a class.

10. An array of six integers — 6, 9, 74, 10, 22, 81 — is being sorted in ascending order. Show the state of the array after one pass through the array for each of the following methods:

a) bubble sort \_\_\_\_\_

b) selection sort (largest element) \_\_\_\_\_

c) insertion sort (insert from the beginning) \_\_\_\_\_

11. ■ Write and test a program that merges two sorted files, with about  $n$  records in each, into one sorted file. Your program should work with  $O(1)$  space and in  $O(n)$  time.

12. ■ What is the state of an array after the “split” phase of the quicksort algorithm is applied at the top level of recursion, if its initial values are

```
a[9] = {6, 9, 74, 10, 22, 81, 2, 11, 54};
```

and the middle element is chosen as a pivot?




## Chapter 28. Inheritance

Mark *true* or *false*:

1. A derived class always inherits all the data members and member functions of its base class. \_\_\_\_\_
2. The base class has to be defined (or `#include'd`) above the derived class. \_\_\_\_\_
3. The base class is a member of the derived class. \_\_\_\_\_
4. A derived class is a *friend* of the base class. \_\_\_\_\_
5. In object-oriented design, inheritance represents a relationship of a part to a whole. \_\_\_\_\_

Complete each sentence with the word *always*, *sometimes*, or *never*:

6. A private member of a base class is \_\_\_\_\_ private in the derived class.
7. Private and protected members of a base class \_\_\_\_\_ change their access properties in a derived class.
8. Public members of a base class \_\_\_\_\_ remain public in a publicly derived class.
9. A derived class \_\_\_\_\_ redefines member functions of its base class.
10. The constructor for a derived class \_\_\_\_\_ executes the constructor for the base class before entering its own code.
11.  Write and test a class `mystack<itemType>` by deriving it from the `apvector` class. Add the `mSp` data element. The `mystack` constructor `mystack(int n)` must use an initializer list to call the base class constructor with the same argument. It should also set `mSp` to 0. Redefine the `length()` member function to return the current number of elements on the stack. Add

```
void push(const itemType &item);
```

and

```
void pop(itemType &item);
```

member functions.

Mark *true* or *false*:

12. A base class pointer can point to an object of the derived class. \_\_\_\_\_
13. A derived class pointer can point to an object of the base class. \_\_\_\_\_
14. In OOP it is good practice to derive classes that represent various specialized objects of a certain category from one abstract class. \_\_\_\_\_
15. Polymorphism allows the pointer of the base class to call a member function of the derived class. \_\_\_\_\_
16. A member function that has polymorphic behavior must be declared `virtual`.  
\_\_\_\_\_
17. A class with one or more pure virtual member functions is called an *abstract* class. \_\_\_\_\_

## ◆◆ Projects

### 1. (1-D arrays)

A signal from an EKG monitor is digitized at the rate of 100 samples per second and stored in an array of amplitudes. Each heartbeat produces a spike in the amplitude of the signal that exceeds the mean value of other amplitudes within some surrounding window by at least a factor of 5. The window is defined by taking  $d$  amplitudes on each side of the spike. Write a function

```
int Pulse(const apvector<double> &ekg, int d);
```

that analyzes 15 seconds worth of digitized EKG and returns an estimated average pulse rate (heartbeats per minute). Your function will be used in a real-time patient monitoring system, so it has to be efficient. Do not use any nested loops or auxiliary arrays.

### 2. (2-D arrays)

A magic square is a square of size  $n$  that holds consecutive numbers from 1 to  $n^2$ , so that the sum of the numbers in each row, in each column, and in each of the two diagonals is the same. This is a magic square of size 3:

|   |   |   |
|---|---|---|
| 8 | 1 | 6 |
| 3 | 5 | 7 |
| 4 | 9 | 2 |

There is a simple method for generating a magic square when its size is an odd number. We place 1 in the middle of the top row and then shift diagonally: up and to the right, to place the next number. If we get out of bounds above the top row, we “wrap around” and continue with the corresponding cell in the bottom row. Likewise, if we get out of bounds on the right, we wrap around and continue with the corresponding cell in the left column. If the next cell is already filled, we instead shift down by one cell from the current cell.

Write a program that prompts the user for an odd integer  $n$  greater than or equal to 3 (but less than 20), validates the input, and prints out a magic square of size  $n$ .

