Some Practice Problems for the C++ Exam

and Solutions for the Problems

The problems below are *not* intended to teach you how to program in C++. You should not attempt them until you believe you have mastered all the topics on the "Checklist" in the document entitled "Computer Science C++ Exam".

There are 39 problems. The solutions for the problems are given at the end, after the statement of problem 39.

1. What is the exact output of the program below? Indicate a blank space in the output by writing the symbol ␣ . Indicate a blank line in the output by writing blank line .

```
#include <iostream.h>

main()
{
   int n = 4, k = 2;

   cout << ++n << endl;
   cout <<  n  << endl;

   cout << n++ << endl;
   cout <<  n  << endl;

   cout << -n  << endl;
   cout <<  n  << endl;

   cout << --n << endl;
   cout <<  n  << endl;

   cout << n-- << endl;
   cout <<  n  << endl;

   cout << n + k << endl;
   cout <<   n   << endl;
   cout <<   k   << endl;

   cout << n << k << endl;

   cout <<          n << endl;
   cout << "  " << n << endl;

   cout << " n" << endl;
   cout << "\n" << endl;

   cout << " n * n = "; //CAREFUL!
   cout << n * n << endl;
```

```
    cout << 'n' << endl;

    return 0;
}
```

2. What is the output of the program below?

```
#include <iostream.h>

main()
{
   int n = 3;
   while (n >= 0)
   {
      cout << n * n << endl;
      --n;
   }

   cout << n << endl;

   while (n < 4)
      cout << ++n << endl;

   cout << n << endl;

   while (n >= 0)
      cout << (n /= 2) << endl;

   return 0;
}
```

3. What is the output of the program below?

```
#include <iostream.h>

main()
{
   int n;

   cout << (n = 4)  << endl;
   cout << (n == 4) << endl;
   cout << (n > 3)  << endl;
   cout << (n < 4)  << endl;
   cout << (n = 0)  << endl;
   cout << (n == 0) << endl;
   cout << (n > 0)  << endl;
   cout << (n && 4) << endl;
   cout << (n || 4) << endl;
   cout << (!n)     << endl;

   return 0;
}
```

4. What is the output of the following program?

```
#include <iostream.h>

main()
{
   enum color_type {red, orange, yellow, green, blue, violet};

   color_type shirt, pants;

   shirt = red;
   pants = blue;

   cout << shirt << "  " << pants << endl;

   return 0;
}
```

5. What is the output when the following code fragment is executed?

```
int i = 5, j = 6, k = 7, n = 3;
cout << i + j * k - k % n << endl;
cout << i / n << endl;
```

6. What is the output when the following code fragment is executed?

```
int found = 0, count = 5;
if (!found || --count == 0)
   cout << "danger" << endl;
cout << "count = " << count << endl;
```

7. What is the output when the following code fragment is executed?

```
char ch;
char title[] = "Titanic";

ch = title[1];
title[3] = ch;

cout << title << endl;
cout << ch << endl;
```

8. Suppose that the following code fragment is executed.

```
const int LENGTH = 21;
char  message[LENGTH];

cout << "Enter a sentence on the line below." << endl;
cin  >> message;

cout << message << endl;
```

Suppose that in response to the prompt, the interactive user types the following line and presses Enter:

```
Please go away.
```

What will the *output* of the code fragment look like?

9. Suppose that the following code fragment is executed.

```
const int LENGTH = 21;
char  message[LENGTH];

cout << "Enter a sentence on the line below." << endl;
cin.getline(message, LENGTH, '\n');

cout << message << endl;
```

a. Suppose that in response to the prompt, the interactive user types the following line and presses Enter:

```
Please go away.
```

What will the *output* of the code fragment look like?

b. Suppose that in response to the prompt, the interactive user types the following line and presses Enter:

```
Please stop bothering me.
```

What will the *output* of the code fragment look like?

10. Suppose that the following code fragment is executed.

```
const int LENGTH = 21;
char  message[LENGTH];

cout << "Enter a sentence on the line below." << endl;

int i = 0;

do
{
   cin >> message[i];
   ++i;
}
while (i < LENGTH - 1  &&  message[i] != '\n');

message[i] = '\0'; // Terminate string with NUL char.
```

```
cout << message << endl;
```

[continued on the next page]

a. Suppose that in response to the prompt, the interactive user types the following line and presses Enter:

```
Please go away.
```

What will the *output* of the code fragment look like?

b. Suppose that the statement "`cin >> message[i];`" is replaced by the statement

```
cin.get(message[i]);
```

Now what will the *output* of the code fragment look like if, in response to the prompt, the interactive user types the following line and presses Enter?

```
Please go away.
```

11. The nested conditional statement shown below has been written by an inexperienced C/C++ programmer. The behavior of the statement is not correctly represented by the formatting.

```
if (n < 10)
   if (n > 0)
      cout << "The number is positive." << endl;
else
   cout << "The number is _______________." << endl;
```

a. What is the output of the statement if the variable `n` has the value 7 ? If `n` has the value 15 ? If `n` has the value –3 ?

b. Correct the syntax of the statement so that the *logic* of the corrected statement corresponds to the *formatting* of the original statement. Also, replace the blank with an appropriate word or phrase.

c. Correct the formatting of the (original) statement so that the new format reflects the logical behavior of the original statement. Also, replace the blank with an appropriate word or phrase.

12. The loop shown below has been written by an inexperienced C/C++ programmer. The behavior of the loop is not correctly represented by the formatting.

```
int n = 10;

while (n > 0)
   n /= 2;
   cout << n * n << endl;
```

a. What is the output of the loop as it is written?

b. Correct the syntax of the loop so that the *logic* of the corrected loop corresponds to the *formatting* of the original loop. What is the output of the corrected loop?

c. Correct the formatting of the (original) loop so that the new format reflects the logical behavior of the original loop.

13. Remove all the unnecessary tests from the nested conditional statement below.

```
float income;

cout << "Enter your monthly income: ";
cin  >> income;

if (income < 0.0)
   cout << "You are going farther into debt every month." << endl;

else if (income >= 0.0  &&  income < 1200.00)
   cout << "You are living below the poverty line." << endl;

else if (income >= 1200.00  &&  income < 2500.00)
   cout << "You are living in moderate comfort." << endl;

else if (income >= 2500.00)
   cout << "You are well off." << endl;
```

14. Answer the questions below concerning the following fragment of code.

```
int n;

cout << "Enter an integer: ";
cin  >> n;

if (n < 10)
   cout << "less than 10" << endl;

else if (n > 5)
   cout << "greater than 5" << endl;

else
   cout << "not interesting" << endl;
```

a. What will be the output of the fragment above if the interactive user enters the integer value 0 ?

b. What will be the output of the fragment above if the interactive user enters the integer value 15 ?

c. What will be the output of the fragment above if the interactive user enters the integer value 7 ?

d. What values for `n` will cause the output of the fragment above to be `"not interesting"`?

