Debugging and common errors:
Picking up the pieces when something goes wrong

William Hendrix

Lecture 6

Outline

- Programming errors
- Common syntax errors
- `cout` debugging
- Using a debugger

Electroentomology

- Debugging is usually the most time-consuming step of program development
- Good coding practices can reduce the number of bugs you generate
- Bugs in code are virtually inevitable

- Two kinds of bugs
  - Syntax (compile-time) errors
    - Trying to compile code that is not correct C++
    - Compiler will try to generate a meaningful error message
  - Logic (runtime) errors
    - Valid C++ program that doesn't do what it's supposed to
    - Generally much more difficult to fix

Common syntax errors

- Expected ;
  - Compiler doesn't notice the problem until it hits the next line, so line number may be off
- Undefined symbol
  - You've usually misspelled a variable name or keyword, or you forgot to declare a variable
- Unbalanced parentheses/Expected)
  - Moving left-to-right, check if you ever have more) than (
  - If complex, consider splitting into multiple lines to reduce ()
- End-of-file found without matching brace
  - Forgetting an end brace can cause a lot of strange errors
  - Make sure that all of your indentation levels end with }

Logical errors

- Process of eliminating logical errors is called *debugging*

- Identify which variable(s) are incorrect
- Find when the variable(s) become incorrect for the first time
  - Start at beginning and trace code until incorrect, or
  - Check variable value(s) periodically and zero in on the bug
- Based on that line of code, figure why things are being computed incorrectly
  - If another variable is wrong, trace that variable until you find a mistake in your code
  - Debugging becomes much harder when code involves pointers (later) or multithreading or interprocess communication (another class)

Debugging example

```
int sum, count, num, value;
cout >> “How many numbers do you want to
enter?”;
cin >> count;
num = 0;
while (num < count)
  cout >> “Enter a number: ”;
  cin >> value;
  sum + value;
cout >> “The sum is”, sum;
```

Sample solution

```
int sum, count, num, value;
cout << "How many numbers do you want to
enter? ";
cin >> count;
sum = num = 0;
while (num < count)
{
  cout << "Enter a number: ";
  cin >> value;
  sum += value;
  num++;
}
cout << "The sum is " << sum << endl;
```

`cout` debugging

- Simplest way to debug
- Just add `cout` statements to check variable value at different points in program
- Example

```
int n, sum;
cout << "Enter n: "; cin >> n;
for (int i = 1; i <= n; i++)
{
   sum += i;
cout << i << ": " << sum << endl;
}
cout << "The sum 1 to " << n << " is "
     << sum << endl;
```

Warning: `cout` is buffered

- Output sent to `cout` is not printed immediately
  - Message is copied to temporary location in memory
    - *Buffer*: memory used to store messages temporarily
  - Displayed in console later

- A program error can stop your program after `cout`, but before output is printed

- Solution
  - `cout.flush();`
    - Forces console to output things now
  - `endl`
    - Flushes buffer after printing new line

Pros/Cons

Pros

+ Easy; doesn't require special tools
+ Can read through output with rerunning or rebuilding
+ Can combine with if statements to track down rare errors

Cons

– `cout` is buffered, so can produce confusing output
– Need to modify code if you're not looking at right variable or at right time
– Need to remove debug code before publishing/submitting
– Debuggers are almost always a better choice

Demo: debugging in Visual Studio

- Setting breakpoints and conditional breakpoints
- Step in, step over, continue
- Variable watches

Pros/Cons

Pros

+ Can look at many variables at once
+ No need to modify code

Cons

– Need to learn how to use
– Some code can be tricky to debug without conditional breakpoints
  – E.g., 1000th iteration of a loop

Tonight

Lab 1 is due Tuesday!

Recommended reading: Advanced Topic 2.2, Sections 2.4, 2.5, Appendix E (cmath and cstdlib)