**3.** (*Dynamic programming* technique)

The Number Cruncher drives along Interstate 911 crunching numbers placed every 10 yards along each lane. He may change lanes once after crunching a number. Given the table of numbers placed along the lanes, find the maximum possible total of the numbers he can crunch between Boston and Los Angeles.

In other words, the first  $k$  rows and  $n$  columns in a 2-D array are filled with numbers. Write a function

```
int OptimalPathTotal(apmatrix<int> &matrix);
```

that finds the maximum possible total of numbers along a path that starts in the first column and ends in the last column. The path crosses each column once and when we go from a column to the next, the path may only stay in the same row or go to the row immediately above or below it. For example:

```
1 3 2 2 3
4 1 1 2 4
  \     /
2 1 1 3 1
   \ /
2 2 4 1 2
```

The optimal sum is  $4+1+4+3+4 = 16$ . (Hint: your function is allowed to modify the original array.)

#### 4. (2-D arrays)

Write a program that traces the contour of a blob in an image. A black and white image can be represented as a 2-D array of bytes with 0 representing white, 1—black. A blob is a connected set of black pixels. Two pixels are considered connected if their sides or corners touch. Assume that the image contains only one blob. In the example below, 0's are shown as dots and 1's as 'x' for better readability:

```

.....
..xxxxxxxx.....
..xxxx.x.....
..xxxxxxxx.....
...xxx.x.x....
.....xxxx.....

```

The `TraceContour(...)` function finds the topmost black pixel in the blob and, starting at that point, traces the outer contour counterclockwise, saving the x-y coordinates of contour points. In narrow spots the same point may be traced twice if it is traversed in two opposite directions. For example, the above image has 26 points in the blob's outer contour. If (0,0) is the upper left corner, they are:

```

(3,1), (2,2), (2,3), (3,4), (4,4), (5,4), (6,3), (7,3),
(8,4), (8,5), (9,5), (10,5), (11,5), (11,4), (10,5), (9,5),
(8,4), (7,3), (7,2), (8,1), (9,1), (8,1), (7,1), (6,1),
(5,1), (4,1).

```

The function may save contour points in an array or a linked list of x-y coordinates and return the number of contour points. It should run in  $O(n)$  time, where  $n$  is the number of contour points.

#### 5. (Monte Carlo methods)

Write a program that uses a Monte Carlo simulation to estimate the probability of getting a blackjack on three cards pulled randomly out of a deck of 52 cards. A blackjack is a combination of cards totaling 21 points. An ace counts as either 1 or 11 points (player's choice); a king, queen, and jack score 10 points each; all other cards (10-2) score the face value of the card.

#### 6. (Strings)

Compile a small dictionary of two dozen words in a file and write a toy spellchecker that loads the dictionary file into an array of strings, then prompts the user to type in a word and verifies that it is in the dictionary or displays the closest candidates with the corrected spelling. Your spellchecker should catch words with a missing letter, a letter substitution, or two letters reversed.

**7.** (Strings)

In the Madlibs party game, the leader has the text of a short story with a few missing words in it. The missing words are tagged by their function: [noun], [verb], [place], etc. For example:

It was a [adjective] summer day.  
Jack was sitting in a [place].

The leader examines the text and prompts the players for the missing words:

Please give me an/a:  
adjective  
place  
...

He then reads the text with the supplied words inserted into their places.

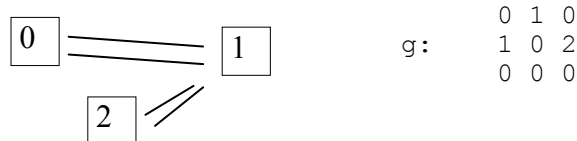
Write a program that acts as the Madlibs leader. It should prompt the user for a file name (or display a menu of available files), read the text from the chosen file, find the tags for the missing words (in square brackets), prompt the user for the words, and, finally, display the completed text.

**8.** (Linked lists)

Write a program that reads a text file and appends each word, together with the line number on which it occurs, to a linked list. Use a linked list with a tail. Assume that a word is any sequence of letters and that words do not split between lines. Skip all other characters. Convert all words to lower case. Sort the list using radix sort and print the alphabetized “index”: a word, followed by the list of all line numbers on which it occurs. (Do not repeat the same words or the same line numbers in the printout.)

