Starting Out with C++ Early Objects 10th edition Gaddis Solutions Manual

SKU: 9780135235003-SOLUTIONS

Starting Out With C++: Early Objects, Tenth Edition
Solutions to End-of-Chapter Review Questions

Chapter 1

1. programmed
2. CPU
3. arithmetic logic unit (ALU) and control unit
4. disk drive
5. operating systems and application software
6. instructions
7. programming
8. Machine language
9. High-level
10. Low-level
11. portability
12. key
13. programmer-defined symbols
14. Operators
15. Punctuation
16. syntax
17. variable
18. defined (or declared)
19. input, processing, output
20. Input
21. Output
22. hierarchy chart

23. Main memory, or RAM, is volatile, which means its contents are erased when power is removed from the computer. Secondary memory, such as a disk, CD, or flash drive, does not lose its contents when power is removed from the computer.

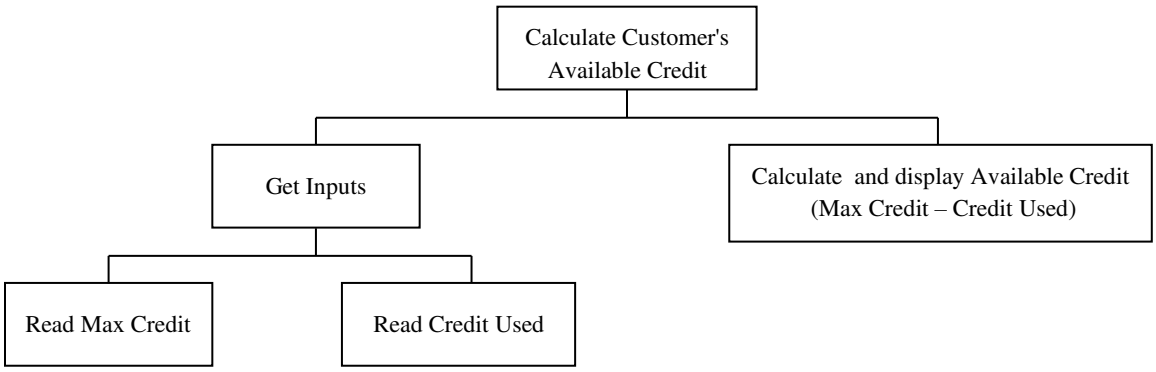
24. A syntax error is the misuse of a key word, operator, punctuation, or other part of the programming language. A logical error is a mistake that tells the computer to carry out a task incorrectly or to carry out tasks in the wrong order. It causes the program to produce the wrong results.

25. C

26. B

27. D

28. Hierarchy Chart:



29. ***Account Balance High Level Pseudocode***

```
Have user input starting balance
Have user input total deposits
Have user input total withdrawals
Calculate current balance
Display current balance
```

Account Balance Detailed Pseudocode

```
Input startBalance           // with prompt
Input totalDeposits          // with prompt
Input totalWithdrawals       // with prompt
currentBalance = startBalance + totalDeposits - totalWithdrawals
Display currentBalance
```

30. ***Sales Tax High Level Pseudocode***

```
Have user input retail price
Have user input sales tax rate
Calculate tax amount
Calculate sales total
Display tax amount and sales total
```

Sales Tax Detailed Pseudocode

```
Input retailPrice             // with prompt
Input salesTaxRate            // with prompt
taxAmount = retailPrice * salesTaxRate
salesTotal = retailPrice + taxAmount
Display taxAmount, salesTotal
```

31. 45
32. 9
33. 7
34. 28
35. 365
36. Line 3 adds *num1* + *num1*. It should add *num1* + *num2*.
    Line 4 displays *avg*, but it should display *average*. The name of the displayed variable must match the name of the variable holding the answer.
37. The error is that the program performs its math operation before the user has entered values for the variables `width` and `length`.
38. Some of the questions that should be asked are:
    What standard ceiling height should be used, or is this figure to be input?
    How many square feet should be subtracted out for windows and doors, or do you also want this information input since it could vary by room?
    Are the ceilings also to be painted, or just the walls?
    How many square feet will 1 gallon of paint cover?
    How many coats of paint will you use, or should this information be input?

