

Students:

Section 2.20 is a part of 1 assignment: **CSC108 CH02.11-2.24 P2B**

This assignment's due date has passed. Activity will still be recorded, but will not count towards this assignment (unless the due date is changed). See [this article](#) for more info.

Includes:

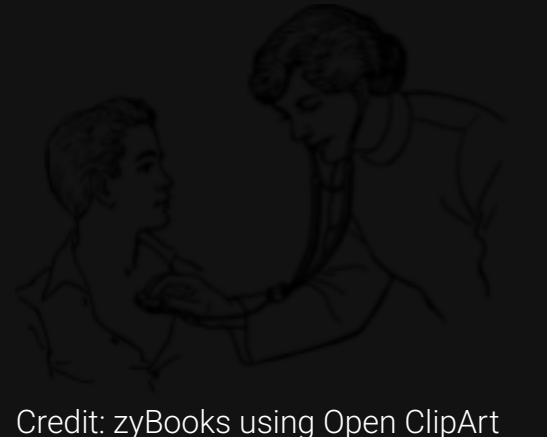
PA

Due: 02/06/2025, 11:59 PM EST

## 2.20 Debugging

**Debugging** is the process of determining and fixing the cause of a problem in a computer program. **Troubleshooting** is another word for debugging. Far from being an occasional nuisance, debugging is a core programmer task, like diagnosing is a core medical doctor task. Skill in carrying out a methodical debugging process can improve a programmer's productivity.

Figure 2.20.1: A methodical debugging process.



- Predict a possible cause of the problem
- Conduct a test to validate that cause
- Repeat

Credit: zyBooks using Open ClipArt

Feedback?

A common error among new programmers is to try to debug without a methodical process, instead staring at the program, or making random changes to see if the output is improved.

Consider a program that, given a circle's circumference, computes the circle's area. Below, the output area is clearly too large. In particular, if circumference is 10, then radius is  $10 / (2 * \text{PI\_VAL})$ , so about 1.6. The area is then  $\text{PI\_VAL} * 1.6 * 1.6$ , or about 8, but the program outputs about 775.

Figure 2.20.2: Circle area program: Problem detected.

```
#include <iostream>
using namespace std;

int main() {
    const double PI_VAL = 3.14159265;

    double circleRadius;
    double circleCircumference;
    double circleArea;

    cout << "Enter circumference: ";
    cin >> circleCircumference;

    circleRadius = circleCircumference / 2 * PI_VAL;
    circleArea = PI_VAL * circleRadius * circleRadius;

    cout << "Circle area is: " << circleArea << endl;

    return 0;
}
```

Enter circumference: 10

Circle area is: 775.157

Feedback?

First, a programmer may predict that the problem is a bad output statement. This prediction can be tested by adding the statement `circleArea = 999`; The output statement is OK, and the predicted problem is invalidated. Note that a temporary statement commonly has a 'FIXME' comment to remind the programmer to delete this statement.

Figure 2.20.3: Circle area program: Predict problem is bad output.

```
#include <iostream>
using namespace std;

int main() {
    const double PI_VAL = 3.14159265;

    double circleRadius;
    double circleCircumference;
    double circleArea;

    cout << "Enter circumference: ";
    cin >> circleCircumference;

    circleRadius = circleCircumference / 2 * PI_VAL;
    circleArea = PI_VAL * circleRadius * circleRadius;

    circleArea = 999; // FIXME delete
    cout << "Circle area is: " << circleArea << endl;

    return 0;
}
```

Enter circumference: 0

Circle area is: 999

Feedback?

Next, the programmer predicts the problem is a bad area computation. This prediction is tested by assigning the value 0.5 to radius and checking to see if the output is 0.7855 (which was computed by hand). The area computation is OK, and the predicted problem is invalidated. Note that the statement is again marked with a 'FIXME' comment to make clear it is temporary.