## 9. (Recursion)

A directed graph is a diagram of  $N$  points connected by arrows (two points may be connected by more than one arrow). It is convenient to represent a directed graph as an  $N$  by  $N$  array (called connectivity matrix)  $g$ , in which  $g[i][j]$  is the number of arrows that go directly from point  $i$  to point  $j$ . For example:



Assume that the graph does not have circular paths. Write and test a (recursive) function:

```
bool HasPath(const amatrix<int> &g, int i, int j)

// g is the connectivity matrix of a directed graph:
// g[i][j] is the number of arrows going from point i to point j
//   (0 <= i,j < N).
// The graph does not have circular paths.
// Returns true if there is a continuous path (along arrows)
//   from node i to node j, false otherwise.

{
    ...
}
```

## 10. (Data representation and recursion)

Define a class `TicTacToe` that helps a computer program to play the game. The class should represent the current position, who plays x's and who plays o's, and whose turn it is to play. It should provide member functions to announce the computer's moves and to process the opponent's moves. It should also have a private member function that determines whether the current position is a winning position for the computer or its opponent or a tie, and another (recursive) member function that determines the computer's best move by analyzing the consequences of all possible responses. Write a program that plays Tic-Tac-Toe with the user.

**11.** (Data representation and recursion)

Write a program that solves the “IQ” puzzle. 14 pegs are arranged on a triangular board (with one empty hole):

```
      x
     x x
    x x x
   x x x x
  x x o x x
```

In a valid move, a peg jumps over a neighboring peg into an empty slot, and the jumped-over peg is removed. The objective is to have only one peg left after thirteen moves.

**12.** (Data representation, bit-wise logical operators, permutations)

A physics lab has six weights: they weigh 1, 2, 3, 4, 5, and 6 grams, and all look alike. The technician's daughter has been playing with the weights and might have switched the labels around. Design a test consisting of two weighings on a balance scale (with no other weights) that would determine whether any labels have been switched. Each weighing establishes which of the balance plates is heavier or confirms that the total weights on both sides are equal. (This problem has two solutions, one of which is rather hard to find without a computer.)

**13.** (Trees)

The nodes of a binary tree are implemented as:

```
struct NODE {
    int value;
    NODE *left;
    NODE *right;
};
```

(a) Write a function

```
NODE *NodeCountTree(NODE *root);
```

that takes a binary tree pointed to by `root` and builds a new tree. The shape of the new tree is exactly the same as the shape of the original tree, but the values in its nodes are different: in the new tree the value in each node is equal to the total number of nodes in the subtree rooted at that node. (For example, the value at the root is equal to the total number of nodes in the tree.) The function returns the pointer to the root of the new tree.

(b) Suppose you have a binary search tree (BST) and a companion “node count” tree (NCT) as described in (a). Write an efficient function

```
int Median(NODE *bst_root, NODE *nct_root);
```

that finds the median of the values stored in the BST in  $O(\log n)$  time (where  $n$  is the number of nodes in the tree).

Hints: (1) The median is the  $(n/2)$ -th value in the tree, in ascending order. Paradoxically, it is easier to write a more general function that finds the  $k$ -th value in the tree for any  $k$ . (2) Using either iterations or recursion, build parallel paths in the BST and the NCT from the root to the node you are looking for.

**14.** (Expression trees)

C++ has bit-wise logical operators `&` (and) and `|` (or) for Boolean operations on individual bits (see Appendix A in the textbook). `&` takes precedence over `|`. i86 assembly language has instructions `and` and `or` that correspond to these C++ operators. For example, the C++ statement

```
byte3 = byte1 & byte2;
```

could generate the following assembly language code:

```
mov  ax,byte1      ; move byte1 into the register ax
and  ax,byte2      ; "and" ax and byte2, put result in ax
mov  byte3,ax      ; move ax into byte3
```

The microprocessor has general-purpose registers `ax` and `bx` that can be used in `and` and `or` instructions with the second operand coming from memory (from a variable). It also supports a hardware stack with the `push` and `pop` instructions. For example:

```
push ax
pop  ax
```

Write and test a function that parses a C++ assignment statement

*Name = Expr;*

generates assembly-language code for that statement, and writes it to a file. *Expr* is a string of characters that contains only names of variables, parentheses, and `&` and `|` operators.