Chapter 2

1. semicolon
2. `iostream`
3. `main`
4. `#`
5. braces `{}`
6. literals (also sometimes called constants)
7. 9.7865E14
8. 1, 2

9. A) valid B) invalid C) valid D) valid
10. A) valid B) valid C) invalid D) invalid
11. All are valid.
12. A) invalid B) invalid C) invalid D) valid
13. A) valid B) invalid C) valid D) valid, but it prints the contents of variable `Hello`, not the string `"Hello"`.
14. A) valid B) invalid C) valid
15. A) 11 B) 14 C) 3 D) 3.5
16. A) 9.5 B) 14.0 C) 1.0 D) 1.75

17.
```
double temp,          // This can also be done on a single line.
       weight,
       height;
```

18.
```
int months = 2,
    days,
    years = 3;
```

19. A)
```
d2 = d1 + 2;
```
B)
```
d1 = d2 * 4;
```
C)
```
c = 'K';
```
D)
```
i = 'K';
```
E)
```
i = i - 1;
```

20. A)
```
d1 = d2 - 8.5;
```
B)
```
d2 = d1 / 3.14;
```
C)
```
c = 'F';
```
D)
```
i = i + 1;
```
E)
```
d2 = d2 + d1;
```

21.
```
cout << "Two mandolins like creatures in the\n\n\n";
cout << "dark\n\n\n";
cout << "Creating the agony of ecstasy.\n\n\n";
cout << "                    - George Barker\n\n\n";
```

22.
```
cout << "L\n"
     << "E\n"              // This can also be written as a single string literal:
     << "A\n"              // cout << "L\nE\nA\nF\n";
     << "F\n";
```

23.
```
Input weeks                // with prompt
days = weeks * 7
Display days
```

24.
```
Input eggs                  // with prompt
cartons = eggs / 12         // perform integer divide
Display cartons
```

25.
```
Input speed                 // with prompt
Input time                  // with prompt
distance = speed * time
Display distance
```

26.
```
Input miles                 // with prompt
Input gallons               // with prompt
milesPerGallon = miles / gallons
Display milesPerGallon
```

27. A)
```
0
100
```
B)
```
8
2
```
C)
```
I am the incrediblecomputing
 machine
and I will
amaze
you.
```

28. A)
```
Be careful!
This might/n be a trick question.
```
B) 23
1

29. On line 1 the comments symbols are backwards. They should be `/*  */`.
On line 2 `iostream` should be enclosed in angle brackets.
On line 5 there shouldn't be a semicolon after `int main()`.
On lines 6 and 13 the opening and closing braces of function `main` are reversed.
On line 7 there should be a semicolon after `int a, b, c`.
In addition, the comment symbol is incorrect. It should be `//`.
On lines 8-10 each assignment statement should end with a semicolon.
On line 11 `cout` begins with a capital letter. In addition, the stream insertion operators should read << instead of >> and the variable that is ouput should be `c` instead of capital `C`.

30. Whatever problem a pair of students decides to work with they must determine such things as which values will be input vs. which will be set internally in the program, how much precision is required on calculations, what output will be produced by the program, and how it should be displayed. Students must also determine how to handle situations that are not clear cut. In the paint problem many of these considerations are listed in the teacher answer key (Chapter 1, Question 34). In the recipe program students must determine such things as how to handle quantities, like one egg, that cannot be halved. In the driving program, knowing distance and speed are not enough. Agreement should be reached on how to handle delays due to traffic lights and traffic congestion. Should this be an input value, computed as a percent of overall driving time, or handled some other way?

Lab 1

To begin

- Follow the instructions your professor gives you to log on to your system and create a folder named `Lab1` in your work space.
- Copy all the files in the `Chapter01` lab folder to your `Lab1` folder.
- Follow the instructions your professor gives you to run the IDE your class will use for your C++ programs and create a project named `Lab1`.