Figure 2.20.4: Circle area program: Predict problem is bad area computation.

```
#include <iostream>
using namespace std;

int main() {
    const double PI_VAL = 3.14159265;

    double circleRadius;
    double circleCircumference;
    double circleArea;

    cout << "Enter circumference: ";
    cin >> circleCircumference;

    circleRadius = circleCircumference / 2 * PI_VAL;

    circleRadius = 0.5; // FIXME delete
    circleArea = PI_VAL * circleRadius * circleRadius;

    cout << "Circle area is: " << circleArea << endl;

    return 0;
}
```

Enter circumference: 0

Circle area is: 0.785398

Feedback?

The programmer then predicts the problem is a bad radius computation. This prediction is tested by assigning PI\_VAL to the circumference, and checking to see if the radius is 0.5. The radius computation fails, and the prediction is likely validated. Note that unused code was temporarily commented out.

Figure 2.20.5: Circle area program: Predict problem is bad radius computation.

```
#include <iostream>
using namespace std;

int main() {
    const double PI_VAL = 3.14159265;

    double circleRadius;
    double circleCircumference;
    double circleArea;

    cout << "Enter circumference: ";
    cin >> circleCircumference;

    circleCircumference = PI_VAL; // FIXME delete
    circleRadius = circleCircumference / 2 * PI_VAL;
    cout << "Radius: " << circleRadius << endl; // FIXME delete

    /*
    circleArea = PI_VAL * circleRadius * circleRadius;

    cout << "Circle area is: " << circleArea << endl;
    */

    return 0;
}
```

Enter circumference: 0

Radius: 4.9348

Feedback?

The last test seems to validate that the problem is a bad radius computation. The programmer visually examines the expression for a circle's radius given the circumference, which looks fine at first glance. However, the programmer notices that `radius = circumference / 2 * PI_VAL`; should have been `radius = circumference / (2 * PI_VAL)`; The parentheses around the product in the denominator are necessary and represent the desired order of operations. Changing to `radius = circumference / (2 * PI_VAL)`; solves the problem.

The above example illustrates several common techniques used while testing to validate a predicted problem:

- Manually set a variable to a value.
- Insert print statements to observe variable values.
- Comment out unused code.
- Visually inspect the code (not every test requires modifying/running the code).

Statements inserted for debugging must be created and removed with care. A common error is to forget to remove a debug statement, such as a temporary statement that manually sets a variable to a value. Including a FIXME comment can help the programmer remember. Another common error is to use `/* */` to comment out code that itself contains `/* */` characters. The first `*/` ends the comment before intended, which usually yields a syntax error when the second `*/` is reached or sooner.

The predicted problem is commonly vague, such as "Something is wrong with the input values." Conducting a general test (like printing all input values) may give the programmer new ideas as to a more-specific predicted problems. The process is highly iterative—new tests may lead to new predicted problems. A programmer typically has a few initial predictions, and tests the most likely ones first.

zyDE 2.20.1: Debugging using a repeated two-step process.

Use the above repeating two-step process (predict problem, test to validate) to find the problem in the following code for the provided input.

Load default template...

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int sideLength;
6     int cubeVolume;
7
8     cout << "Enter cube's side length: " << endl;
9     cin >> sideLength;
10
11     cubeVolume = sideLength * sideLength * sideLength;
12
13     cout << "Cube's volume is: " << cubeVolume << endl;
14
15     return 0;
16 }
17
```

1500

Run

Feedback?

PARTICIPATION ACTIVITY

2.20.1: Debugging.

Answer based on the above discussion.

1) The first step in debugging is to make random changes to the code and see what happens.

True

False

2) A common predicted-problem testing approach is to insert print statements.

True

False

3) Variables in temporary statements can be written in uppercase, as in MYVAR = 999, to remind the programmer to remove them.

True

False

4) A programmer lists all possible predicted problems first, then runs tests to validate each.

True

False

5) Most beginning programmers naturally follow a methodical process.

True

False

Feedback?

How was this section?

Provide section feedback

Activity summary for assignment: CSC108 CH02.11-2.24 P2B

Due: 02/06/2025, 11:59 PM EST

This assignment's due date has passed. Activity will still be recorded, but will not count towards this assignment (unless the due date is changed). See [this article](#) for more info.

147 / 147 points

147 / 147 points submitted to BlackboardLearn

Completion details

2.21 Style guidelines