**15.** (Bit-wise logical operators)

Write a program that plays the Nim game defined in Question 42 for Chapter 1.

## Answers and Solutions

### Chapter 1

11. H 12. S 13. 8, 256 15. 4096 16. 4Gb 17. 40 20. 47, hex 2F 22. 193, hex C1  
24. 245, hex F5 26. 3851, hex 0F0B 28. 65280, hex FF00

30.

(a) 'A' 65, hex 41, 01000001 (b) 00100000, hex 20  
'a' 97, hex 61, 01100001  
'Q' 81, hex 51, 01010001  
'q' 113, hex 71, 01110001

33. -2 34. See part1\dict.dat 37. For example: 'x' = 11, 'o' = 10, blank = 00.

Each square needs 2 bits; 9 squares need 18 bits = 2.25 bytes. Yes.

38. 4 bits \* 512 \* 512 = 128K 39. mask: hex F8, stat. register: hex B0

40. (a) hex 19; (b) hex 03

41.

| base 3       | dec       | add 1 |
|--------------|-----------|-------|
| 21 02 10     | 7 2 3     | 8 3 4 |
| 00 11 22 ==> | 0 4 8 ==> | 1 5 9 |
| 12 20 01     | 5 6 1     | 6 7 2 |

42.

|        |                                                                                                        |
|--------|--------------------------------------------------------------------------------------------------------|
| 1: 001 | This is a "plus" position, because the count of 1's<br>in each column is even. The second player wins. |
| 3: 011 |                                                                                                        |
| 5: 101 |                                                                                                        |
| 7: 111 |                                                                                                        |

First takes 5 from 7:

|        |
|--------|
| 1: 001 |
| 3: 011 |
| 5: 101 |
| 7: 010 |

Second should take all from 5:

|     |
|-----|
| 001 |
| 011 |
| 000 |
| 010 |

### Chapter 2

2. F 5. F 7. F 8. F 9. int, return

10.

```
cout << firstName
    << ", congratulations on your first program!\n";
```

19. style 23. F 25. F 26. T 27. sometimes 28. never 29. sometimes

(e.g., in the Dictionary program, after struct ENTRY.)

30. sometimes (may come from a file). 32. compilation 33. read by

34. LoadDictionary(...) function 35. 2 43. valid 45. valid 47. valid 48. F (>> is the operator).

49. T 51. F (could go to a file) 52. D 54. E, A

56. Missing semicolon after iMin = i. Should return iMin.

### Chapter 3

2. 0 5. invalid 7. valid 9. 98 11. valid 12. valid 14. invalid 17. F 20. T 21. F

24. `int temp` should be `double temp`.

### Chapter 4

3. 0 6. 5 7. -5

8.

```
cout << "Seconds in a year = "  
      << double(hours) * mins * secs * days << endl;
```

9.

```
// 1./2 instead of 1/2:  
cout << "The travel distance is " << 1. / 2 * (g * t * t) << endl;
```

13. 3455 14. F (b has been incremented) 17. F (e.g., a = 6) 18. `int(x+.5)`

### Chapter 5

3. T 4. F 7. valid 10. valid

12.

```
apmatrix<int> chart(10, 4);
```

14.

```
...  
// Repeat as long as i is less than 16:  
while (i < 16) ...
```

### Chapters 1-5 Review

3. sometimes 4. sometimes 5. sometimes 6. Yes (1 bit for size, 5 bits for toppings) 8. T

11. T 13. invalid 16. invalid 18. invalid

20.

```
4 mi = 4 km
```

22.

```
c = 0
```

25. 8

### Chapter 6

2.

```
bool isdigit(char d)  
{  
    return d >= '0' && d <= '9';  
}
```

4.

```
...  
// halfHour = (mins % 30 == 0);  
halfHour = (mins % 60 == 30);
```

6. `bool inside = (x >= left && x <= right && y >= top && y <= bottom);`
8. `if (pixels[row][col] != pixels[row][col+1])  
count++;`
9. `if ((x + 2 > a || x - 2 < b) && y >= 0)`
14. Only (d)