LAB 1.1 – TRY IT: Compiling and Running Your First Program

Step 1: Add the `greeting.cpp` program from your `Lab1` folder to the project. Your professor will show you how to do this. Here is a copy of the source code.

```
 1 // greeting.cpp
 2 // This program prints a message to greet the user.
 3 #include <iostream>          // Needed to do C++ I/O
 4 #include <string>            // Needed by some compilers to use strings
 5 using namespace std;
 6
 7 int main ()
 8 {
 9    string name;              // This declares a variable to
10                              // hold the user's name
11    // Get the user's name
12    cout << "Please enter your first name: ";
13    cin  >> name;
14
15    // Print the greeting
16    cout << "Hello, " << name << "." << endl;
17
18    return 0;
19 }
```

Step 2: Read the source code. What do you think the program will display when it is run if the user enters the name `Adam` on line 12? _____

Step 3: Follow the instructions your professor gives you to compile the program. You should see a message at the bottom of the screen telling you the program has compiled correctly. This message will be different for each compiler, but may look something like this:

```
greeting.o - 0 error(s), 0 warning(s)
```

Notice what the message looks like on *your* system.

Step 4: Now follow the instructions your professor gives you to execute the program. Enter the name `Adam` when the program prompts you for a name. Did you get the output you expected? _____ Run the program again, and this time enter *your* first name when you are prompted for a name. Write a copy of the output here. _____

Computer Program Errors

The `greeting.cpp` program compiled and ran correctly because it contained no errors. However, there are three different types of errors that can occur in a computer program, and all of these need to be found and fixed before a program will work correctly. Each of these three errors is examined below.

LAB 1.2 – Syntax Errors

A *syntax error* in a program, just like a syntax error in English, means that a grammar rule has been broken. One or more of the programming statements in the source code do not follow the rules for how a C++ program must be written. Let's take a look at the most commonly made C++ syntax error, an omitted semi-colon.

Step 1: Remove the semi-colon on line 12 of the `greeting.cpp` program. To do this you can simply place your cursor at the end of line 12 and press the back space key. Once the semi-colon is gone, recompile the program and look at the error message displayed at the bottom of the screen. The exact message you get will depend on which compiler you are using, but it will likely look something like this:

```
greeting.cpp:13: error: expected ';' before "cin"
```

Always read the message carefully. Sometimes it is clear what is wrong. Other times the message may be confusing to a new programmer, but you will get better at deciphering compiler error messages as you gain more experience. This message is quite clear. It says that a semi-colon is missing before the word `cin`.

Notice that the compiler error message also includes a line number indicating the approximate, but not exact, location where the error occurred. In this case the error occurred at the end of line 12, but the compiler message reports the error as occurring on line 13. This is because the compiler did not detect the problem until it reached the `cin` on line 13. Any time you get a compiler error message and do not see a problem in the line reported , try checking the previous line.

Step 2: A program containing compiler errors cannot be run until the errors are fixed and the program is recompiled. Put the semi-colon back in at the end of line 12 and then compile the program again. If you have done it correctly, it will compile with no errors this time. Now you can run the program again.

Step 3: Follow the instructions your professor gives you to remove `greeting.cpp` from the project. This must be done so the same project can be used to run a different program.

LAB 1.3 – Run-time Errors

A *run-time error* occurs when a program instruction tells the program to do something it is unable to correctly do. This type of error cannot be detected by a compiler because no syntax rules have been broken. So the program will compile but, as the name suggests, something will go wrong when the program is run. In some cases a run-time error causes the program to abort; in others it continues running, but produces incorrect results. Let's take a look at a common run-time error, attempting to divide by zero.

Step 1: Add the `average.cpp` program from your `Lab1` folder to the project. Here is a copy of the source code.

```
 1 // average.cpp
 2 // This program finds the average of two numbers.
 3 // It contains two errors that must be fixed.
 4 #include <iostream>
 5 using namespace std;
 6
 7 int main ()
 8 {
 9    int size = 0;             // The number of values to be averaged
10    double num1,
11           num2,
12           average;           // Average of num1 and num2
13
14    // Get the two numbers
15    cout << "Enter two numbers separated by one or more spaces: ";
16    cin  >> num1 >> num2;
17
18    // Calculate the average
19    average = num1 + num2 / size;
20
21    // Display the average
22    cout << "The average of the two numbers is: " << average << endl;
23
24    return 0;
25 }
```

Step 2: Read through the source code to see if you can spot any errors in the program. Two lines contain errors, but even if you can find them, do not fix them yet.

Step 3: Compile the program. Since it contains no syntax errors, the compiler should not detect any errors.