15. Rewrite the following code fragment so that it uses a `"do...while..."` loop to accomplish the same task.

```
   int n;
   cout << "Enter a non-negative integer: ";
   cin  >> n;

   while (n < 0)
```

```
{
   cout << "The integer you entered is negative." << endl;
   cout << "Enter a non-negative integer: ";
   cin  >> n;
}
```

16. In the code fragment below, the programmer has almost certainly made an error in the first line of the conditional statement.

a. What is the output of this code fragment as it is written?

b. How can it be corrected to do what is the programmer surely intended?

```
int n = 5;

if (n = 0)  // NOTE THE OPERATOR!!!
   cout << "n is zero" << ".\n";
else
   cout << "n is not zero" << ".\n";

cout << "The square of n is " << n * n << ".\n";
```

17. What is the output when the following code fragment is executed?

```
int n, k = 5;
n = (100 % k ? k + 1 : k - 1);
cout << "n = " << n << "   k = " << k << endl;
```

18. What is the output when the following code fragment is executed?

```
int   n;
float x = 3.8;
n = int(x);
cout << "n = " << n << endl;
```

19. What is the output when the following code fragment is executed? Rewrite the fragment to obtain an equivalent code fragment in which the body of the loop is a simple statement instead of a compound statement.

```
int i = 5;
while (i > 0)
{
   --i;
   cout << i << endl;
}
```

20. The following loop is an *endless* loop: when executed it will never terminate. What modification can be made in the code to produce the desired output?

```
cout << "Here's a list of the ASCII values of all the upper"
     << " case letters.\n";

char letter = 'A';
```

```
while (letter <= 'Z')
   cout << letter << "   " << int(letter) << endl;
```

21. Write a function named "`sum_from_to`" that takes two integer arguments, call them "`first`" and "`last`", and returns as its value the sum of all the integers between `first` and `last` inclusive. Thus, for example,

```
cout << sum_from_to(4,7) << endl;   // will print 22 because 4+5+6+7 = 22
cout << sum_from_to(-3,1) << endl;  // will print −5 'cause (-3)+(-2)+(-1)+0+1 = −5
cout << sum_from_to(7,4) << endl;   // will print 22 because 7+6+5+4 = 22
cout << sum_from_to(9,9) << endl;   // will print 9
```

22. Write a function named "`enough`" that takes one integer argument, call it "`goal`" and returns as its value the smallest *positive* integer n for which 1+2+3+. . . +n is at least equal to `goal`. Thus, for example,

```
cout << enough(9) << endl;  // will print 4 because 1+2+3+4    9 but 1+2+3<9
cout << enough(21) << endl;// will print 6 'cause 1+2+ . . .+6    21 but 1+2+ . . . 5<21
cout << enough(-7) << endl;// will print 1 because 1    −7 and 1 is the smallest
                           //  positive integer
cout << enough(1) << endl;  // will print 1 because 1    1 and 1 is the smallest
                           //  positive integer
```

DEFINITION: A positive integer d is called a ***divisor*** of an integer n if and only if the remainder after n is divided by d is zero. In this case we also say that " d divides n ", or that " n is divisible by d ". Here are some examples:

-> 7 is a divisor of 35 ; that is, 35 is divisible by 7 .

-> 7 is a not a divisor of 27 ; that is, 27 is not divisible by 7 .

-> 1 is a divisor of 19 ; that is, 19 is divisible by 1 (in fact, 1 is a divisor of every integer n).

-> 12 is a divisor of 0 ; that is, 0 is divisible by 12 .

In C and C++ one can test the expression `n % d` to determine whether d is a divisor of n .

The ***greatest common divisor*** of a pair of integers m and n (not both zero) is the largest positive integer d that is a divisor of both m and n . We sometimes use the abbreviation "g.c.d." for "greatest common divisor. Here are some examples: 10 is the g.c.d. of 40 and 50 ; 12 is the g.c.d. of 84 and −132 ; 1 is the g.c.d. of 256 and 625 ; 6 is the g.c.d. of 6 and 42 ; 32 is the g.c.d. of 0 and 32 .

23. Write a function named "`g_c_d`" that takes two *positive* integer arguments and returns as its value the greatest common divisor of those two integers. If the function is passed an argument that is not positive (i.e., greater than zero), then the function should return the value 0 as a sentinel value to indicate that an error occurred. Thus, for example,

```
cout << g_c_d(40,50) << endl;    // will print 10
cout << g_c_d(256,625) << endl; // will print 1
cout << g_c_d(42,6) << endl;     // will print 6
```

```
cout << g_c_d(0,32) << endl;      // will print  0  (even though  32  is the g.c.d.)
cout << g_c_d(10,-6) << endl;     // will print  0  (even though  2  is the g.c.d.)
```

24. A positive integer n is said to be ***prime*** (or, "a prime") if and only if n is *greater than* 1 and is divisible only by 1 and n. For example, the integers 17 and 29 are prime, but 1 and 38 are not prime. Write a function named "`is_prime`" that takes a *positive* integer argument and returns as its value the integer 1 if the argument is prime and returns the integer 0 otherwise. Thus, for example,

```
cout << is_prime(19) << endl;    // will print 1
cout << is_prime(1) << endl;     // will print 0
cout << is_prime(51) << endl;    // will print 0
cout << is_prime(-13) << endl;   // will print 0
```

25. Write a function named "`digit_name`" that takes an integer argument in the range from 1 to 9, inclusive, and prints the English name for that integer on the computer screen. No newline character should be sent to the screen following the digit name. The function should not return a value. The cursor should remain on the same line as the name that has been printed. If the argument is not in the required range, then the function should print "`digit error`" without the quotation marks but followed by the newline character. Thus, for example,

the statement `digit_name(7);` should print `seven` on the screen;

the statement `digit_name(0);` should print `digit error` on the screen and place the cursor at the beginning of the next line.

26. Write a function named "`reduce`" that takes two positive integer arguments, call them "`num`" and "`denom`", treats them as the numerator and denominator of a fraction, and reduces the fraction. That is to say, each of the two arguments will be modified by dividing it by the greatest common divisor of the two integers. The function should return the value 0 (to indicate failure to reduce) if either of the two arguments is zero or negative, and should return the value 1 otherwise. Thus, for example, if `m` and `n` have been declared to be integer variables in a program, then

```
m = 25;
n = 15;
if (reduce(m,n))
   cout << m << '/' << n << endl;
else
   cout << "fraction error" << endl;
```

will produce the following output:

```
5/3
```

Note that the values of m and n were modified by the function call. Similarly,

```
m = 63;
n = 210;
if (reduce(m,n))
   cout << m << '/' << n << endl;
else
   cout << "fraction error" << endl;
```

will produce the following output:

```
3/10
```

Here is another example.

```
m = 25;
n = 0;
if (reduce(m,n))
   cout << m << '/' << n << endl;
else
   cout << "fraction error" << endl;
```

will produce the following output:

```
fraction error
```

The function `reduce` is allowed to make calls to other functions that you have written.

27. Write a function named `"swap_floats"` that takes two floating point arguments and interchanges the values that are stored in those arguments. The function should return no value. To take an example, if the following code fragment is executed

```
float x = 5.8, y = 0.9;
swap_floats (x, y);
cout << x << "   " << y << endl;
```

then the output will be

```
0.9   5.8
```

28. Write a function named `"sort3"` that takes three floating point arguments, call them `"x"`, `"y"`, and `"z"`, and modifies their values, if necessary, in such a way as to make true the following inequalities: x y z. The function should return no value. To take an example, if the following code fragment is executed

```
float a = 3.2, b = 5.8, c = 0.9;
sort3 (a, b, c);
cout << a << "   " << b << "   " << c << endl;
```

then the output will be

```
0.9   3.2   5.8
```

The function `sort3` is allowed to make calls to other functions that you have written.

29. Write a function named "`reverse`" that takes as its arguments the following:

(1) an array of floating point values;

(2) an integer that tells how many floating point values are in the array.

The function must reverse the order of the values in the array. Thus, for example, if the array that's passed to the function looks like this:

0	1	2	3	4
5.8	2.6	9.0	3.4	7.1

then when the function returns, the array will have been modified so that it looks like this:

0	1	2	3	4
7.1	3.4	9.0	2.6	5.8

The function should not return any value.

30. Write a function named "`sum`" that takes as its arguments the following:

(1) an array of floating point values;

(2) an integer that tells how many floating point values are in the array.

