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>
using namespace std;

int 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>
using namespace std;

int 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>
using namespace std;

int 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>
using namespace std;

int 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];
      int   i;

      cout << "Enter a sentence on the line below." << endl;
      cin  >> message[0];

      i = 0;
      while (i < LENGTH - 1  &&  message[i] != '\n')
        {
          ++i;
          cin >> message[i];
        }

      message[i] = '\0'; // Replace '\n' if that was read.
                         // Otherwise replace the last character read.

      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 the statement "`cin >> message[0];`" is replaced by the statement "`cin.get(message[0]);`" , and similarly "`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

[continued on the next page]

```
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: $x \le y \le z$. 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
58	26	91	34	70	34	88

and

0	1	2	3
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
'b'	'o'	'a'	's'	't'	'\0'

and

0	1	2	3	4
'b'	'o'	'a'	't'	'\0'

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
_x( )_x( )4
_x( ) n
blank line
blank line
_x( ) 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. (White space characters include the blank character and the newline character as well as several other control characters.) 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.
```

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. 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 white 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;
}
```