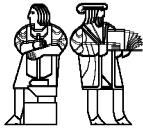


6.170 Lecture 5 Debugging



John Guttag
MIT EECS

Does My Program Work, Why Not

Validation (start in a couple of weeks)

Process designed to uncover problems, increase confidence
Combination of reasoning and test

Debugging

Ascertaining why a program is not functioning as intended
Function
Performance

Defensive programming (woven throughout term)

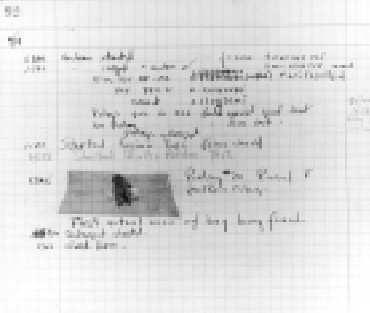
Programming to abet validation and debugging
E.g., exploiting compile-time type checking

© Michael Ernst/John Guttag

Spring 2001

Slide 2

A Bug: September 9, 1947



© Michael Ernst/John Guttag

Spring 2001

Slide 3

Some Myths About Bugs

Myth: Bugs crawl unbidden into programs

Fact: If there is a bug in your program, you put it there
I.e., you made a mistake

Myth: Bugs breed in programs

Fact: If there are many bugs, you put each of them there
I.e., you made many mistakes



© Michael Ernst/John Guttag

Spring 2001

Slide 4

A Matter of Attitude

When you find a bug in your code, feel relieved

Don't feel proud
Be at least a little embarrassed

When you run a program

Expect it to work
Debugging usually more time consuming than getting it
right in the first place

© Michael Ernst/John Guttag

Spring 2001

Slide 5

Debugging

Experience is clearly not the best teacher

Goal is not to eliminate one bug quickly

Goal is to move quickly towards a bug-free program

Key is to be systematic

When testing exposes a problem

Don't run off to the debugger

Study the program text

How could it have produced the result it did?
Is this part of a family of bugs?
How can it be fixed?

Using the debugger is a desperate measure

© Michael Ernst/John Guttag

Spring 2001

Slide 6

6.170 **When You Use Debugger**

Decide what you want to accomplish
Decide how you will accomplish it
Know what inputs you plan to use
Otherwise you will waste gobs of time
Know what outputs you expect
Otherwise you will be fooled

© Michael Ernst/John Guttag Spring 2001 Slide 7

6.170 **The Scientific Method**

Study available data
Test results
Program text
Keeping in mind that you don't understand it

Form a hypothesis consistent with all the data

Design and run a repeatable experiment
With potential to refute hypothesis
Often with useful intermediate results
With expected values
Repeatability often a challenge
Multi-threaded programs
Interactive programs

© Michael Ernst/John Guttag Spring 2001 Slide 8

6.170 **Designing the Experiment**

Find simplest input that will provoke bug
Usually not the input that revealed existence of bug

Start with data that revealed bug
Keep paring it down
Often leads directly to bug

Binary search is a good thing
On data
On program text

© Michael Ernst/John Guttag Spring 2001 Slide 9

6.170 **When You Have a Good Data Set**

Decide what intermediate states to look at
Predict intermediate values
If necessary, write code to display states
Don't throw this away after finding bug

Don't expect of find bug first time
Consider making check points
Especially when runs take a long time

© Michael Ernst/John Guttag Spring 2001 Slide 10

6.170 **Some Things to Keep in Mind**

The bug is not where you think it is

Ask yourself where it cannot be
Explain why

Try simple things first, e.g.,
Reversed order of arguments
Spelling of identifiers
Failure to reinitialize a variable
Same object vs. equal
Deep vs. shallow copy

Make sure that you have correct source code
Recompile everything

Keep a record of what you've tried

© Michael Ernst/John Guttag Spring 2001 Slide 11

6.170 **When the Going Gets Tough**

Reconsider assumptions
E.g., has the OS changed

Debug the code, not the comments

Start working on documentation
Way to approach things from a different angle

When it gets really tough, get help
We all develop blind spots
Explaining the problem often helps

Walk away
Trade latency for efficiency
One reason to start early

© Michael Ernst/John Guttag Spring 2001 Slide 12

6.170 When You Find A [sic] Bug

Ask how it got there

- Careless error
- Misunderstanding
 - Language
 - Requirements
 - Environment
- In attempt to fix something else
- Design vs. coding

Ask if the bug

- Is symptomatic of a larger problem
- Has relatives

© Michael Ernst/John Guttag Spring 2001 Slide 13

6.170 Fixing Bugs

Don't rush into anything
Haste makes waste

Does bug explain all observed symptoms?
If not, are other symptoms independent?

Should it be fixed in concert with other changes?
Think about Y2K remediation

What are the ramifications of the proposed "fix?"
Will it break other things?
Does it allow you tidy other things up?
Code should not always grow

Make sure that you can get back to where you are

© Michael Ernst/John Guttag Spring 2001 Slide 14

6.170 Key Concepts in Review

Testing and debugging are different
Testing reveals existence of bugs
Debugging pinpoints location of bugs

Goal is to get program to work
Not to find bugs

Debugging should be a systematic process
Use the "scientific method"

It's important to understand source of bugs
To decide on appropriate repair

© Michael Ernst/John Guttag Spring 2001 Slide 15