When Trouble Comes:
The Basics of Debugging

FIT 100

No one is capable of writing a flawless program of more than several lines on the first try. Therefore, algorithm design, programming and any other logical activity will require debugging or trouble shooting.

An Aside on Program Correctness

- One area of computer science research is proving programs correct. This is a difficult problem, and the techniques are almost never used in practice.
- We *can* prove mathematically that certain algorithms are correct. For example, we can prove that binary search will always find the desired value (and within a certain number of steps).
- A major problem with proofs of correctness: does the specification cover all the circumstances of use?

Testing

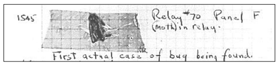
- In practice, what we must do is carefully define and refine the problem specification, use good programming style, and test. Testing:
  - By the developer
  - By testers (in large organizations)
  - Beta test
  - User feedback

Bugs vs Faults

- When the car doesn't start because of a dead battery, figuring out the problem uses debugging skills … but it is not technically debugging, but rather "fault identification"
  - When the error is a failing component of a correct design, it is a fault … when the battery is fixed the car runs
  - When the error is a failure of the design, it is a bug
- While programming the chances are overwhelming that the error is a bug, since you've likely made a reasoning error
- In "mature" systems it could be either one, since the error could be a fault or a latent logical error

The First Computer Bug Was A Moth

- The term "bug" for a computer glitch was coined by Adm. Grace Murray Hopper when working on the Harvard Mark II computer



The moth was found in Relay #70 -- an electro-mechanical switch -- and taped into the logbook with the caption "First actual case of a bug being found"

Debugging Programs into Existence

- Sometimes students in beginning programming classes try to debug their programs into existence.
  - It's gotta have an if statement…
  - and it's gotta have an assignment statement…
  - maybe I should try switching them and see if that works?
- This is a big mistake!

FIT 100 Guidelines For Debugging

- Try to avoid bugs by thinking through the design first.
- For big programs, test parts of it as you go.
- Some basic techniques:
  - Stepping by hand through the program
  - Adding “print” statements to get debugging output
  - Using the Visual Basic debugger
- Often the error will be blindingly obvious. If not, and you’re having to spend some time finding it, here are some additional suggestions:

FIT 100 Guidelines – Transient Errors?

1. Verify that the error is reproducible, i.e. make it happen again
   - “Transient errors” can occur
   - The error may have been caused by a state or configuration that was unknowingly set … get a “clean” instance of the bug
   - When reproducing the error, try to formulate a “minimal” version of the system or program with the bug

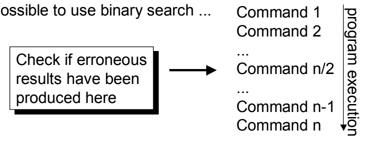
FIT 100 Guidelines -- Check obvious

2. Check for the “obvious” problems
   - Verify that the inputs are as required -- case, syntax, etc.
     - Are there 0-O 1-l I-l or other substitution mistakes
   - If there are multiple components or files in the buggy system, establish that these are properly “connected”
   - Has anything been changed recently
   - When there are multiple inputs, does the order matter
   - In programming, are all variables ...
     - Declared
     - Initialized

The chances that the problem is something “obvious” are small because if it were so “obvious” you would have already found the problem … but you must check

FIT 100 Guidelines -- Isolate error

3. Isolate the problem -- since the error is likely located in a specific place in the system or program, large sections of it are correct and should be removed from consideration
   - Isolating the problem to a specific procedure is best
   - Verifying that parts thought to be correct are correct is essential
   - It is even possible to use binary search ...



Check if erroneous results have been produced here

Command 1
Command 2
...
Command n/2
...
Command n-1
Command n

FIT 100 Guidelines -- Step through process

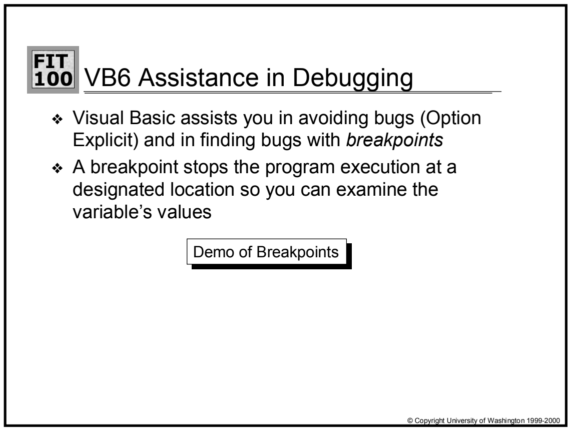
4. Once the error is isolated, reason through the process start-to-finish, predicting what should be computed and then verifying that it has been
   - When a prediction is inconsistent with an observation, the problem has been further isolated to the current step
     - The process was OK prior to this step
     - The process is incorrect after this step
   - Check the inputs and reason through the step
   - If bug not found, continue applying the guidelines

FIT 100 Guidelines -- Assess Objectively

5. It frequently occurs that everything checks out and is found to be OK … but the bug still persists

   Don’t become so frustrated that you stop thinking logically. Rather, evaluate your progress objectively: how are you doing?
   - Are you making a wrong assumption?
   - Do you misunderstand what the data means?
   - Have you made a wrong deduction?

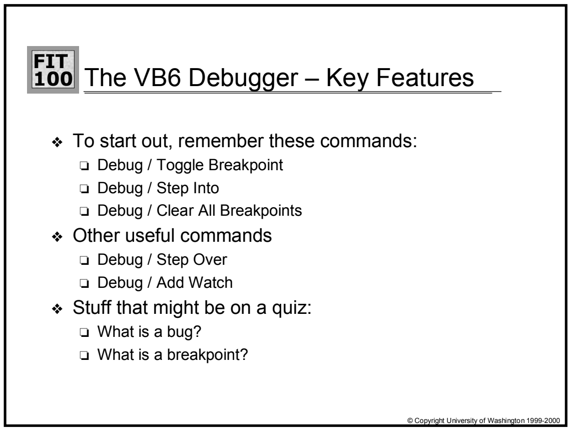
Remember … it’s a mystery and you are Jane Marple or Hercule Poirot … using those “gray cells” you can find the culprit



FIT 100 VB6 Assistance in Debugging

- ❖ Visual Basic assists you in avoiding bugs (Option Explicit) and in finding bugs with *breakpoints*
- ❖ A breakpoint stops the program execution at a designated location so you can examine the variable's values

Demo of Breakpoints



FIT 100 The VB6 Debugger – Key Features

- ❖ To start out, remember these commands:
  - ❑ Debug / Toggle Breakpoint
  - ❑ Debug / Step Into
  - ❑ Debug / Clear All Breakpoints
- ❖ Other useful commands
  - ❑ Debug / Step Over
  - ❑ Debug / Add Watch
- ❖ Stuff that might be on a quiz:
  - ❑ What is a bug?
  - ❑ What is a breakpoint?