Debugging

Programming Fundamentals

Dr Kevin Chalmers

School of Computing
Edinburgh Napier University
Edinburgh

k.chalmers@napier.ac.uk

Notes

Outline

Notes

References

- No specific book this week, however refer to the documentation of your favourite platform:
  - Visual Studio on Windows check out MSDN: `https://msdn.microsoft.com/`
  - Xcode or anything Apple: `http://developer.apple.com/`
  - Command line on Linux, check the `man` command, it supports the standard C/C++ functions.
- Good practice for the labs - before looking for solutions on Google / Stack Overflow, search and understand the documentation!
  - It should solve 99.99% of your problems.

Notes

Notes

Outline

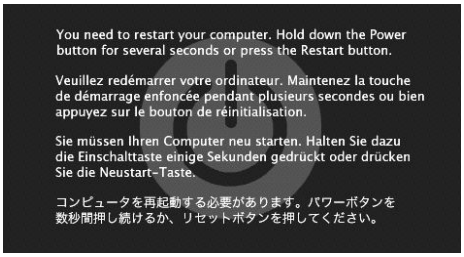
Notes

What is Debugging?

- Error is human. Most of you have encountered bugs and crashes when working with C and C++.
- Various software engineering processes exists in order to minimise the risk of errors and ensure software quality.
  - Some put a lot of emphasis on testing, e.g. Test Driven Development (TDD)
  - Others, like Extreme Programming, focus on feedback by having people working in pairs.

Notes

What is Debugging?



- Despite this, bugs still happen, for various reasons
  - Poor design, developer's attention, typos, etc.
- This is where debugging is useful.

Notes

What is Debugging?

- Debugging is the process of investigating and fixing programming errors
- The investigation can involve:
  - Getting your program to output some values to a console, a log file, or a bug/crash report for further investigation
  - Executing your program step-by-step to check its behaviour
  - Observing the values in the heap, stack, and registers

Notes

Question

Who knows where the term "bug" comes from?

Notes

Actual Bugs

- Legend has it that the term was coined from early electromechanical computers
- Insects would jam them and provoke errors
  - There is a logbook that has the statement "First instance of bug"
- This is one of the many recurring computing "anecdotes" whose sources are hard to verify
  - As an aside, the term "bug" was used at least as early as Thomas Edison in 1878 - long before electromechanical errors

Notes

Questions?

Notes

Outline

Notes

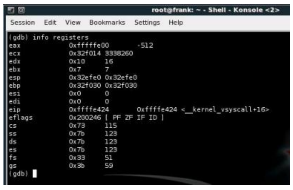
The Debugger

- The debugger is an application able to attach to a running process and analyse its inner workings
  - It can run step-by-step, observe variables and other values in memory
- We need to build our program using appropriate compiler or linker options to make it *easier*
  - This enables *debugger symbols*, allowing you to view your program as source code while debugging instead of machine/assembly code

Notes

Command Line Debugging

- These are tools available for command line debugging
- Each compiler tool chain will normally come with its own debugger
  - WinDBG for Windows/Visual Studio
  - gdb is the one provided with the GNU tool suite
  - lldb is the one provided with the clang/LLVM suite
- Typically developers will use an Integrated Development Environment (IDE) for debugging



Notes

Questions?

Notes

Outline

Notes

IDEs

- So far you have been working with basic tools
  - Text editor
  - Command line compiler and linker
  - Makefiles
  - A lot of frustration...
- As project complexity grows, you will find that you need to use more sophisticated tools
- An *Integrated Development Environment* (IDE) bundles up much of the tools we have been using, and more
- There are numerous IDEs available. However, there are a few that you are more likely to come across
  - Visual Studio (Windows)
  - Xcode (Mac OS X)
  - Eclipse (cross platform - very common for Java development)

Notes

IDEs

- IDEs are integrated solutions for the development, testing, and debugging of larger software projects
- They usually offer:
  - Support for different languages and Software Development Kits (SDKs)
  - Code editing, building, debugging
  - Graphical User Interface (GUI) editing tools
  - Unit testing tools
  - The more comprehensive IDEs also provide data modelling tools, web service tools, version control tools, etc.