The function should return as its value the sum of the floating point values in the array. Thus, for example, if the array that's passed to the function looks like this:

0	1	2	3	4
5.8	2.6	9.0	3.4	7.1

then the function should return the value 27.9 as its value.

31. Write a function named "`location_of_largest`" that takes as its arguments the following:

(1) an array of integer values;

(2) an integer that tells how many integer values are in the array.

The function should return as its value the subscript of the cell containing the largest of the values in the array. Thus, for example, if the array that's passed to the function looks like this:

0	1	2	3	4
58	26	90	34	71

then the function should return the integer 2 as its value. If there is more than one cell containing the largest of the values in the array, then the function should return the *smallest* of the subscripts of the cells containing the largest values. For example, if the array that's passed to the function is

0	1	2	3	4	5	6
58	26	91	34	70	91	88

then the largest value occurs in cells 2 and 5 , so the function should return the integer value 2 .

32. Write a function named "`location_of_target`" that takes as its arguments the following:
(1) an array of integer values;
(2) an integer that tells how many integer values are in the array;
(3) an integer "target value".
The function should determine whether the given target value occurs in any of the cells of the array, and if it does, the function should return the subscript of the cell containing the target value. If more than one of the cells contains the target value, then the function should return the largest subscript of the cells that contain the target value. If the target value does not occur in any of the cells, then the function should return the sentinel value -1. Thus, for example, if the target value that's passed to the function is 34 and the array that's passed to the function looks like this:

```
  0    1    2    3    4    5    6
 58 | 26 | 91 | 34 | 70 | 34 | 88
```

then the target value occurs in cells 3 and 5, so the function should return the integer value 5.

33. Write a function named "`rotate_right`" that takes as its arguments the following:
(1) an array of floating point values;
(2) an integer that tells the number of cells in the array;
The function should shift the contents of each cell one place to the right, except for the contents of the last cell, which should be moved into the cell with subscript 0. Thus, for example, if the array passed to the function looks like this:

```
  0     1     2     3     4
 5.8 | 2.6 | 9.1 | 3.4 | 7.0
```

then when the function returns, the array will have been changed so that it looks like this:

```
  0     1     2     3     4
 7.0 | 5.8 | 2.6 | 9.1 | 3.4
```

The function should not return a value.

34. Write a function named "`shift_right`" that takes as its arguments the following:
(1) an array of floating point values;
(2) an integer, call it "`left`", that tells the leftmost cell of the part of the array to be shifted;
(3) an integer, call it "`right`", that tells the rightmost cell of the part of the array to be shifted;
(4) a positive integer, call it "`distance`" that tells how many cells to shift by.
The function should make sure that `left` is less than or equal to `right`, and that `distance` is greater than zero. If either of these conditions fails, the function should return the value 1 to indicate an error. Otherwise it should shift by `distance` cells the contents of the array cells with subscripts running from `left` to `right`. Thus, for example, if the array passed to the function looks like this:

```
  0     1     2     3     4     5     6     7     8     9    10   ....
 5.8 | 2.6 | 9.1 | 3.4 | 7.0 | 5.1 | 8.8 | 0.3 | -4.1 | 8.0 | 2.7 | etc.
```

and if `left` has the value 3, `right` has the value 7, and `distance` has the value 2, then the function should shift the contents of cells 3, 4, 5, 6, and 7 to the right by 2 cells, so that when the function returns, the array will have been changed so that it looks like this:

```
  0     1     2     3     4     5     6     7     8     9     10    ....
| 5.8 | 2.6 | 9.1 | ??? | ??? | 3.4 | 7.0 | 5.1 | 8.8 | 0.3 | 2.7 | etc.
```

The question marks in cells 3 and 4 indicate that we don't care what numbers are in those cells when the function returns. Note that the contents of cells 8 and 9 have changed, but the contents of cell 10 is unchanged. The function need not take any precautions against the possibility that the cells will be shifted beyond the end of the array (the calling function should be careful not to let that happen).

35. Write a function named "`subtotal`" takes as its arguments the following:
(1) an array of floating point values;
(2) an integer that tells the number of cells in the array.
The function should replace the contents of each cell with the sum of the contents of all the cells in the original array from the left end to the cell in question. Thus, for example, if the array passed to the function looks like this:

```
  0     1     2     3     4
| 5.8 | 2.6 | 9.1 | 3.4 | 7.0
```

then when the function returns, the array will have been changed so that it looks like this:

```
  0     1     2      3      4
| 5.8 | 8.4 | 17.5 | 20.9 | 27.9
```

because 5.8 + 2.6 = 8.4 and 5.8 + 2.6 + 9.1 = 17.5 and so on. Note that the contents of cell 0 are not changed. The function should not return a value.

36. Write a function named "`concatenate`" that copies the cells of one array into a larger array, and then copies the cells of another array into the larger array just beyond the contents of the first array. The contents of the cells will be integers. The arguments will be as follows:
(1) the first array that will be copied;
(2) the number of cells that will be copied from the first array;
(3) the second array that will be copied;
(4) the number of cells that will be copied from the second array;
(5) the large array into which all copying will be performed;
(6) the number of cells available in the large array.
If the function discovers that the number of cells in the large array is not large enough to hold all the numbers to be copied into it, then the function should return 0 to indicate failure. Otherwise it should return 1. The function should not alter the contents of the first two arrays. To take an example, if the first two arrays passed to the function look like this:

```
  0     1     2     3     4     5     6                    0     1     2     3
```

58	26	91	34	70	34	88

and

29	41	10	66

then, provided the size of the large array is at least 11, the large array should look like this when the function returns:

0	1	2	3	4	5	6	7	8	9	10
58	26	91	34	70	34	88	29	41	10	66

37. Write a function named "`number_of_matches`" that compares the initial parts of two character arrays to see how many pairs of cells match before a difference occurs. For example, if the arrays are

0	1	2	3	4	5
Error!					

and

0	1	2	3	4
Error!				

then the function should return the value 3 because only the first three pairs of cells in the arrays match (cell 0 matches cell 0, cell 1 matches cell 1, and cell 2 matches cell 2). Each of the character arrays will end with the character whose ASCII value is zero; this character, called NUL, is denoted by `'\0'` in the C and C++ programming languages (see the two arrays shown above). The pairwise cell comparisons should not go beyond the end of either array. If the two arrays are identical all the way to their terminating NUL characters, return the number of non-NUL characters. The function should take only two parameters, namely the two character arrays to be compared.

38. Write a function named "`eliminate_duplicates`" that takes an array of integers in random order and eliminates all the duplicate integers in the array. The function should take two arguments:
(1) an array of integers;
(2) an integer that tells the number of cells in the array.
The function should not return a value, but if any duplicate integers are eliminated, then the function should change the value of the argument that was passed to it so that the new value tells the number of distinct integers in the array. Here is an example. Suppose the array passed to the function is as shown below, and the integer passed as an argument to the function is 11.

0	1	2	3	4	5	6	7	8	9	10
58	26	91	26	70	70	91	58	58	58	66

Then the function should alter the array so that it looks like this:

0	1	2	3	4	5	6	7	8	9	10
58	26	91	70	66	??	??	??	??	??	??

and it should change the value of the argument so that it is 5 instead of 11 . The question marks in the cells after the 5th cell indicate that it does not matter what numbers are in those cells when the function returns.

39. Write an entire C++ program that reads a positive integer entered by an interactive user and then prints out all the positive divisors of that integer in a column and in decreasing order. The program should allow the user to repeat this process as many times as the user likes. Initially, the program should inform the user about how the program will behave. Then the program should prompt the user for each integer that the user wishes to enter.

