CSCI 1106
Lecture 10

Debugging

Announcements

• Today’s Topics
  – Motivation
  – Asking the right questions
  – Where to start
  – The “printf” method
  – Divide and conquer
The Bearable Heaviness of Bugs

- Fact: Most programs have bugs
  - Design flaws
  - Typos
  - Bad assumptions
- Fact: Bugs cause programs to misbehave
  - Crash
  - Have incorrect behaviour
  - Corrupt data
  - Can cause loss of life, limb, and property
- Fact: Buggy programs must be debugged (fixed)

This Program Does Not Work... Why?
Asking the Right Questions

• Why? Because the program has a bug...
• Assumption: Most of the program is correct
• Observation: The bug’s location is the point in the program where it starts to misbehave
• Conclusion: So, we ask where is the bug?

• Corollary 1: We ask when does the bug appear?
• Corollary 2: We ask how does the bug manifest?

The When and the How

• Question: Why do we care about
  – When the bug manifests?
  – How the bug manifests?
• Answer:
  – Programs are large and complicated
  – Want to restrict our bug search to part of the program
  – This makes debugging easier, but …
• Still need to find the bug
Where to Start ...

- Recall: We assume that program misbehaviour begins shortly after bug is encountered
- Goal: Narrow our search for the bug
- Idea: Determine the first instance of program misbehaviour

- So... where in the program do things go wrong?

Manifestation, Location, Location

- Idea:
  - Bugs manifest in program misbehaviour
  - Misbehaviour corresponds to a program location
  - Need to match the manifestation to the location
- To do:
  - Identify the bug manifestation
    - How do we know that something is wrong?
  - Identify the manifestation location
    - Where in the code does this something occur?
Bug Manifestation
This program stops while trying to find a lost line

Where in the code does it stop?

The “printf” Method

• We have two options:
  – Stare the code and guess at where the bug is
  – Use a mechanical procedure to narrow our search

• Goal:
  – Need to determine when we have reached specific locations in our program
  – Want the program to let us know when it has reached a specific location

• Idea:
  – “Print” to the screen when it has reached a given location in the program
Add Print Blocks

Run the program ...

Resulting Output when Program Stops
Deduction

- Our program printed out “wait” hence
  - The program is currently doing what?
  - Why is it doing this?
  - Is this correct?
  - Why or why not?
- Assume: Bug is near by (not always the case)

Where is the bug?
Drowning in Complexity

• Observations:
  – This is a simple program
  – Yet, debugging it was not easy
  – Imagine what happens with more complex programs

• Question: How do we debug large programs?
  – Sometimes bugs are not near their manifestation
  – We cannot put output blocks everywhere
    • Too much output
    • Takes too long to do
  – We need to be selective

• We need a debugging strategy!

Divide and Conquer

• Question: How do you search a phonebook?

• Idea: We can search a program for bugs in the same manner

• Observation:
  – Programs are linear entities
  – Programs comprise phases or stages

• Question: Does the bug occur before Stage 3?
Finding the Bug

Key Idea: The partitions are where you place print blocks

![Diagram showing stages 1 to 5 with partitions and bugs]

Question: What happens if the program cannot be subdivided further?

Discussion

- Debugging is an art, not a science
  - It’s hard to do
  - A little different each time
  - Requires you to solve many small problems
  - Can take a long time
- There is no silver bullet (no quick fix)
- There systematic approaches to ease debugging
  - Use output to identify location of bug manifestation
  - Use “divide and conquer” to narrow your search
  - Have someone look over your shoulder (really!)