COSC 051: Debugging and Testing Tips

Jeremy Bolton, PhD
Teaching Professor

GEORGETOWN UNIVERSITY

Outline

- Debugging and Errors
- How to avoid errors / best practices
- Debugging techniques and methods

Errors

- There are various types of errors you may encounter when programming
  - Syntax errors: compiling errors
    - Syntax is incorrect
  - Run time errors:
    - Program execution could not complete due to issue not recognized during compilation
  - Logical errors:
    - Errors in design. The program runs to completion but does not have the desired result

How to avoid errors or weed them out almost immediately

- Debugging is an essential part of coding. Saving all debugging until the “end” will likely make the debugging process more difficult and more arduous. To save on debugging time (and headache): Follow a principled coding procedure.

- Modularize your code. Compile and test small pieces of code individually.

Best Practice

1. Start with empty main method, skeleton program.
2. Check if it Compiles
    1. Check if it runs (if appropriate)
3. Add 1 line of code (or a few)
4. Repeat steps 2 – 3 until complete

EG

Assume you are asked to

1. Read input
2. Compare value to valid inputs
3. Compute volume based on value

First Modularize and test. First write the code to simply read the input … only a few lines of code. Compile and test. Be sure it works. Use many print statements to assure the value stored is correct!!

```
int main(){
        string input;
        cin >> input;
        cout << “The value of input is  ” << input << endline;
// compare value to valid inputs
// compute volume
}
```

Error finding

- Finding syntax errors is generally easy.
- Finding runtime errors can be harder.
- Finding logical errors is generally hard.

Finding a runtime error can be project unto itself. It is best to perform a methodical search.

This will involve some problem solving and detective work. Make use of print statements throughout your code to confirm values of conditionals and other variables!

Detective Work

- The main difficulty in finding a logical error in a program is that you may not know the true values of variables , expressions, etc.
- Resolution: “Peek” into the program at each stage of execution
- You can peek using “cout”.
- Sequentially insert cout statements at every few lines of code and print out the values of the **variables** and **conditional** values.
- This will take time. Do not rush or become frustrated. It is best to simply realize this will take a large amount of time and allocate sufficient time for this process. **You do not want to be “in a rush” during this process**.

Error finding: Insert Print statements

```
int main(){
string input; string shape;

cin >> input;
cout << “The value of input is  ” << input << endline;

cout << “The value of input = c  is  ” << input = “c”) << endline;
if (input = “c”)
{
        cout << “Enter If Statement c  ”) << endline;
        shape = “Cone”;
}
cout << “Exit If Statement c  ”);
```

Error finding: Divide and conquer. The process of elimination

- If you are testing a large piece of code and are getting errors somewhere …
  - Sequential Elimination. **Isolate** small parts of your code and test, rather than testing all at once. Start at the beginning and move forward.
  - Attempt to **divide and conquer**. Rather than testing all of the code at once test smaller pieces of code one-at-a-time
    - For example, if your code consists of 10 major sections or portions. Comment out the last 5. Try to run and compile the first half of your code. If the first half does not run correctly … there is likely an error again. Divide and conquer again: Comment out 3 of the first 5 lines of code and repeat. Keep repeating until you have located the source of the error.
    - You may need to add print statements to help with debugging.

Examples:

```
int main() {
…….
<code>
…….
…….
<code>
…….
…….
<code>

return 0;
}
```

Coming up with a test plan

- Programs have inputs and outputs / side effects
- A program is logically correct if it provides the correct outputs / side effects for ***all possible inputs.***
- Cannot always check ALL possible input combinations – not practical
  - Test inputs that are most common or most expected
    - 1.0*rad <= height <= 1.5*rad
  - Try to “break” your program

Lecture Design

- Strategies
- Concepts & syntax
- Examples / in-class exercises