The program may be terminated in any of two ways. One way is to have the program halt if the user enters an integer that's negative or zero. In this case the user should be reminded with each prompt that the program can be terminated in that way. Alternatively, after an integer has been entered and the divisors have been printed, the program can ask the user whether he/she wishes to enter another integer. In this case, when the user accidentally enters a zero or negative integer to have its divisors calculated, the program should inform the user that the input is unacceptable and should allow the user to try again (and again!).

Here is an illustration of how the program and the interactive user might interact. The user's responses to the program are shown in bold italics.

```
   This program is designed to exhibit the positive divisors of
   positive integers supplied by you.  The program will repeatedly
   prompt you to enter a positive integer.  Each time you enter a
   positive integer, the program will print all the divisors of
your
   integer in a column and in decreasing order.

   Please enter a positive integer: 36
   36
   18
   12
   9
   6
   4
   3
   2
   1

   Would you like to see the divisors of another integer (Y/N)? y
   Please enter a positive integer: -44

   -44 is not a positive integer.
   Please enter a positive integer: 0

   0 is not a positive integer.
   Please enter a positive integer: 109
   109
   1

   Would you like to see the divisors of another integer (Y/N)? m

   Please respond with Y (or y) for yes and N (or n) for no.
```

```
Would you like to see the divisors of another integer (Y/N)? n
```

Answers

1. The output would be as shown below.

```
5
5
5
6
-6
6
5
5
5
4
6
4
2
42
4
□□ 4
□ n
blank line
blank line
□ n * n = 16
n
```

2. The output would be as shown below. *The program contains an endless loop*.

```
9
4
1
0
-1
0
1
2
3
4
4
2
1
0
0 [endlessly]
```

3. The output would be as shown below.

```
4
1
1
0
0
```

1
0
0
1
1

4. The output would be as shown below.

```
0   4
```

5. The output would be as shown below.

```
46
1
```

6. The output would be as shown below, because when it is discovered that `"!found"` is true, the second half of the OR expression is not evaluated, and thus `count` is not decremented.

```
danger
count = 5
```

7. The output would be as shown below. The 'a' in "Titanic" has been changed to an 'i'.

```
Titinic
i
```

8. The output would be as shown below. The line in italics would be typed by the interactive user.

```
Enter a sentence on the line below.
Please go away.
Please
```

The reason that only the word "Please" is picked up in the `message` array is that the extraction operator `>>` ignores leading white space characters and then reads non-white space characters up to the next white space character. That is, `>>` is "word oriented", not line oriented.

9 a. The output would be as shown below. The line in italics would be typed by the interactive user.

```
Enter a sentence on the line below.
Please go away.
Please go away.
```

9 b. The output would be as shown below. The line in italics would be typed by the interactive user.

```
Enter a sentence on the line below.
Please stop bothering me.
Please stop botherin
```

The reason that the entire line typed by the user is not picked up in the `message` array is that the `getline` instruction will read at most 20 characters from the keyboard.

10 a. If the interactive user types only `Please go away.` and presses the Enter key, then there will be no output; instead, the program will "hang" and wait for more input. The reason for this is that the instruction "`cin >> message[i];`" removes and discards each white space character that it encounters in the input stream. (White space characters include the blank character and the newline character as well as several other control characters.) Thus the loop will discard the two blanks in the input and the newline at the end, producing this condition in the input array:

0	1	2	3	4	5	6	7	8	9	10	11	12
'P'	'l'	'e'	'a'	's'	'e'	'g'	'o'	'a'	'w'	'a'	'y'	'.'

Since `message[i]` will never pick up the newline character, the only way that the loop can end is for the loop variable `i` to reach the value 20. This cannot happen unless the interactive user types at least eight more non-white space characters.

b. The instruction "`cin.get(message[i];`" inputs into the array whatever character it finds in the input stream, including all blank space characters. Thus if the user types `Please go away.` and presses the Enter key, then the loop will end when `message[15]` gets the newline character, and the output will be exactly the same as the input.

11 a. The original statement is formatted in such a way that it *appears* that the "`else`" clause of the statement is the alternative to the "`if (n < 10)`" case, but in fact, the C or C++ compiler will treat the "`else`" clause as the alternative to the "`if (n > 0)`" clause. If `n` has the value 7 , the output will be "`The number is positive`". If `n` has the value 15 , then there will be no output. If `n` has the value −3 then the output will be "`The number is _____`".

b. If we want the statement to behave according the logic that's suggested by the formatting of the original statement, we can write

```
if (n < 10)
{
   if (n > 0)
      cout << "The number is positive." << endl;
}
else
   cout << "The number is greater than 10." << endl;
```

c. If we want the statement to be formatted so as to reflect the actual logic of the original statement, we can write

```
if (n < 10)
   if (n > 0)
      cout << "The number is positive." << endl;
   else
      cout << "The number is negative." << endl;
```

12 a. Since there are no braces surrounding the last two lines of the code, the compiler treats only the statement "`n /= 2;`" as the body of the loop. Thus n will successively assume the values 5, 2, 1, and 0, at which point the loop will exit and the "`cout`" statement will print 0. The values 5, 2, and 1 will not be printed.

b. If we want the statement to behave according the logic that's suggested by the formatting of the original statement, we can put braces around the last two lines of code to make a compound statement as the body of the loop:

```
int n = 10;

while (n > 0)
{
   n /= 2;
   cout << n * n << endl;
}
```

In this case the output of the loop will be

```
25
4
1
0
```

c. If we want the statement to be formatted so as to reflect the actual logic of the original statement, we can write

```
int n = 10;

while (n > 0)
   n /= 2;

cout << n * n << endl;
```

13. The conditional statement should be modified as follows:

```
if (income < 0.0)
   cout << "You are going farther into debt every month." << endl;

else if (income < 1200.00)
   cout << "You are living below the poverty line." << endl;

else if (income < 2500.00)
   cout << "You are living in moderate comfort." << endl;

else
   cout << "You are well off." << endl;
```

14 a. The output will be "`less than 10`".

b. The output will be "`greater than 15`".

c. The output will be "`less than 10`".

d. There is no value for `n` that will cause the output to be `"not interesting"`. That part of the code can never be executed!

15. Using `do...while...` in this situation makes the code a little simpler.

```
int n;

do
{
   cout << "Enter a non-negative integer: ";
   cin  >> n;

   if (n < 0)
      cout << "The integer you entered is negative." << endl;
}
while (n < 0);
```

16 a. The output of the code fragment as it is written will be

```
n is not zero.
The square of n is 0.
```

The reason for this is that the "`if`" part *assigns* the value `0` to `n`, and the value returned by that assignment statement is `0`, so this is treated as "false", which causes the alternative statement to be executed. But even though the program prints "`n is not zero.`", in fact, `n` *does* have the value zero after the conditional statement is finished.

b. The correction consists simply of replacing "`n = 0`" by "`n == 0`".

17. The output of the code fragment is as follows:

```
n = 4   k = 5
```

18. The output of the code fragment is as follows:

```
n = 3
```

The fractional part of the floating point value is discarded when the value is cast to integer type.

