Computer Science I

Debugging

Dr. Chris Bourke
cbourke@cse.unl.edu

## Outline



# Part I: Introduction

## Debugging

- A *bug* is an error or flaw in a program or system that produces incorrect or unexpected results.
- Process of *debugging* involves *identifying* and *resolving* bugs so that the program/system performs as expected
- General techniques & strategies
- Demonstration of a *debugger*

## Types of Errors

- Syntax Errors
- Runtime Errors
- Logic Errors

## Poor Man's Debugging

- Up to now: *poor man's debuggin*
- Using print statements to determine value(s) of variables at different parts of a program
- Completely insufficient in practice
- Lots of manual time and effort
- Does not create reproducible tests/artifacts
- Extremely fragile and error prone
- Ad-hoc testing can serve a purpose but is similarly insufficient
- Even worse: blind coding, aimlessly changing things with not thought

## Debugging & Testing

- A bug indicates not only an error in your program but also
- that your *tests are insufficient*
- Unit tests & good code coverage should *prevent* bugs
- Debugging involves not only correcting the issue but also
- correcting the test suite by adding tests to cover the identified bug

*Learning and practicing proper testing is a result of accepting that you will not write perfect code.*

*Learning and practicing proper debugging is a result of accepting that you will not write perfect tests.*

## Debugging Strategies

General Debugging Strategies

- Understand the bug:
  - *How* is the program failing?
  - *What* inputs or conditions are causing the error
  - Is it a bug in our code or an error in our *understanding*
- Reproduce the bug:
  - Forces us to formally identify *correct behavior*
  - Design a (reusable) test case that highlights the bug
  - Gives us something to work with

## Debugging Strategies

- Isolate the bug
  - Rule out other issues: configuration, out-of-synch code, etc.
  - Reexamine assumptions & understanding
  - Narrow the failure point down to a testable unit (module/function/line)
- Investigate
  - Use your knowledge of the code to formulate a hypothesis about what is wrong
  - Test your hypothesis
  - Use a debugging tool to walk through your code
- Fix it, test it (and regression test it), document it

# Part II: Demonstration

## Debuggers

- A *debugger* is a program that simulates/runs another program and allows you to:
  - Pause and continue its execution
  - Set "break points" or conditions where the execution pauses so you can look at its state
  - View and "watch" variable values
  - Step through the program line-by-line (or instruction by instruction)
- GNU Debugger (gdb)

## GDB: Getting Started

- Compile for debugging:
  `gcc -Wall -g program.c`
- Preserves identifiers and symbols
- Start GDB:
  `gdb a.out`
- Optionally start with CLAs:
  `gdb --args a.out arg1 arg2`
- Can also set in GDB

## Useful GDB Commands

- Refresh the display: `refresh` (or control-L)
- Run your program: `run`
- See your code: `layout next`
- Set a break point: `break POINT`, can be a line number, function name, etc.
- Step: `next` (`n` for short)
- Continue (to next break point): `continue`
- Print a variable's value: `print VARIABLE`
- Print an array: `print *arr@len`
- Watch a variable for changes: `watch VARIABLE`

## Demonstration

Demonstration