Step 4: Run the program and, at the prompt, enter `10 5`. Did the run-time error cause the program to abort or to produce incorrect results? If it ran without aborting it probably displayed something like this:

```
Enter two numbers separated by one or more spaces: 10 5
The average of the two numbers is: 1.#INF
```

Step 5: The error occurs on line 19 when a quantity is divided by `size`. Notice that `size` is set in line 9 to 0. Correct the error by changing line 9 of the source code to set `size` equal to `2` so that the quantity on line 19 will be divided by 2. Then recompile and rerun the program, again entering `10` and `5` for the two numbers. The output should now look like this:

```
Enter two numbers separated by one or more spaces: 10 5
The average of the two numbers is: 12.5
```

LAB 1.4 – Logic Errors

A *logic error* causes a program to work incorrectly. It doesn't break any syntax rules and doesn't tell the computer to do anything it is unable to do. Instead it occurs when the program logic is wrong because what the programmer tells the program to do does not match what he or she means for it to do. If you were explaining to someone how to do laundry, it would be a logic error to tell them to put the laundry in the oven, when what you meant was for them to put it in the washer. Another logic error would occur if you told them to wash the laundry and then fold it, but forgot to tell them to dry it. It would also be a logic error if you got the steps out of order and told them to first fold the laundry, then dry it, and then finally to wash it. Likewise, for a program to be correct each instruction must be correct, no instructions must be omitted, and the instructions must be carried out in the right order.

Let's look at a logic error that causes a program to carry out two mathematical operations in the wrong order.

Step 1: Look again at the output created by the `average.cpp` program in Step 5 of Lab 1.3 above. Notice that the user entered `10` and `5`, but the program reported the average to be `12.5`, rather than `7.5`. The error occurs on line 19. To find an average of a set of values, you must add the values *before* you divide by the number of values. But the code on line 19 tells the computer to divide the value stored in `num2` by `size` before adding the result to the value stored in `num1`. You will learn more in chapters 2 and 3 about how to write correct mathematical statements in C++. For now, just add parentheses on line 19 so it says:

```
average = (num1 + num2) / size;
```

Step 2: Recompile and rerun the program, again entering 10 and 5 at the prompt. Now that you have corrected the logic error, you should get the following correct result.

```
Enter two numbers separated by one or more spaces: 10 5
The average of the two numbers is: 7.5
```

LAB 1.5 – Fix the Errors

Step 1: Remove `average.cpp` from the project and add `findErrors.cpp` to the project. Here is a copy of the source code.

```
 1 // findErrors.cpp
 2 // This program has one syntax error and one logic error. Find and fix them.
 3 // PUT YOUR NAME HERE.
 4 #include <iostream>
 5 using namespace std
 6
 7 int main ()
 8 {
 9    double length = 0,         // Length of a room in feet
10           width = 0,          // Width of a room in feet
11           area;               // Area of the room in sq. ft.
12
13    // Get the room dimensions
14    cout << "Enter room length (in feet): ";
15    cin  >> length;
16
17    cout << "Enter room width (in feet): ";
18    cin  >> length;
19
20    // Compute and display the area
21    area = length * width;
22    cout << "The area of the room is " << area << " square feet." << endl;
23
24    return 0;
25 }
```

Step 2: Put your name on line 3. Then compile the program. It contains one syntax error and one logic error.

Step 3: Use the compiler error message to help you locate the syntax error and fix it.

Step 4: Once the program compiles with no errors, run the program, and examine the output. Analyze what is going wrong so you can find and fix the logic error in the program. Once you have it running correctly the output should look like the following:

```
Enter room length (in feet): 15
Enter room width (in feet): 10
The area of the room is 150 square feet.
```

Step 5: Follow the instructions your professor gives you to print the final, correct `findErrors.cpp` source code and the output the program displays when it runs.

Lab 1 KEY

To begin

- Follow the instructions your professor gives you to log on to your system and create a folder named `Lab1` in your work space.
- Copy all the files in the `Chapter01` lab folder to your `Lab1` folder.
- Follow the instructions your professor gives you to run the IDE your class will use for your C++ programs and create a project named `Lab1`.