20. The loop can be modified as follows:

```
while (letter <= 'Z')
{
   cout << letter << "  " << int(letter) << endl;
   ++letter;
}
```

21.

```
/******************  S U M   F R O M   T O
**********************

DESCRIPTION:  Computes and returns the sum of all the integers
              between "first" and "last" inclusive.

PARAMETERS:

  first, last The two "endpoints" of the sequence of integers to be
              summed.

RETURNS:      Returns the sum of all the integers from "first" to
              "last".  For example, if first is 9 and last is 12
              then the function will return 42 ( = 9+10+11+12).
              If first is 11 and last is 8, then the function
              will return 38 ( = 11+10+9+8).  If first is 5 and
              last is 5, the function will return 5.

ALGORITHM:    If first <= last, the addition begins at first and
              goes up to last.  If, instead, first > last, then the
              addition begins at first and goes down to last.

AUTHOR:       W. Knight

*************************************************************
/

int sum_from_to (int first, int last)
{
   int i, partial_sum = 0;

   if (first <= last)
      for (i = first; i <= last; ++i)
         partial_sum += i;

   else
      for (i = first; i >= last; --i)
         partial_sum += i;

   return partial_sum;
}
```

22.

```
/************************  E N O U G H  ***********************

DESCRIPTION:  Computes and returns the smallest positive integer n
              for which 1+2+3+...+n equals or exceeds the value of
              "goal".

PARAMETER:

  goal        The integer which 1+2+3+...+n is required to meet or
              exceed.

RETURNS:      Returns the smallest positive integer n  for which
              1+2+3+...+n equals or exceeds "goal".  For example,
if
              goal has the value 9, then the function will return 4
              because 1+2+3+4 >= 9 but 1+2+3 < 9.  If goal has a
value
              less than or equal to 1, then the function will
return
              the value 1.

ALGORITHM:    First the value n = 1 is tried to see if 1 >= goal.
              If that is not true, then successively larger values
              of n are added to a summing variable until that sum
              reaches or exceeds goal.

AUTHOR:       W. Knight

************************************************************
/

int enough (int goal)
{
   int n = 1, sum = 1;

   while (sum < goal)
      sum += ++n;

   return n;
}
```

23. There is a well-known algorithm called "Euclid's Algorithm" for computing the g.c.d. of two positive integers. The second of the two solutions below employs that algorithm. The first solution employs a somewhat more straight-forward search process, in which we simply try one integer after another, starting with the smaller of the two arguments, until we find an integer that divides both. For the purposes of the C++ exam, the first solution is acceptable (even though it is far less efficient than the solution that uses Euclid's Algorithm).

```
/**********************  G_C_D (VERSION 1)
************************

DESCRIPTION:  Computes and returns the greatest common divisor
              (g.c.d.) of the arguments passed to it.

PARAMETERS:

  a , b       The integers whose g.c.d. will be computed.

RETURNS:      If either argument is less than or equal to zero, the
              value zero will be returned as a sentinel to indicate
              an error.  If both arguments are strictly positive,
              then their g.c.d. will be returned.  Thus, for
              example, if parameter a has the value 28 and b has
the
              value 70, then the function will return the value 14.

ALGORITHM:    If both arguments are positive, then the smaller of
the
              two arguments is tested to see whether it is a
divisor
              of both arguments.  If it is not, then successively
              smaller integers are tried.  If no common divisor
larger
              than 1 is found, then the loop will automatically
stop
              with trial_divisor having the value 1 because 1 is a
              divisor of every positive integer.

AUTHOR:       W. Knight

************************************************************
/

int g_c_d (int a, int b)
{
   if (a <= 0  ||  b <= 0)  // a parameter is not positive
      return 0;             // exit and return the error sentinel

   int trial_divisor;
```

```
   trial_divisor = ( a <= b  ?  a  :  b ); // set it to the smaller

   while (a % trial_divisor != 0  ||  b % trial_divisor != 0)
      --trial_divisor;

   return trial_divisor;
}
```

[A version that uses the Euclidean algorithm is given on the following page.]

```
/**********************  G_C_D (VERSION 2)
************************

DESCRIPTION:  Computes and returns the greatest common divisor
              (g.c.d.) of the arguments passed to it.

PARAMETERS:

  a , b       The integers whose g.c.d. will be computed.

RETURNS:      If either argument is less than or equal to zero, the
              value zero will be returned as a sentinel to indicate
              an error.  If both arguments are strictly positive,
              then their g.c.d. will be returned.  Thus, for
              example, if parameter a has the value 28 and b has
the
              value 70, then the function will return the value 14.

ALGORITHM:    If both arguments are positive, then Euclid's
algorithm
              for the g.c.d. is employed.  The first integer is
divided
              by the second, and then the second is divided by the
              remainder, and then the first remainder is divided by
the
              second, and so on until a remainder of 0 is obtained.
              The g.c.d. will be the divisor that produced 0
remainder.

AUTHOR:       W. Knight

************************************************************************
/

int g_c_d (int a, int b)
{
   if (a <= 0  ||  b <= 0)  // a parameter is not positive
      return 0;             // exit and return the error sentinel

   int remainder = a % b;   // Get remainder when a is divided by
b.

   while (remainder != 0)
   {
      a = b;
      b = remainder;
      remainder = a % b;
   }
```

```
    return b;  // Return the divisor that produced a remainder of 0.
}
```

24.

```
/************************  I S   P R I M E
***********************

DESCRIPTION:  Determines whether an integer is prime.

PARAMETER:

  n           The integer to be examined to see whether it is
prime.

RETURNS:      1 if the integer n is prime;  otherwise it returns 0.

ALGORITHM:    If n is less than or equal to 1, then it is not
prime.
              If n is greater than 1, then trial divisors, starting
              with 2 and going no farther than n-1 are examined to
              determine whether they divide n.  If no divisor less
              than n is found, then n is prime.

AUTHOR:       W. Knight

*************************************************************
/


int is_prime (int n)
{
   if (n <= 1)
      return 0; // n cannot be prime if n <= 1.

   int trial_divisor = 2;

   while (trial_divisor < n  &&  n % trial_divisor != 0)
      ++trial_divisor;

   // When the loop exits, one of two conditions must be satisfied:
   // either trial_divisor will have reached the value  n -- which
   // means that  n  is prime or else  n  will be divisible by
   // trial_divisor, in which case  n  will not be prime.  The
   // only exception to this is when n is 2, in which case n is
prime.

   if (trial_divisor == n) // n must be prime
      return 1;
   else
      return 0; // n is not prime.
}
```

25.

```
/*********************  D I G I T   N A M E
**********************

DESCRIPTION:  Prints the English name of an integer from 1 to 9.

PARAMETER:

  n           The integer whose English name will be printed.

RETURNS:      Void (no value).

ALGORITHM:    A switch statement selects the appropriate word.
              If "n" is not in the range 1,2,3,...,9, then an
              error phrase is printed.

AUTHOR:       W. Knight

*************************************************************
/


void digit_name (int digit_value)
{
   switch (digit_value)
   {
      case 1  : cout << "one";    break;
      case 2  : cout << "two";    break;
      case 3  : cout << "three";  break;
      case 4  : cout << "four";   break;
      case 5  : cout << "five";   break;
      case 6  : cout << "six";    break;
      case 7  : cout << "seven";  break;
      case 8  : cout << "eight";  break;
      case 9  : cout << "nine";   break;
      default : cout << "digit error" << endl;
   }
}
```

26.

```
/************************  R E D U C E  ************************

DESCRIPTION:  Reduces a positive fraction to lowest terms.

PARAMETERS:

  num, denom  The numerator and denominator of the fraction to be
              reduced.  These are reference parameters, so the
arguments
              passed to them may be changed by the function.

RETURNS:      1 if the arguments are both positive, 0 otherwise.

ALGORITHM:    If both arguments are positive, then each of them is
              divided by their greatest common divisor.

AUTHOR:       W. Knight

*****************************************************************
/

int reduce (int & num, int & denom)
{
   if (num <= 0  ||  denom <= 0)
      return 1;

   else
   {
      int common = g_c_d (num, denom);
      num   /= common;
      denom /= common;
   }
}
```

