THE UNIVERSITY OF ALABAMA AT BIRMINGHAM

CS330
Review and Performance

Spring 2022

Lab 13

Success Heuristics
or How to be successful in CS330 and have the most fun

- Start homework / labs as soon as possible
- Begin with design (this is hard since we all want to start with code)
  - Describe the problem
  - Break the problem into smaller pieces, solve, integrate as you go
  - Draw a picture
- Learn how to debug and test
- Use Labs and Office Hours to maximum advantage
- Code every day, it will get easier, we promise
- You (mostly) can’t break the computer, so experiment
  - If you’re curious about how/if something works, so are we, let’s break things together
- Ask questions
  - Please allow at least 24 hours for email replies
- Help us help you
  - Describe your problem (line number, compiler error messages), Describe the symptoms
  - Send a copy of your code
  - What have you tried so far to fix the problem? (So we don’t waste time on things that don’t work)
- Make a ‘safety’ submittal, sometimes Canvas crashes. No penalty for multiple submittals, we’ll grade the last one
- Have fun!

Anything we should add? Revise?

Draft Lab Outline (subject to change)

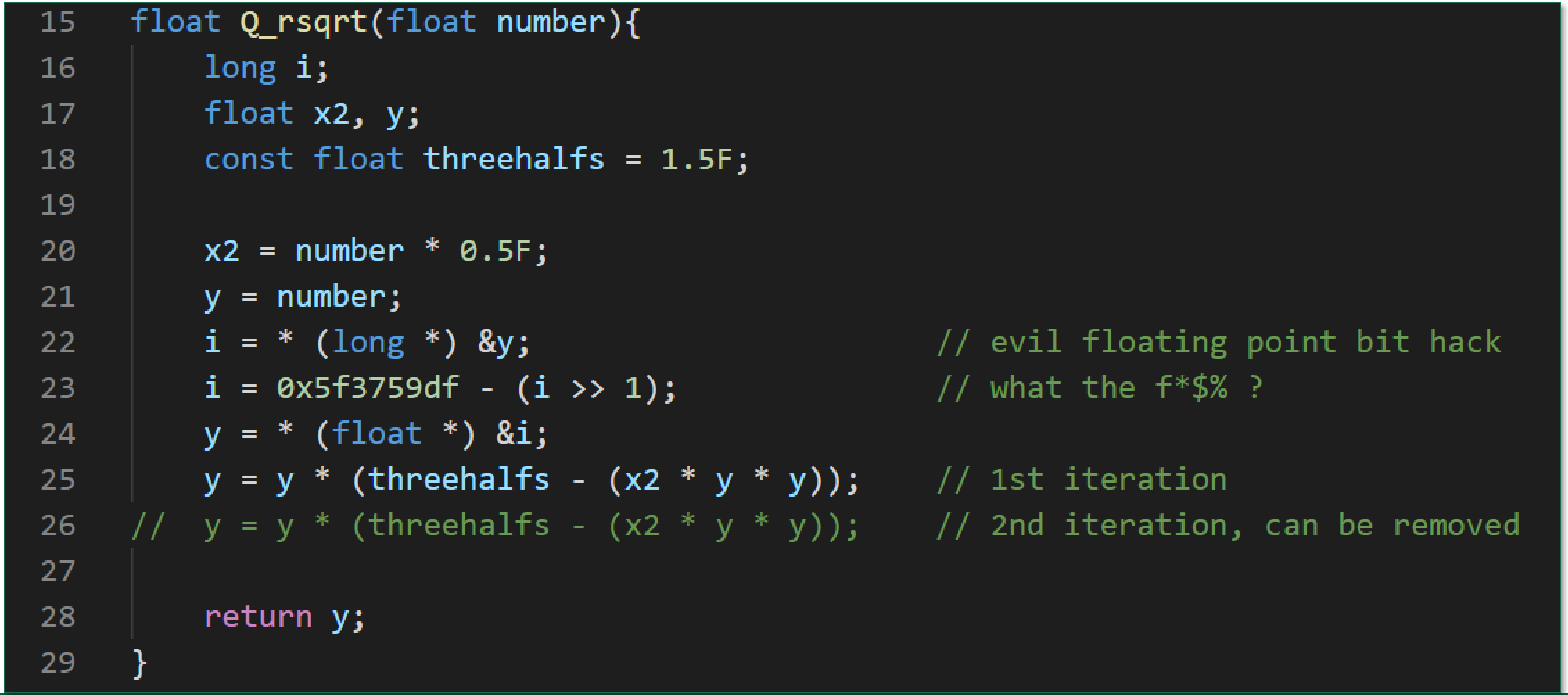
Things we'll learn:
C
Command Line interface
Bit math, Boolean algebra
Assembly
How computers work "behind the scenes"
How to improve performance and our higher level code

Match faces with names ! (Well, at least eyeballs, eyebrows, and beards)

Why CS330?
To understand something, it's often helpful to understand one level below
Helps us to optimize our code, understand 'why', make valuable stackoverflow comments

- Lab1: Env config, Hello World!
- Lab2: C: Intro, Make
- Lab3: C: Fib, Factorial, isPrime
- Lab4: C: pointers, GDB intro
- Lab5: C: Arrays, Strings
- Lab6: C: Debugging
- Lab7: Binary, Boolean algebra
- Lab8: C to Assembly
- Lab9: Two's Complement, IEEE574, Booth's
- Lab10: AS: Intro
- Lab11: AS: Arithmetic Ops
- Lab12: AS: Conditional / Unconditional Jumps
- Lab13: AS: Arrays, Functions

Let’s revisit the Quake Code



```
float Q_rsqrt(float number){
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y = number;
    i = * (long *) &y;                      // evil floating point bit hack
    i = 0x5f3759df - (i >> 1);              // what the f*$% ?
    y = * (float *) &i;
    y = y * (threehalfs - (x2 * y * y));    // 1st iteration
//  y = y * (threehalfs - (x2 * y * y));    // 2nd iteration, can be removed

    return y;
}
```

Performance Improvement Considerations

- Measure what's important: can't improve what we don't measure
- Pick the right tools and design
  - Language: see CS401
  - Algorithm, Data Structure: see CS303
- (Optional) Code Profilers for large code: e.g. gprof
  - Compile with –pg   gcc myfile.c –pg –o myfile
  - Run the program   ./myfile
  - gprof <program>   gprof myfile
- Compiler Optimization Flag: -O3
  - Reduce optimization blockers: Textbook, Chapter 5 items, and Lecture 17

Performance Improvement Considerations (cont'd)

- Code Motion
  - Focus on inner most loops first
  - Unless absolutely necessary, move code outside the loop
  - Don't forget about the loop evaluator: e.g. strlen(myString)
- Reduce Procedure Calls
- Eliminate unnecessary memory references
  - Use a temp variable to hold results, store result in array or global variable only when final value has been computed
- Low Level Optimizations
  - Loop Unrolling (Compiler will also attempt this)
  - And others: Chapter 5 items, and Lectures 17