LAB 1.1 – TRY IT: Compiling and Running Your First Program

Step 1: Add the `greeting.cpp` program from your `Lab1` folder to the project. Your professor will show you how to do this. Here is a copy of the source code.

```
 1 // greeting.cpp
 2 // This program prints a message to greet the user.
 3 #include <iostream>          // Needed to do C++ I/O
 4 #include <string>           // Needed by some compilers to use strings
 5 using namespace std;
 6
 7 int main ()
 8 {
 9    string name;              // This declares a variable to
10                              // hold the user's name
11    // Get the user's name
12    cout << "Please enter your first name: ";
13    cin  >> name;
14
15    // Print the greeting
16    cout << "Hello, " << name << "." << endl;
17
18    return 0;
19 }
```

Step 2: Read the source code. What do you think the program will display when it is run if the user enters the name `Adam` on line 12? ____`Hello, Adam.`_____

Step 3: Follow the instructions your professor gives you to compile the program. You should see a message at the bottom of the screen telling you the program has compiled correctly. This message will be different for each compiler, but may look something like this:

```
greeting.o - 0 error(s), 0 warning(s)
```

Notice what the message looks like on *your* system.

Step 4: Now follow the instructions your professor gives you to execute the program. Enter the name `Adam` when the program prompts you for a name. Did you get the output you expected? _____ Run the program again, and this time enter *your* first name when you are prompted for a name. Write a copy of the output here. _____

Computer Program Errors

The `greeting.cpp` program compiled and ran correctly because it contained no errors. However, there are three different types of errors that can occur in a computer program, and all of these need to be found and fixed before a program will work correctly. Each of these three errors is examined below.

LAB 1.2 – Syntax Errors

A *syntax error* in a program, just like a syntax error in English, means that a grammar rule has been broken. One or more of the programming statements in the source code do not follow the rules for how a C++ program must be written. Let's take a look at the most commonly made C++ syntax error, an omitted semi-colon.

Step 1: Remove the semi-colon on line 12 of the `greeting.cpp` program. To do this you can simply place your cursor at the end of line 12 and press the back space key. Once the semi-colon is gone, recompile the program and look at the error message displayed at the bottom of the screen. The exact message you get will depend on which compiler you are using, but it will likely look something like this:

```
greeting.cpp:13: error: expected ';' before "cin"
```

Always read the message carefully. Sometimes it is clear what is wrong. Other times the message may be confusing to a new programmer, but you will get better at deciphering compiler error messages as you gain more experience. This message is quite clear. It says that a semi-colon is missing before the word `cin`.

Notice that the compiler error message also includes a line number indicating the approximate, but not exact, location where the error occurred. In this case the error occurred at the end of line 12, but the compiler message reports the error as occurring on line 13. This is because the compiler did not detect the problem until it reached the `cin` on line 13. Any time you get a compiler error message and do not see a problem in the line reported , try checking the previous line.

Step 2: A program containing compiler errors cannot be run until the errors are fixed and the program is recompiled. Put the semi-colon back in at the end of line 12 and then compile the program again. If you have done it correctly, it will compile with no errors this time. Now you can run the program again.

Step 3: Follow the instructions your professor gives you to remove `greeting.cpp` from the project. This must be done so the same project can be used to run a different program.

LAB 1.3 – Run-time Errors

A *run-time error* occurs when a program instruction tells the program to do something it is unable to correctly do. This type of error cannot be detected by a compiler because no syntax rules have been broken. So the program will compile but, as the name suggests, something will go wrong when the program is run. In some cases a run-time error causes the program to abort; in others it continues running, but produces incorrect results. Let's take a look at a common run-time error, attempting to divide by zero.

Step 1: Add the `average.cpp` program from your `Lab1` folder to the project. Here is a copy of the source code.

```
 1 // average.cpp
 2 // This program finds the average of two numbers.
 3 // It contains two errors that must be fixed.
 4 #include <iostream>
 5 using namespace std;
 6
 7 int main ()
 8 {
 9    int size = 0;             // The number of values to be averaged
10    double num1,
11           num2,
12           average;           // Average of num1 and num2
13
14    // Get the two numbers
15    cout << "Enter two numbers separated by one or more spaces: ";
16    cin  >> num1 >> num2;
17
18    // Calculate the average
19    average = num1 + num2 / size;
20
21    // Display the average
22    cout << "The average of the two numbers is: " << average << endl;
23
24    return 0;
25 }
```

Step 2: Read through the source code to see if you can spot any errors in the program. Two lines contain errors, but even if you can find them, do not fix them yet.