27.

```
/*********************  S W A P   F L O A T S
*********************

DESCRIPTION:  Interchanges the values of the two floating point
              variables passed to it.

PARAMETERS:

  a, b        The two parameters whose values will be interchanged.
              These are both reference parameters, so their values
              may be changed by the function.

RETURNS:      Void (no value).

ALGORITHM:    Stores the value of parameter a in a temporary
location,
              then copies the value in b into a, and finally copies
the
              stored original value of a into b.

AUTHOR:       W. Knight

*****************************************************************
/

void swap_floats (float & a, float & b)
{
   float temp = a;
   a = b;
   b = temp;
}
```

28. Here is one solution among many possible. It uses the `swap_floats` function of problem 27.

```
/*****************************  S O R T 3
*****************************
DESCRIPTION:  Sorts the three values passed to it into increasing
order.

PARAMETERS:
  x, y, z     The three floating point numbers to be sorted.
              All three parameters are reference parameters, so the
              arguments passed to them may be changed by the
function.

RETURNS:      Void (no value).

ALGORITHM:    First x, y, and z are compared to see if they are
already
              in increasing order.  If they are, no changes are
made.
              If x <= y but y > z, then y and z are swapped, and
then
              x and the new value of y (what was z) are compared
and
              put in correct order.
              If x > y, then x and y are swapped so that x < y.
              Next it is necessary to discover where z belongs in
              relation to x and y.  If y <= z, nothing more needs
doing.
              If, instead, z < y, then y and z are swapped and then
              x and the new y are compared and put in order.

AUTHOR:       W. Knight
*************************************************************
/

void sort3 (float & x, float & y, float & z)
{
   float temp;

   if (x <= y  &&  y <= z) // the values are already in order;
      ;                    // do nothing

   else if (x <= y) // then z < y (or we'd have stopped above)
   {
      swap_floats (z, y);  // After this call, y < z and x <= z are
true
                           // but we don't know how x and y
compare.
      if (x > y)
```

```
         swap_floats (x, y);  // Now x < y <= z must be true.
   }

   else // If neither of the above is true, then we know that y < x
   {
      swap_floats (x, y);  // After this call, x < y  is true.

      if (y <= z)  // the values are now in correct order;
         ;         // do nothing

      else // it must be the case that z < y
      {
         swap_floats (y, z);  // After this call, y <= z is true.

         if (x > y)
            swap_floats (x, y);
      }
   }
}
```

29. The following function calls the `swap_floats` function given in problem 27.

```
/*************************  R E V E R S E
**************************

DESCRIPTION:  Reverses the order of the objects in an array.

PARAMETERS:

  a           The array of floating point numbers whose objects
will
              be reversed.

  n           The number of objects in the array, starting at cell
0.

RETURNS:      Void (no value).

ALGORITHM:    One array index starts at the left end of the array
and
              moves to the right, while another starts at the right
end
              and moves to the left.  Objects in the cells
indicated by
              these two indexes are swapped.  The process ends when
              the two indexes meet or cross each other.

AUTHOR:       W. Knight

************************************************************
/


void reverse (float a[], int n)
{
   int i = 0, j = n - 1;

   while (i < j)
      swap_floats (a[i++], a[j--]);
}
```

30.

```
/*****************************  S U M
**************************

DESCRIPTION:  Calculates and returns the sum of the numbers in an
array.

PARAMETERS:

  a           The array of floating point numbers to be summed

  n           The number of objects in the array, starting at cell
0.

RETURNS:      The sum  a[0] + a[1] + . . . + a[n-1].

ALGORITHM:    A summing variable is initialized to zero and then
each
              floating point value in the array is added in turn.

AUTHOR:       W. Knight

*************************************************************
/

float sum (const float a[], int n)
{
   float sum_so_far = 0.0;
   int   i;

   for (i = 0; i < n; ++i)
      sum_so_far += a[i];

   return sum_so_far;
}
```

31. Here is the simplest solution.

```
/************** L O C A T I O N   O F   L A R G E S T
**************

DESCRIPTION:  Finds the index of the largest number in an array.

PARAMETERS:

  a           An array of integers to be scanned.

  n           The number of objects in the array, starting at cell
0.

RETURNS:      The index of the cell containing the largest integer.
              If the largest integer appears in more than one cell,
              then the index of the leftmost cell where it appears
              will be returned.

ALGORITHM:    A variable that will hold the index of the largest
integer
              is initialized to zero, and we pretend to have
scanned
              that cell and found that it contains the largest
integer
              seen "so far".  Then successive cells are examined
and
              compared with the cell containing the largest integer
              so far.  When a cell containing a new largest integer
is
              encountered, the variable that holds the index of the
              largest integer seen so far is modified to the new
index.

AUTHOR:       W. Knight

************************************************************
/

int location_of_largest (const int a[], int n)
{
   int best = 0; // Location of the largest so far.
   int i;

   for (i = 1; i < n; ++i) // Start comparing at the second cell.
      if (a[i] > a[best])
         best = i;
```

```
    return best;
}
```

[Another solution, not as elegant as the one above, is given on the following page.
An INCORRECT solution, commonly given by students, is shown two pages father along.]

The following solution is not as elegant as the one on the preceding page.

```
/************** L O C A T I O N   O F   L A R G E S T
**************

DESCRIPTION:  Finds the index of the largest number in an array.

PARAMETERS:

  a           An array of integers to be scanned.

  n           The number of objects in the array, starting at cell
0.

RETURNS:      The index of the cell containing the largest integer.
              If the largest integer appears in more than one cell,
              then the index of the leftmost cell where it appears
              will be returned.

ALGORITHM:    A variable that will hold the index of the largest
integer
              is initialized to zero, and we pretend to have
scanned
              that cell and found that it contains the largest
integer
              seen "so far".  We initialize a "largest_value"
variable
              to the value a[0]. Then successive cells are examined
and
              compared with the value of the largest integer so
far.
              When a cell containing a new largest integer is
              encountered, the variable that holds the index of the
              largest integer seen so far is modified to the new
index,
              and the largest_value variable is updated.

AUTHOR:       W. Knight

************************************************************
/

int location_of_largest (const int a[], int n)
{
   int largest_value = a[0]; // First cell contains largest so far.
   int best = 0;             // Location of the largest so far.
   int i;
```

```
   for (i = 1; i < n; ++i)
      if (a[i] > largest_value)
      {
         largest_value = a[i];
         best = i;
      }

   return best;
}
```

[A commonly given -- but INCORRECT -- solution is shown on the next page.]

The following is NOT an acceptable solution because it does not allow for the possibility that all the integers in the array may be negative or zero.

```
int location_of_largest (const int a[], int n) // NOT ACCEPTABLE
{
   int largest_value = 0; // Assume the largest value will be > 0.
   int best;              // Eventual location of the largest
value.
   int i;

   for (i = 0; i < n; ++i)
      if (a[i] > largest_value)  // Will be true for some i only if
      {                          // the array contains at least one
         largest_value = a[i];   // value strictly greater than 0.
         best = i;
      }

   return best;
}
```

The incorrect solution above can be corrected by initializing `largest_value` to the most negative possible integer value (`INT_MIN`, which is defined in the header file `<limits.h>`) and initializing `best` to `0`.

