

cs171: Introduction to Computer Science  
Writing and Debugging Programs

## 1 Motivation

Now that we are writing larger more complex programs we are finding it more difficult to figure out how to get started, to find errors in our program, to find errors in our thinking, to understand how the program is supposed to work. There are a number of proven techniques that can assist us in overcoming these difficulties. It may seem that this is just extra work that wastes time not spent programming. True to an extent; these techniques require extra time and effort that is not spent at the keyboard. But it has been universally proven that time spent with these techniques **reduces** the **overall** time to write a program. We spend slightly more time with an important technique but then save lots of time that we would have wasted attempting to make headway with a frustrating program or chasing down an illusive bug.

There are two major components: design and debugging. Design refers to what you do before you start writing code. It includes understanding the problem, sketching a solution, and maturely planning your workload to give yourself proper time to work on the program. Debugging refers to fixing a broken program after it has been typed. Debugging techniques usually help us resolve the difference between what we think our programs are doing and what they are actually doing.

## 2 Planning

The **single most important thing** about writing programs under a deadline is to start early! It cannot be stressed enough. If you have a week to write a program that takes three days to write, it is always infinitely better to spend the first three days than the last three days.

- There are so many more options available to us if we start early. These include things like seeking help from others, researching/reading sources, and the option of starting from scratch if we think we are chasing a dead end. Have you ever tried to seek help the night before something is due?
- We are under less stress and can think more clearly in a more relaxed framework.

- How often have we underestimated the amount of time it takes to complete a program? Starting early gives us flexibility to deal with longer-than expected programs.
- I know – you are too busy with other things to start early. Does it seem like you are scheduling your whole life according to the principles of procrastination. Of course, statistically there is no difference in the amount of other work you have to do early in a deadline than late in the deadline – you just feel like the program isn't a priority since it you still have time to get it done.

You probably already know that starting early is a good thing. Ultimately the extent to which you can force yourself to start early is a matter of personal maturity, an expression of your value system and a statement about the principles by which you live your life. In the programming industry, the best workers are the ones who manage their own schedules effectively and don't produce sloppy programs at the last minute or submit them late.

### **3 Design**

A good rule of thumb is to never touch the keyboard unless you have a complete design finished for your program. You should never create the design while typing – this will get you into trouble and waste your time. For very simple programs, you might be able to keep the whole design in your head and can type from there. But even modest programs require that you write your design on paper first and then type from the paper. The purpose of typing is merely to record statements that you already have planned out on your paper design.

An important technique is called top-down design. You take a big problem and break it into smaller pieces. You work on each piece individually and then assemble them all to compose the big program.

### **4 Know your stuff**

It goes without saying that you need to have a strong grasp of the basic language constructs. You need to know decisions, booleans, arithmetic, loops, etc. You need to fully understand how these things work so that you can immediately see how to apply them to the tasks at hand. It is always a good idea to keep a reference book (or API website)

## 5 Tracing a Program

A great debugging technique is to trace the execution of a program. You pretend to be the computer and execute each step, one at a time. You write the variables on paper and record their values after each step of the program. Be very careful to READ a program statement and follow it explicitly. The biggest mistake in using this technique is to ASSUME you know what a statement does and make the marks on your paper; but in fact, the statement does something different. Having a good paper record can aide in the next two techniques.

## 6 Println Debugging

The next step is to put `System.out.println` statements in your program. You can put them anywhere. A typical place is in a loop where you can print the loop variables and statements that are being operated on inside the loop body. Then you can "watch" the program process the loop as it prints things to the screen. This is sometimes the best way to resolve the difference between what you think your program is doing and what it actually is doing.

## 7 Debugger

Some environments have a built-in debugger. Dr. Java is one such editor. The debugger allows you to execute a program in slow motion, you can press a button to execute one statement at a time. In between, you can examine (and often change) variables and their values. You can literally watch as the program executes before your eyes.

In Dr. Java there is a Debug menu at the top. Select "Debugger Mode". Then you need to set a breakpoint in your program. This is a statement at which the computer will stop executing. You can set a break point by right-clicking (control-click on a single button mouse) on a statement at the beginning of your program. Then press F2 to begin execution. You should see that the program has executed up to (but not through) your breakpoint statement.

There are some buttons to the right that allow you to STEP THROUGH and STEP INTO a statement. If you step into a subroutine call, you will be temporarily whisked away from your code into someone else's; you will need to STEP OUT to return to your code again. In the WATCH window you can enter variable names to see their values. You can also type things in the interactions window to find values. Pressing CONTINUE will allow the program to finish quickly (unless it hits another break point).