## Chapter 7

1. 200 2. 246 3. D (count += 0.1 does not change count because it is int.)
7. `int i, n = v.length();  
for (i = 0; i < n; i++)  
v[i] = i+1;`
8. `...  
i = 1; // Not i = 0;  
while (i < 6) ...`
9. 3 1 4 1 12. Leading zeroes

## Chapter 8

2. `const int daysInMonth[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};  
days = daysInMonth[month-1];`
3. D

## Chapters 6-8 Review

1. Only (c)
6. `int i = 1;  
w.resize(n + n - 1);  
  
while (i <= n) {  
w[i-1] = i;  
w[2*n-1-i] = i;  
i++;  
}`

## Chapter 9

6.

```
bool IsPerfectSquare(int n)
{
    int p, sum = 0;
    for (p = 1; sum < n; p += 2)
        sum += p;
    return (sum == n);
}
```

8.

```
void SwapPosNeg(apvector<double> &v)
{
    int size = v.length(), i = 0, j = size - 1;
    double temp;

    while (i < j) {
        if (v[i] < 0)
            i++;
        else if (v[j] >= 0)
            j--;
        else { // if both out of place -- swap them
            temp = v[i]; v[i] = v[j]; v[j] = temp;
            i++;
            j--;
        }
    }
}
```

## Chapter 10

Several solutions in this chapter rely on the function

```
int random(int n);
```

that returns a random integer between 0 and n-1. This function, if not already provided in the standard library for your compiler, can be defined as:

```
int random(int n)
{
    return int(long(rand()) * n / (RAND_MAX + 1));
}
```

2.

```
#include <stdlib.h>

int TwoDice()
{
    return random(6) + random(6) + 2;
}
```

## Chapter 11

5. F 6. T 8. invalid 9. invalid 11. valid 12. valid 14. 30

19.

```
void SquareComplex(double &a, double &b)
{
    //    a = a*a - b*b; // "a" changes before it has been used
    //    b = 2*a*b;     //    in the next line.
    double a2 = a*a - b*b;
    b = 2*a*b;
    a = a2;
}
```

## Chapter 12

1. T 3. F (different data types) 4. Use "\\\" to indicate single backslash.

5. No. `strlwr(...)` actually changes the string to lower case.

7. valid 9. invalid

11. `name.length()` is more efficient because the length of the string is stored with the string. `strlen(name.c_str())` has to scan the string to find the terminating null.

## Chapters 9-12 Review

3.

```
void InsertMiddle(apvector<double> &v, double x)
{
    int i, len = v.length();

    v.resize(len + 1);
    for (i = len; i > len/2; i--)
        v[i] = v[i-1];
    v[i] = x;
}
```

8. no 10. -73

14.

```
apstring SwapFirstLast(const apstring &name)
{
    int pos = name.find(' ');
    return name.substr(pos+1, name.length() - pos - 1) +
           ", " +
           name.substr(0, pos);
}
```

## Chapter 13

3. T 5. T

7.

```
bk.chapters[2].title;
```

8.

```
bk.chapters[bk.nChapters-1].nPages;
bk.chapters[bk.nChapters-1].firstPageNo;
```

11.

```
ostream &operator<< (ostream &outp, const BOOK &book)
{
    outp << book.title << ": " << book.nChapters << " chapters, "
        << book.nPages << " pp.";
    return outp;
}
```

## Chapter 14

1. D 3. B 5. E 9. T 11. T 13. E

## Chapter 15

1. T 3. T 5. F 7. sometimes 9. always

10. sometimes (the whole class definition may be in the source file)

12. sometimes 15. goes out of scope 17. change 19. style 20. style 22. style

24. A (getX(...) needs a prototype)

25.

```
void POINT::MoveTo(double ax, double ay)
{
    // double x, y; this extraneous declaration shields class members x,y.
    ...
}
```

27. E 29. T 31. F 37. sometimes

## Chapter 16

2. T 3. T 5. F

7.

```
template <class ANYTYPE>
double Average(ANYTYPE a, ANYTYPE b)
{
    return (double(a) + double(b)) / 2;
}
```

