Beyond the Mouse – A Short Course on Programming

10a. Backup and Debugging Solving Major (and minor) Crises

Ronni Grapenthin

Geophysical Institute, University of Alaska Fairbanks

November 13, 2011

YOU'LL NEVER FIND A PROGRAMMING LANGUAGE THAT FREES YOU FROM THE BURDEN OF CLARIFYING YOUR IDEAS.
BUT I KNOW WHAT I MEAN!

"The Uncomfortable Truths Well", http://xkcd.com/568 (April 13, 2009)

Today's schedule . . .

1. Backup Strategies
2. Debugging

What is a *backup*?

Backup, backup!

- Creating a copy of something that must never get lost.
- data, results, settings, figures, writing (YOUR THESIS), . . .
- . . . because hard drives sometimes die, laptops get lost, houses burn down, etc.

What is a *backup*?

Backup, backup!

- Creating a copy of something that must never get lost.
- data, results, settings, figures, writing (YOUR THESIS), . . .
- . . . because hard drives sometimes die, laptops get lost, houses burn down, etc.

General strategies

- Episodically create a physical copy on a medium different from primary hard drive (e.g. usb drive).

What is a *backup*?

Backup, backup!

- Creating a copy of something that must never get lost.
- data, results, settings, figures, writing (YOUR THESIS), ...
- ...because hard drives sometimes die, laptops get lost, houses burn down, etc.

General strategies

- Episodically create a physical copy on a medium different from primary hard drive (e.g. usb drive).
- Use one of the gazillion tools that help you with this.

What is a *backup*?

Backup, backup!

- Creating a copy of something that must never get lost.
- data, results, settings, figures, writing (YOUR THESIS), . . .
- . . . because hard drives sometimes die, laptops get lost, houses burn down, etc.

General strategies

- Episodically create a physical copy on a medium different from primary hard drive (e.g. usb drive).
- Use one of the gazillion tools that help you with this.

What is a *backup*?

Backup, backup!

- Creating a copy of something that must never get lost.
- data, results, settings, figures, writing (YOUR THESIS), ...
- ...because hard drives sometimes die, laptops get lost, houses burn down, etc.

General strategies

- Episodically create a physical copy on a medium different from primary hard drive (e.g. usb drive).
- Use one of the gazillion tools that help you with this.
- Whatever method you choose, make sure the files can indeed be recovered (i.e. test the backup)

Review: Software Development Cycle

1. Design
2. Coding
3. Test
4. **Debugging**
5. go back to 1,2, or 3, . . .

What is “debugging”?

Debugging is the **art** of finding and fixing mistakes in computer programs. To be successful you need insight, creativity, logic, and determination.

What is “debugging”?

Debugging is the **art** of finding and fixing mistakes in computer programs. To be successful you need insight, creativity, logic, and determination.

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

Brian Kernighan

Truths about bugs and debugging . . .

- Bugs are static – they won’t run away.
- Often, the problem is **simple**.
- You created the bug! It’s nobody else’s fault - suck it up!
- Debugging is a great way to learn being self-critical. Good luck!
- Be critical – did you mean ’<’, ’<=’, ’>’, ’>=’?
- Don’t panic – be systematic!
- Sleep, go for a walk, come back later.

Debugging Styles

- **echoing**: place print statements at useful points in a program (function entry, exit)
- **unit testing**: write calls to particular function, throw artificial values at it
- **exception handling**: in high level languages: sources of mistakes easier to spot
- **online debuggers**: for our purposes not necessary, useful if you want to step through your code, or for memory problems
- **version control**: have a tool keep track of changes you make; roll back to bug-free code is simple (not covered here)

Debugging Styles: echoing

. . . we find stepping through a program less productive than thinking harder and **adding output statements and self-checking code at critical places**. *Clicking over statements takes longer than scanning the output of judiciously-placed displays. It takes less time to decide where to put print statements than to single-step to the critical section of code, even assuming we know where that is. More important,* **debugging statements stay with the program; debugging sessions are transient**.

From: Brian Kernighan, Rob Pike "The Practice of Programming"

Debugging Styles: echoing

. . . we find stepping through a program less productive than thinking harder and **adding output statements and self-checking code at critical places***. Clicking over statements takes longer than scanning the output of judiciously-placed displays. It takes less time to decide where to put print statements than to single-step to the critical section of code, even assuming we know where that is. More important,* **debugging statements stay with the program; debugging sessions are transient***.*

From: Brian Kernighan, Rob Pike "The Practice of Programming"

- write method that displays text only if a global DEBUG flag is set
- find ways to implement such external switches – for SHELL: environment vars, Matlab: create your own preferences
- call this method whenever necessary: entry, exit of functions, to display certain values, to follow the program flow, . . .

Debugging Styles: echoing

…see `t_debug` demo …

Debugging Styles: unit testing

- at the simplest: write calls to your functions with artificial values
- execute these calls at the beginning of your code, check function results
- this helps to detect errors due to changes in functions immediately
- also: assertion that function works for tested TYPES
- can be done for any language (some languages come with fancy frameworks)

Debugging Styles: exception handling

Full exception handling support in Matlab:

Matlab – `try-catch`

```
% try , STATEMENT, catch ME, STATEMENT, end.
% EXAMPLE: file opening
clc;
try
    fid  = fopen('whatever.txt', 'r'); % open a non—existing file
    data = fread(fid);                  % now try to get its data
    fclose(fid)
catch myException                       % define any name for an error message object
   %let the user know, implement graceful program termination ... write to stderror
    fprintf(1, '??? Error using ==> fread\n\n')     % recreate Matlab error message
    fprintf(1, '%s\n', myException.message);        % actual message from error message object
    fprintf(1, 'Error in ==> %s at %d\n\n\n', ...   % where did things occur?
                      myException.stack.name, myException.stack.line);

    fprintf(1, 'Simpler:\n')                        % use internal function to get Matlab
    fprintf(1, '%s\n', getReport(myException));     % style report
end

disp('———————> We do get here!'), pause

%now without try—catch ...
fid  = fopen('whatever.txt', 'r');
data = fread(fid);

disp('We cannot get here!')              % We'll only make it here if 'whatever.txt' exists!
```