Notes

Demo 1

Notes

Questions?

Notes

Outline

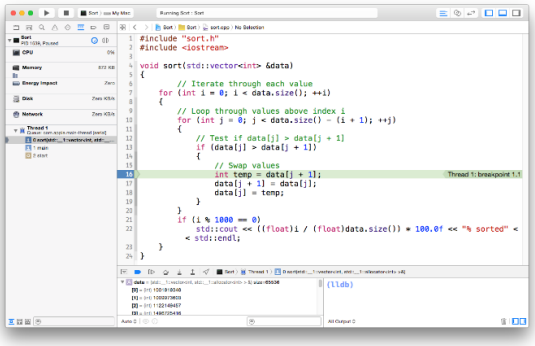
Notes

Breakpoints

- You can markup your code on certain lines to order the debugger to pause the execution of a program
- This is called a *breakpoint*
  - It is a point where the execution of the program is stopped (broken)
- Breakpoints can also be set to occur under certain conditions. These are sometimes called *Advanced Breakpoints* or *Conditional Breakpoints*

Notes

Example



A program stopped on a breakpoint in Xcode (Mac OS X). It is possible to view the values stored in the `vector<int>` at the bottom.

Notes

Step by Step

- Once a program has stopped at a breakpoint, it is possible to step through the code in order to check the flow of the application and the values of any in scope variables.
- Common commands in IDEs are:

  Step Into if the debugger is stopped on a function call, step into that function

  Step Over step to the next line, executing a complete function if currently stopped on a function call

  Step Out completes the current function and returns to the calling scope

Figure: Left to right: Xcode, Visual Studio, Eclipse

Notes

Assertions

- Assertions are a convenient way to end program and give feedback in case something invalid happens
- It uses the `assert(condition)` function where `condition` evaluates to true or false

  true program continues as normal

  false assertion error killing the program (unless exception handling is used - outside the scope of this module)
- For example, to avoid a division by zero `assert(denominator != 0)` could be used
- In this example the program would fail and output an error message such as (below for Mac OS X)
  - `Assertion failed: (denominator != 0), function main, file assert.c, line 10.`

Notes

Watching Variables

- The debugger views usually provide a representation of the heap and stack
- However, we can also add variables to a *watch list* to keep track of its value as the program executes
- A debugger will normally have some method to add a watch on a named value
- This is a very convenient method to keep track of a variables lifecycle

Notes

Demo 2

Notes

Advanced Functions

- Debugging allows us to do more advanced tasks when investigating issues
- Most low level debuggers will allow you to perform the following actions
  - Analyse the current *call stack* i.e. the trace of currently called functions
  - Using the disassembly to check the individual assembly instructions being executed by the CPU
  - Examine the memory
  - Examine the CPU registers
- These are very powerful features for exploring a running application, although they don't make understanding a problem necessarily easy
  - It is a machine eyed view of what is happening rather than a source code view

Notes

Demo 3

Notes

Questions?

Notes

Outline

Notes

Summary

- Debugging is an essential step in the life cycle of software development
- Debuggers are tools allowing the step-by-step execution of programs and the analysis of variables, memory, and registers
- Most IDEs shop with a debugging toolset
- Debugging is a useful and practical skill. You will need to experiment and practice it a lot to become familiar with techniques
  - The reason you might see lecturers being able to spot bugs quickly is because they know how to debug and where problems are likely to occur. You will become *far more* productive if you spend your time debugging.

Notes

To do...

- In the lab: debugging practice using Visual Studio.
  - Probably the most important skill to practice in the entire module.
- Coursework 2 should appear on Moodle by the end of the week.
- Next week - introduction to object orientation.

Notes