11. Yes 12. No 13. Yes 14. No

17.

```
template <class COORD>
SEGMENT::Stretch3()
{
    COORD dx = x2 - x1;    // COORD xMid = (x1 + x2)/2;
    x1 -= dx;             // x1 = xMid + 3*(x1 - xMid);
    x2 += dx;             // x2 = xMid + 3*(x2 - xMid);
    COORD dy = y2 - y1;    // COORD yMid = (y1 + y2)/2;
    y1 -= dy;             // y1 = yMid + 3*(y1 - yMid);
    y2 += dy;             // y2 = yMid + 3*(y2 - yMid);
}
```

## Chapters 13-16 Review

1. No 3. Yes 5. Yes

7.

```
#include <iostream.h>
#include <iomanip.h>
...
ostream &operator<< (ostream &outFile, const STUDENT &s)
{
    outFile.setf(ios::left, ios::adjustfield); // Left justify
    outFile << setw(40) << s.name;
    outFile.setf(ios::right, ios::adjustfield);
    outFile.setf(ios::fixed | ios::showpoint);
    outFile << setprecision(2) << setw(6) << s.GPA << endl;
    return outFile;
}
```

9.C

11.

```
void NastyMessage()
{
    cout <<
#ifdef HITDISK
    "Please send $25.00 to have the data on your hard drive restored"
#else
    "Thank you for using X-SELL by Ransom Software"
#endif
    << endl;
}
```

14. sometimes 16. sometimes

18.

```
int main()
{
    LADDER ld(30);
    cout << ld.Add();
    return 0;
}
```

## Chapter 17

1. F 3. T 5. T 6. 0, &node3, &node2

7.

```
void Rotate (NODE* &head)
{
    if (!head || !head->next) return;

    // Save current head and adjust it
    NODE *temp = head;
    head = head->next;

    // Find the last node:
    NODE *tail = head;
    while (tail->next) tail = tail->next;

    // Append saved node to tail
    tail->next = temp;
    temp->next = 0;
}
```

12.

```
int Sum(NODE *head)
{
    if (!head) return 0;
    int sum = head->data;
    for (NODE *node = head->next; node != head; node = node->next)
        sum += node->data;
    return sum;
}
```

## Chapter 18

2. -1, -2, -6, 3, 7, 4 3. C 4. Pop cursor.x, cursor.y in reverse order.

## Chapter 19

1. D 2. D

4.

```
...
// Base case: when the list is empty, the for loop is not entered
for (NODE *node = picture; node; node = node->next) {
    Draw(node->elmt); // Draw the element
    DrawPicture(node->subpicture); // Draw the subpicture
}
...
```

5.  $\text{MystSum}(n) = n^2$ .

Proof:  $\text{MystSum}(1) = 1$ ; If  $\text{MystSum}(n-1) = (n-1)^2$  then  
 $\text{MystSum}(n) = (n-1)^2 + 2n - 1 = n^2 - 2n + 1 + 2n - 1 = n^2$ .

6.

```

long CountPaths(int x, int y)
{
    if (x <= 0 || y <= 0)
        return 1;
    else
        return CountPaths(x-1, y) + CountPaths(x, y-1);
}

```

## Chapter 20

2.

```

int RBQUEUE::Length()
{
    int len = rear - front;
    if (len < 0) len += BUFSIZE;
    return len;
}

```

3. Empty. Last 8 characters: d<SPACE>40:1a<CR>

## Chapters 17-20 Review

1.

```

NODE *MiddleNode(NODE *head)
{
    bool skip = false;
    NODE *node, *midnode = head;

    for (node = head; node; node = node->next) {
        if (!skip)
            midnode = midnode->next;
        skip = !skip;
    }
    return midnode;
}

```

4. E

8.

```

int GCF (int m, int n)
{
    int r = m % n;
    if (r == 0) return n;
    return GCF(n, r);
}

```

## Chapter 21

2. Copy constructor 3. Overloaded assignment operator  
 5. Constructor from compatible class 7. F 9. T 11. T 13. T 14. F  
 16. No (offset is a private member) 18. Yes

## Chapter 22

2. 17 3. 1, 7

4. Let  $N(h)$  be the max number of nodes for a binary tree of depth  $h$ .

$$N(1) \leq 2^1 - 1 = 1; N(h) \leq 1 + N(h-1) + N(h-1) \leq 1 + 2(2^{h-1} - 1) = 2^h - 1$$

