3.2 — The debugging process

ALEX FEBRUARY 1, 2020

Let's say you've written a program, and it's not working correctly -- the code all compiles fine, but when you run it, you're getting an incorrect result. You must have a semantic error somewhere. How can you find it? If you've been following best practices by writing a little bit of code and then testing it, you may have a good idea where your error is. Or you may have no clue at all.

All bugs stem from a simple premise: Something that you thought was correct, isn't. Actually figuring out where that error is can be challenging. In this lesson, we'll outline the general process of debugging a program.

Because we haven't covered that many C++ topics yet, our example programs in this chapter are going to be pretty basic. That may make some of the techniques we're showing here seem excessive. However, keep in mind that these techniques are designed to be used with larger, more complex programs, and will be of more use in such a setting (which is where you need them most).

A general approach to debugging

Once a problem has been identified, debugging the problem generally consists of five steps:

1. Find the root cause of the problem (usually the line of code that's not working)
2. Ensure you understand why the issue is occurring
3. Determine how you'll fix the issue
4. Repair the issue causing the problem
5. Retest to ensure the problem has been fixed and no new problems have emerged

Let's use a real-life analogy here. Let's say one evening, you go to get some ice from the ice dispenser in your freezer. You put your cup up to the dispenser, press, and ... nothing comes out. Uh oh. You've discovered some kind of defect. What would you do? You'd probably start an investigation to see if you could identify the root cause of the issue.

Find the root cause: Since you hear the ice dispenser trying to deliver ice, it's probably not the ice delivery mechanism itself. So you open the freezer, and examine the ice tray. No ice. Is that the root cause of the issue? No, it's another symptom. After further examination, you determine that the ice maker does not appear to be making ice. Is the problem the ice maker or something else? The freezer is still cold, the water line isn't clogged, and everything else seems to be working, so you conclude that the root cause is that the ice maker is non-functional.

Understand the problem: This is simple in this case. A broken ice maker won't make ice.

Determine a fix: At this point, you have several options for a fix: You could work around the issue (buy bags of ice from the store). You could try to diagnose the ice-maker further, to see if there's a part that can be repaired. You could buy a new ice maker and install it in place of the current one. Or you could buy a new freezer. You decide to buy a new ice maker.

Repair the issue: Once the ice maker has arrived, you install it.

Retest: After turning the electricity back on and waiting overnight, your new ice maker starts making ice. No new issues are discovered.

Now let's apply this process to our simple program from the previous lesson:

```
1  #include <iostream>
2  // Adds two numbers
3  int add(int x, int y)
   {
       return x - y; // function is supposed to add, but it doesn't
4  }
   int main()
5  {
6      std::cout << add(5, 3) << '\n'; // should produce 8, but
   produces 2
       return 0;
   }
```

This code is nice in one regard: the bug is very apparent, because the wrong answer gets printed to the screen via line 11. That gives us a starting point for our investigation.

Find the root cause: On line 11, we can see that we're passing in literals for arguments (5 and 3), so there is no room for error there. Since the inputs to function *add* are correct, but the output isn't, it's pretty apparent that function*add* must be producing the wrong value. The only statement in function *add* is the return statement, which must be the culprit. We've found the problem line. Now that we know where to focus our attention, noticing that we're subtracting instead of adding is something you're likely to find via inspection.

Understand the problem: In this case, it's obvious why the wrong value is being generated -- we're using the wrong operator.

Determine a fix: We'll simply change *operator-* to *operator+*.

Repair the issue: This is actually changing *operator-* to *operator+* and ensuring the program recompiles.

Retest: After implementing the change, rerunning the program will indicate that our program now produces the correct value of 8. For this simple program, that's all the testing that's needed.

This example is trivial, but illustrates the basic process you'll go through when diagnosing any program.

Next lesson
3.3 A strategy for debugging

Back to table of contents

Previous lesson
3.1 Syntax and semantic errors

Leave a comment... Put C++ code between triple-backticks (markdown style):```Your C++ coc

Name*

Email*

Avatars from https://gravatar.com/ are connected to your provided email address.

Notify me about replies: POST COMMENT

12 COMMENTS

Newest

x