The UNIVERSITY of VERMONT

Benjamin Fenster
CS-100 (Damon)
25 January 2003

Notes – Debugging

I. Process
   a. Somebody ran the program and got the "wrong" result
   b. That means you need a clear specification of what the "right" thing is.
      i. What does it do?
      ii. What are the valid/required inputs and resulting outputs and effects?
      iii. Define clearly the relationship between pre-and post-.
      iv. Specify for the whole program, whole class, and individual methods.
      v. Can specify in terms of other methods. (Eg: push() causes pop() to return the new value).
   c. Approaches to Solve
      i. Read the code! Broadly used, often preventatively.
      ii. Explain the problem to someone (anyone!) and it may become obvious to you.
      iii. Examine code as it runs using debugger tools.

II. Technique
   a. Start with a theory about what could cause the bug.
   b. If you don't have a theory, examine the state as it crashes.
   c. Attempt to disprove the theory by examining the state and flow.
   d. Too many or too few theories?
      i. Divide and conquer.
      ii. Does the problem occur before or after a certain point?
      iii. Is some particular part of memory affected?
      iv. "Wolf Fence" debugging – on which side of the fence is the wolf heard howling?
   e. Other Approaches
      i. Step through the program until something breaks.
         1. Very tedious for long programs and non-productive
         2. Great for short ones
      ii. Start where you know it's broken and work backwards until the source is revealed.

III. BlueJ's Debugger
   a. Quick to learn, easy to use, but not full featured.
   b. Capabilities
      i. Examine state ("inspect)"
      ii. Breakpoints
      iii. Step / Step Into
      iv. Continue / Halt
      v. Terminate
   c. Object Bench
      i. Store interesting objects that you create by running constructors directly (with a right-click)
      ii. Can call methods, examine data, et cetera.
      iii. Cannot store runtime objects here, unfortunately.