5. If root is a leaf,  $L = 1, N = 1, l = (1+1)/2$ ;

$$L = L_{\text{left}} + L_{\text{right}} \leq (N_{\text{left}} + 1)/2 + (N_{\text{right}} + 1)/2 = (N_{\text{left}} + N_{\text{right}} + 2)/2 = (N + 1)/2$$

8. Leaves

11.

```

NODE *MirrorImage(NODE *root)
{
    if (!root) return 0;
    NODE *newroot = new NODE;
    newroot->data = root->data;
    newroot->left = MirrorImage(root->right);
    newroot->right = MirrorImage(root->left);
    return newroot;
}

```

14. F 16. T 17. T 18. F (e.g., a tree that degenerated into a list)

21.

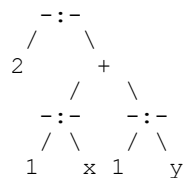
```

int MaxElement(NODE *root)
{
    while (root->right)
        root = root->right;
    return root->data;
}

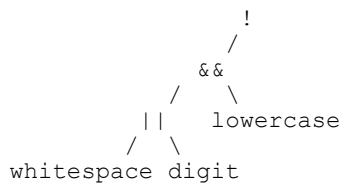
```

## Chapter 23

3.



4.



7. leaves 8. recursion 10. postfix 12. T 14. T 17. 7 19. false

## Chapter 24

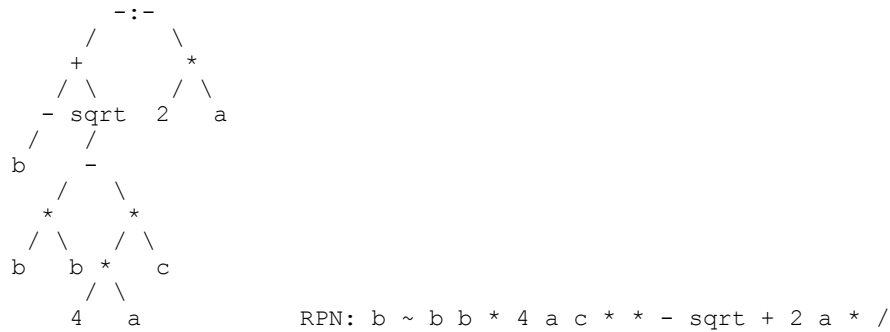
2. F 4. T 7.  $2^i > N$  8.  $\lceil \log_2 N \rceil + 1$

Chapter 25

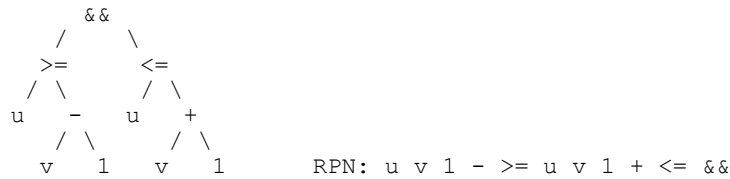
2.  $f < g$  4.  $f > g$  5.  $f = g$  7. T 9. T 11. T 14.  $O(1)$  15.  $O(\log n)$  18.  $O(\log n)$   
 19.  $O(3^n)$  20.  $O(\log n)$

Chapters 22-25 Review

2. 4 4. Pascal C Algol Ada Cobol Lisp Fortran C++ 6. C and C++ or C and Cobol  
 10.



12.



14. None

Chapter 26

2. The targets that are much more likely than others are placed at the beginning of the list  
 7. F 9. F (may have large buckets) 11. T

Chapter 27

1. T 3. F 5. F 7. sometimes 8. sometimes 9. sometimes  
 12. 6, 9, 11, 10, 2, 22, 81, 74, 54

Chapter 28

2. T 4. F 6. always 7. sometimes (in private inheritance) 9. sometimes

**11.**

```
// Change "private" to "protected" in apvector.h

template <class itemType>
class mystack : public apvector<itemType> {

public:
    mystack(int size) : apvector<itemType>(size), mSp(0) {}
    int length() {return mSp;}
    void push(const itemType &item) {
        if (mSp < mySize) myList[mSp++] = item;
    }
    void pop(itemType &item) {
        if (mSp > 0) item = myList[--mSp];
    }

private:
    int mSp;
};
```

**13. F 15. T 17. T**