CSCI 1108

Debugging
Bugs Suck (Mosquitoes too)

- Most programs have bugs
  - Design flaws
  - Typos
  - Bad assumptions
  - Logic and calculation errors

- Bugs cause programs to misbehave
  - Crash
  - Have incorrect behaviour
  - Corrupt data
  - Can cause loss of life, limb, and property

- Buggy programs must be debugged (fixed)
This Program Does Not Work... Why?

The robot is moving the distance $d=2$ in a given time interval. We want to calculate the position $x$ of the robot at each of the 10 intervals when the position at the first time interval is $x[0]=1$

```plaintext
var i
def x[10]=[0,0,0,0,0,0,0,0,0,0]
def var distance=2
def x[0]=1
def for i in 1:9 do
def x[1]=x[i-1]+distance
def end
[1,3,5,7,9,11,13,15,17,19] [1,2,0,0,0,0,0,0,0,0]
```
Asking the Right Questions

• Why is the program not working?
  – Because it 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?
  – When does the bug appear?
  – How does the bug manifest?
The When and the How

• **Question:** Why do we care about
  – *When* the bug appears?
  – *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, Match

• 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 fails to make the robot move forward after the robot starts to turn

• Where in the code does it fail?
How do we know what part of execution corresponds to what part of the program?
The “printf” Method

- **We have two options:**
  - Visually match code to execution (ok for small programs)
  - 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:**
  - Perform output when specific locations are reached
  - I.e., Turn on LEDs when our program reaches a set location
Add LED Activations

```plaintext
var min
var max
var mean
var state = STOPPED

call leds.circle(0,0,0,0,0,0,0,0)

onevent button.forward
    state = FORWARD
    motor.left.target = SPEED
    motor.right.target = SPEED
end

onevent button.backward
    state = STOPPED
    motor.left.target = 0
    motor.right.target = 0
end

onevent prox
    call math.stat( prox.horizontal[0:4],
                    min, max, mean )
    when STATE == FORWARD and max > THRESHOLD do
        state = TURN
        motor.left.target = -SPEED
    end

    when state == TURN and max <= THRESHOLD do
        call leds.circle(32,0,0,0,0,0,0,0)
        state = FORWARD
        call leds.circle(32,32,0,0,0,0,0,0)
        call leds.circle(32,32,32,0,0,0,0,0)
    end
```

- Use the circle of LEDs on top of the robot
  
  call leds.circle(a,b,c,d,e,f,g,h)

- Parameters range between 0 (off) and 32 (very bright)

- Run the program
The Result

```plaintext
var min
call leds.circle(0,0,0,0,0,0,0,0)

var max
onevent prox
    call math.stat( prox.horizontal[0:4],
                    min, max, mean )
when STATE == FORWARD and max > THRESHOLD do
    state = TURN
        motor.left.target = -SPEED
    end
when state == TURN and max <= THRESHOLD do
    call leds.circle(32,0,0,0,0,0,0,0)
    state = FORWARD
    call leds.circle(32,32,0,0,0,0,0,0)
    motor.right.target = SPEED
    call leds.circle(32,32,32,0,0,0,0,0)
end

var mean
var state = STOPPED

onevent button.forward
    state = FORWARD
    motor.left.target = SPEED
    motor.right.target = SPEED

onevent button.backward
    state = STOPPED
    motor.left.target = 0
    motor.right.target = 0
```

- Observation: The LEDs light up
- Therefore, the second when statement is being executed
- But the motors are not behaving correctly
- So the bug is likely in this part of the code
Deduction

• All three LEDs came on
  – Where in the program does this occur?
  – What else happens in the same part of the program?
  – Is this correct?
  – Why or why not?

• Assume: Bug is near by (not always the case)
Where is the Bug?

```plaintext
var min
var max
var mean
var state = STOPPED

call leds.circle(0,0,0,0,0,0,0,0)

onevent prox
    call math.stat( prox.horizontal[0:4],
                    min, max, mean )

when STATE == FORWARD and max > THRESHOLD do
    state = TURN
    motor.left.target = -SPEED
end

when state == TURN and max <= THRESHOLD do
    call leds.circle(32,0,0,0,0,0,0,0)
    state = FORWARD
    call leds.circle(32,32,0,0,0,0,0,0)
    motor.right.target = SPEED
    call leds.circle(32,32,32,0,0,0,0,0)
end

onevent button.forward
    state = FORWARD
    motor.left.target = SPEED
    motor.right.target = SPEED

onevent button.backward
    state = STOPPED
    motor.left.target = 0
    motor.right.target = 0
```

• Should be
  ```plaintext
  motor.left.target = SPEED
  ```

• Because the left motor was set to \(-SPEED\) earlier on
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 use LEDs everywhere
    • Too few LEDs
    • 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 (LEDs)

Stage 1  Stage 2  Stage 3  Stage 4  Stage 5

Stage 2a  Stage 2b  Stage 2c

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

```plaintext
var min
var max
var mean
var state = STOPPED

onevent button.forward
    state = FORWARD
    motor.left.target = SPEED
    motor.right.target = SPEED

onevent button.backward
    state = STOPPED
    motor.left.target = 0
    motor.right.target = 0

ever prox
    call math.stat( prox.horizontal[0:4],
            min, max, mean )

when STATE == FORWARD and max > 0 do
    state = TURN
    motor.left.target = -SPEED
end

when state == TURN and max <= 0 do
    state = FORWARD
    motor.right.target = SPEED
end
```
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!)
Debugging Rules of Thumb

• Use an output mechanism (such as LEDs) to locate the point in your program where the bug manifests
• Use divide and conquer to narrow your search in large programs
• Use as few LEDs as possible
• Compare closely your expectation with program output
• Have good luck