32. Here is the simplest solution. It searches from right to left and quits when (if) if finds the target.

```
/************** L O C A T I O N   O F   T A R G E T
**************

DESCRIPTION:  Finds the index of the cell (if any) where a "target"
              integer is stored.

PARAMETERS:

  a           An array of integers to be scanned.

  n           The number of objects in the array, starting at cell
0.

  target      The integer that we hope to find in some cell of
array a.

RETURNS:      The index of the cell containing the integer "target"
              provided there is such a cell.  If there is not, then
              the function returns -1 as a sentinel value.
              If the target integer appears in more than one cell,
              then the index of the rightmost cell where it appears
              will be returned.

ALGORITHM:    The value parameter "n" is used to scan backward
across
              the array, looking for a cell containing a copy of
              "target".  If a copy is found, the search is halted
and
              the index of that cell is returned.  If no such cell
is
              found, "n" will run off the left end of the array and
              wind up with value -1.

AUTHOR:       W. Knight

************************************************************
/


int location_of_target (const int a[], int n, int target)
{
   --n;  // Make n "point" to the last cell of the array.

   while (n >= 0  &&  a[n] != target) // Search right to left
      --n;
```

```
    return n;  // Returns -1 if target is not in array a[].
}
```

[Another solution, which in general is not as efficient as the one above, is on the next page.]

The following solution is acceptable, although it is not in general as efficient as the one above because it always examines every cell of the array.

```
/************** L O C A T I O N   O F   T A R G E T
**************

DESCRIPTION:  Finds the index of the cell (if any) where a "target"
              integer is stored.

PARAMETERS:

  a           An array of integers to be scanned.

  n           The number of objects in the array, starting at cell
0.

  target      The integer that we hope to find in some cell of
array a.

RETURNS:      The index of the cell containing the integer target,
              provided there is such a cell.  If there is not, then
              the function returns -1 as a sentinel value.
              If the target integer appears in more than one cell,
              then the index of the rightmost cell where it appears
              will be returned.

ALGORITHM:    A location variable is initialized to -1 in case no
copy
              of "target" is found.  Then each cell of the array is
              examined, starting at the left end, and each time a
copy
              "target" is found, the location variable is updated
to
              the index of that cell.

AUTHOR:       W. Knight

*****************************************************************
/

int location_of_target (const int a[], int n, int target)
{
   int location = -1;  // Target not seen yet.
   int i;

   for (i = 0; i < n; ++i)
      if (a[i] == target)
         location = i;
```

```
    return location;
}
```

33.

```
/********************  R O T A T E   R I G H T
********************

DESCRIPTION:  Shifts the contents of array cells one cell to the
right,
              with the last cell's contents moved to the left end.

PARAMETERS:

  a           The floating point array to be modified.

  n           The number of objects in the array, starting at cell
0.

RETURNS:      Void (no value).

ALGORITHM:    The object in the right-most cell is copied to a
temporary
              location, and then the object each cell to the left
of
              the last cell is copied to its immediate right
neighbor;
              the process moves from right to left.  Finally, the
object
              in the temporary location is copied to the leftmost
cell.

AUTHOR:       W. Knight

************************************************************
/

void rotate_right (float a[], int n)
{
   float temp = a[n-1];  // Hold the contents of the last cell.

   int i;

   for (i = n - 1;  i >= 1;  --i)
      a[i] = a[i-1];

   a[0] = temp;
}
```

34.

```
/*********************  S H I F T   R I G H T
*********************

DESCRIPTION:  Shifts the contents of some subarray in an array to
the
              right by a specified number of cells.

PARAMETERS:

  a           The floating point array to be modified.

  left        The index of the leftmost cell of the subarray to be
              shifted.

  right       The index of the rightmost cell of the subarray.

  distance    The number of cells by which the subarray will be
shifted.

RETURNS:      1 provided left <= right and distance > 0;  returns 0
              if either of these conditions is violated.

ALGORITHM:    An index variable is initialized to "point" to the
              rightmost cell of the subarray to be shifted;
another
              index variable is initialized to point to the cell to
              which the rightmost object will be copied.  Then the
              copying takes place and the indexes are moved to the
              left (decremented by 1).  The occurs repeatedly until
              the object in the leftmost cell of the subarray has
              been copied.

AUTHOR:       W. Knight

*************************************************************
/

int shift_right (float a[], int left, int right, int distance)
{
   if (left > right  ||  distance <= 0)
      return 1;

   int i = right,        // points to the cell to be shifted
       j = i + distance; // points to the receiving cell

   while (i >= left)
   {
```

```
        a[j] = a[i];
        --i;
        --j;
    }

    return 0;
}
```

[A more elegant version is given on the next page.]

Here is a slightly more elegant version.

```
/*********************  S H I F T   R I G H T
*********************

DESCRIPTION:  Shifts the contents of some subarray in an array to
the
              right by a specified number of cells.

PARAMETERS:

  a           The floating point array to be modified.

  left        The index of the leftmost cell of the subarray to be
              shifted.

  right       The index of the rightmost cell of the subarray.

  distance    The number of cells by which the subarray will be
shifted.

RETURNS:      1 provided left <= right and distance > 0;  returns 0
              if either of these conditions is violated.

ALGORITHM:    An index variable is initialized to "point" to the
              rightmost cell of the subarray to be shifted.  Then
              the object in that cell is copied to the cell that's
              "distance" units to the right.  Then the index is
              decremented by 1 and the same process is repeated.
              This continues until all objects in the subarray have
              been copied.

AUTHOR:       W. Knight

*************************************************************
/

int shift_right (float a[], int left, int right, int distance)
{
   if (left > right  ||  distance <= 0)
      return 1;

   int i;

   for (i = right; i >= left; --i)
      a[i + distance] = a[i];

   return 0;
```

```
}
```

35.

```
/************************  S U B T O T A L
**********************

DESCRIPTION:  Replaces each number in an array with the sum of all
the
              numbers up to that location in the original array.

PARAMETERS:

  a           The floating point array to be modified.

  n           The number of objects in the array, starting at cell
0.

RETURNS:      Void (no value).

ALGORITHM:    Starting with cell 1 and moving right, the number in
each
              cell is replaced by the sum of that number and the
sum
              that's now in the cell just to the left.

AUTHOR:       W. Knight

*****************************************************************
/

void subtotal (float a[], int n)
{
   int i;

   for (i = 1; i < n; ++i)
      a[i] += a[i-1];
}
```

36.

```
/************************  C O N C A T E N A T E
*********************

DESCRIPTION:  Copies numbers from two arrays into a third array.
The
              numbers from the second array are placed to the right
of
              the numbers copied from the first array.

PARAMETERS:

  a, b        The arrays from which numbers will be copied into
"c".

  c           The array that will receive the copied numbers.

  m, n        The number of numbers to be copied from arrays a and
b
              respectively, starting in each case at cell 0.

  p           The capacity of array c.

RETURNS:      1, provided the capacity of c is at least equal to
the
              number of numbers that are to be copied from arrays a
and
              b;  0 is returned if this condition fails.

ALGORITHM:    If c has adequate capacity, then the leftmost m cells
of
              array a are copied to the leftmost m cells of c, and
              then the leftmost n cells of b are copied into the
              n cells of c lying just to the right of the first m
cells.

AUTHOR:       W. Knight

*****************************************************************
/

int concatenate (const int a[], int m,
                 const int b[], int n,
                       int c[], int p)
{
   if (m + n > p)
      return 1;

   int i, j;
```

```
   for (i = 0; i < m; ++i)
      c[i] = a[i];

   for (j = 0; j < n; ++j)
      c[i++] = b[j];       // Increment i and j after each
assignment

   return 0;
}
```

37.

```
/***************  N U M B E R   O F   M A T C H E S
***************

DESCRIPTION:  Compares two NUL-terminated character arrays to
determine
              how many of the initial characters match.

PARAMETERS:

  a, b        The character arrays to be compared.  Each is
terminated
              by the NUL character '\0'.

RETURNS:      The number of non-NUL initial matches in the two
arrays.
              For example, if the arrays contain "boasted" and
"boats"
              (with NUL following the final 'd' and 's'
respectively),
              then 3 will be returned because "boa" matches "boa".

ALGORITHM:    The characters are compared pairwise, starting at
cell 0,
              until a NUL is reached or a pair of characters does
not
              match.

AUTHOR:       W. Knight

************************************************************
/

int number_of_matches (const char a[], const char b[])
{
   const char NUL = '\0';

   int i = 0; // i keeps track of how many pairs of cells match

   while (a[i] != NUL  &&  b[i] != NUL  &&  a[i] == b[i])
      ++i;

   return i;
}
```