Step 3: Compile the program. Since it contains no syntax errors, the compiler should not detect any errors.

Step 4: Run the program and, at the prompt, enter `10 5`. Did the run-time error cause the program to abort or to produce incorrect results? If it ran without aborting it probably displayed something like this:

```
Enter two numbers separated by one or more spaces: 10 5
The average of the two numbers is: 1.#INF
```

Step 5: The error occurs on line 19 when a quantity is divided by `size`. Notice that `size` is set in line 9 to 0. Correct the error by changing line 9 of the source code to set `size` equal to `2` so that the quantity on line 19 will be divided by 2. Then recompile and rerun the program, again entering `10` and `5` for the two numbers. The output should now look like this:

```
Enter two numbers separated by one or more spaces: 10 5
The average of the two numbers is: 12.5
```

<Note: The program now runs, but the answer is wrong because there is still a logic error that will be dealt with in the next exercise.>

LAB 1.4 – Logic Errors

A *logic error* causes a program to work incorrectly. It doesn't break any syntax rules and doesn't tell the computer to do anything it is unable to do. Instead it occurs when the program logic is wrong because what the programmer tells the program to do does not match what he or she means for it to do. If you were explaining to someone how to do laundry, it would be a logic error to tell them to put the laundry in the oven, when what you meant was for them to put it in the washer. Another logic error would occur if you told them to wash the laundry and then fold it, but forgot to tell them to dry it. It would also be a logic error if you got the steps out of order and told them to first fold the laundry, then dry it, and then finally to wash it. Likewise, for a program to be correct each instruction must be correct, no instructions must be omitted, and the instructions must be carried out in the right order.

Let's look at a logic error that causes a program to carry out two mathematical operations in the wrong order.

Step 1: Look again at the output created by the `average.cpp` program in Step 5 of Lab 1.3 above. Notice that the user entered `10` and `5`, but the program reported the average to be `12.5`, rather than `7.5`. The error occurs on line 19. To find an average of a set of values, you must add the values *before* you divide by the number of values. But the code on line 19 tells the computer to divide the value stored in `num2` by `size` before adding the result to the value stored in `num1`. You will learn more in chapters 2 and 3 about how to write correct mathematical statements in C++. For now, just add parentheses on line 19 so it says:

```
average = (num1 + num2) / size;
```

Step 2: Recompile and rerun the program, again entering 10 and 5 at the prompt. Now that you have corrected the logic error, you should get the following correct result.

```
Enter two numbers separated by one or more spaces: 10 5
The average of the two numbers is: 7.5
```

LAB 1.5 – Fix the Errors

Step 1: Remove `average.cpp` from the project and add `findErrors.cpp` to the project. Here is a copy of the source code.

```
 1 // findErrors.cpp
 2 // This program has one syntax error and one logic error. Find and fix them.
 3 // PUT YOUR NAME HERE.
 4 #include <iostream>
 5 using namespace std           // Syntax error – missing semi-colon
 6
 7 int main ()
 8 {
 9    double length = 0,         // Length of a room in feet
10           width = 0,          // Width of a room in feet
11           area;               // Area of the room in sq. ft.
12
13    // Get the room dimensions
14    cout << "Enter room length (in feet): ";
15    cin  >> length;
16
17    cout << "Enter room width (in feet): ";
18    cin  >> length;            // Logic error – length is entered twice
19                               // width remains zero
20    // Compute and display the area
21    area = length * width;
22    cout << "The area of the room is " << area << " square feet." << endl;
23
24    return 0;
25 }
```

Step 2: Put your name on line 3. Then compile the program. It contains one syntax error and one logic error.

Step 3: Use the compiler error message to help you locate the syntax error and fix it.

Step 4: Once the program compiles with no errors, run the program, and examine the output. Analyze what is going wrong so you can find and fix the logic error in the program. Once you have it running correctly the output should look like the following:

```
Enter room length (in feet): 15
Enter room width (in feet): 10
The area of the room is 150 square feet.
```

Step 5: Follow the instructions your professor gives you to print the final, correct `findErrors.cpp` source code and the output the program displays when it runs.

See findErrors-KEY.cpp