38.

```
/************  E L I M I N A T E   D U P L I C A T E S
**************

DESCRIPTION:  Examines an array of integers and eliminates all
              duplication of values.  The distinct integers are all
              moved to the left part of the array.

PARAMETERS:

  a           The integer array to be modified.

  n           The number of integers in the array, starting at cell
0.
              This is a reference parameter;  its value will be
changed
              if necessary to indicate the number of distinct
integers
              that end up in the left part of the array.

RETURNS:      Void (no value).

ALGORITHM:    At all times during the algorithm, an integer keeps
track
              of the last cell of the subarray containing all the
              distinct integers that have been found and moved to
that
              subarray.  Another integer moves along the unexamined
              portion of the array.  Each time a new integer is
              encountered, it is checked against all the integers
in
              the subarray of distinct integers to see whether it
              should be added to that subarray.

AUTHOR:       W. Knight

************************************************************
/


void eliminate_duplicates (int a[], int & n)
{
   int last_unique = 0; // Keeps track of the end of the subarray
of
                        // integers known to be all different.
   int i;
```

```
   for (i = 1; i < n; ++i) // Start i at the second cell (number
1).
   {
      // Determine whether a[i] is already present among the
integers in
      // cells a[0],...,a[last_unique].

      int j = 0;

      while (j <= last_unique  &&  a[i] != a[j])
         ++j;

      if (j > last_unique)          // then a[i] is not present
earlier,
         a[++last_unique] = a[i];  // so put a[i] into the list of
                                   // all different integers.
   }

   n = last_unique + 1; // Modify n so that it tells the number of
                        // distinct integers in the processed
array.
}
```

39.

```
/*****************************************************************
*****

PROGRAMMER:     William Knight

DATE COMPLETED: July 10, 1998

COMPILER:       Borland C++ for Windows95

DESCRIPTION:    This program prompts an interactive user to enter
                positive integers, and it prints all the positive
                divisors of each positive integer entered.

NOTE:           If the user enters one or more non-numeric
characters
                when prompted for an integer, the program will go
into
                an endless loop.

*****************************************************************
****/


#include <iostream.h>


/********************  G L O B A L   C O N S T A N T S
*************/

const int FALSE = 0, TRUE = 1;


/*****************  F U N C T I O N   P R O T O T Y P E S
************/

void print_initial_explanation (void);
void get_number_from_user (int & n);
void print_divisors (int n);
int  user_wishes_to_repeat (void);


/***************************  M A I N
*****************************/

int main (void)
{
   int users_integer;
```

```
   print_initial_explanation();

   do
   {
      get_number_from_user (users_integer);

      print_divisors (users_integer);
   }
   while (user_wishes_to_repeat());
}
```

```
/******** P R I N T  I N I T I A L  E X P L A N A T I O N
*********

DESCRIPTION:   This function prints some text on the screen.

PARAMETERS:    None.

RETURNS:       Void (no value).

WRITTEN BY:    W. Knight

**************************************************************
****/

void print_initial_explanation (void)
{
   cout << "\n\nThis program is designed to exhibit the positive
\n";
   cout << "divisors of positive integers supplied by you.  The\n";
   cout << "program will repeatedly prompt you to enter a
positive\n";
   cout << "integer.  Each time you enter a positive integer,
the\n";
   cout << "program will print all the divisors of your integer
in\n";
   cout << "a column and in decreasing order.\n\n";
}


/************* G E T  N U M B E R  F R O M  U S E R
***************

DESCRIPTION:   This function prompts an interactive user for a
positive
               integer, reads the integer, and returns it in the
               variable passed to this function.

PARAMETER:

   n           The positive integer supplied by the user.  This is
a
               reference parameter.  It is intended that the
argument
               passed through this parameter will receive a new
value.

RETURNS:       Void (no value).

ALGORITHM:     The user is prompted for the positive integer.  If
the
```

```
                integer entered by the user is zero or negative, the
                user will be informed that the input is not positive
                and will be given another chance to enter valid
data.
                The function will not exit until the user has
entered
                valid data.

WRITTEN BY:     W. Knight

*****************************************************************
****/

void get_number_from_user (int & n)
{
   do
   {
      cout << "Please enter a positive integer: ";
      cin  >> n;
      if (n <= 0)
         cout << '\n' << n << " is not a positive integer.\n";
   }
   while (n <= 0);
}
// ---------------------------------------------------------------
---
```

// COMMENT: In the function "get_number_from_user", if the interactive user enters a
// non-numeric character in response to the prompt for a positive integer, then the program will go into //
an endless loop. That can be avoided as follows:

```
// place an "#include <stdlib.h>" directive at the top of the program, and then use this loop:
//   do
//   {
//      cout << "Please enter a positive integer: ";
//      cin  >> n;
//      if (!cin.good())  // cin.good() returns 0 if cin >> n has
failed
//      {
//         cout << "\n\n *** Non-numeric data entered.  Program";
//         cout << "  cannot recover, so will exit. *** \n\n" <<
endl;
//         exit (1);  // ABORT THE PROGRAM (exit is defined in
stdlib.h)
//      }
//
//      if (n <= 0)
//         cout << '\n' << n << " is not a positive integer.\n";
//   }
//   while (n <= 0);
```

```
/******************** P R I N T  D I V I S O R S
********************

DESCRIPTION:   This function prints all the divisors of a positive
               integer in a column and in decreasing order.

PARAMETER:

   n           The positive integer whose divisors will be printed.

RETURNS:       Void (no value).

ALGORITHM:     First the value of n is printed.  Then, starting
with
               the next possible smaller divisor (n/2), each
integer
               down to 1 is tested to see if it is a divisor of n,
               and every divisor is printed.

WRITTEN BY:    W. Knight

************************************************************
****/

void print_divisors (int n)
{
   cout << n << endl;

   int trial_divisor;

   for (trial_divisor = n / 2;  trial_divisor >= 1;  --
trial_divisor)
      if (n % trial_divisor == 0)
         cout << trial_divisor << endl;

   cout << endl;
}
```

```
/*********** U S E R   W I S H E S   T O   R E P E A T
***************

DESCRIPTION:   This function determines whether an interactive user
               wishes to enter another positive integer.

PARAMETERS:    None.

RETURNS:       TRUE if the user wishes to repeat, FALSE if not.

ALGORITHM:     The user is asked to type Y for "yes, I wish to
repeat"
               or N for "no, I do not".  The user's response is
then
               read from the keyboard, and if it is neither Y nor N
               (the lower case versions of these characters are
also
               accepted), then the user is asked to re-enter a
response.

WRITTEN BY:    W. Knight

****************************************************************
****/

int user_wishes_to_repeat (void)
{
   char response;

   cout << "Would you like to see the divisors of another";
   cout << " integer (Y/N)? ";
   cin  >> response;

   while (response != 'Y'  &&  response != 'y'  &&  response != 'N'
                                                &&  response !=
'n')
   {
      cout << "\nPlease respond with Y (or y) for yes and N (or
n)";
      cout << " for no." << endl;

      cout << "Would you like to see the divisors of another";
      cout << " integer (Y/N)? ";
      cin  >> response;
   }

   if (response == 'Y'  ||  response == 'y')
      return TRUE;
   else
```

```
        return FALSE;
}
```