

Hints for Solving Large Problems

Or, How to get started on Prog1

Benjamin Brewster

Adapted from slides by Jon Herlocker, OSU

The Challenge

- You have been asked to write a piece of software
 - You have been given specifications
- You are not sure where to start
 - Perhaps you don't understand the underlying technology
 - Perhaps you don't understand all the specifications
 - Perhaps you are not sure how different pieces of the component work together
 - above == overwhelming

Advice to Follow

- The following advice applies to every single software project that you will encounter in your life!
- And possibly much more than software

Making a Large Problem Seem Easy

- DIVIDE AND CONQUER
- Keep decomposing (breaking into smaller pieces) the problem until you reach something that:
 - You already know how to do
 - Looks like it will be easy

Example: Prog1

- First, focus on the core task
 - Basic task: compute statistics
 - Ignore the rest (handling signals, etc)
- Break the task into pieces
 - Need to compute statistics on rows, and on columns
- Focus on one of the sub-pieces:
 - How do we compute stats for rows?

Example: Prog1

- Still not sure what to do, so break it down further
 - Let's try and compute stats for one single row
- Further
 - We need to read one line from a file
 - Then we need to sum the numbers in the line
 - Then we need to divide by the number of numbers on the line.
 - Then we need to print out the result
- Further
 - How do we read one line from a file?
- Now this doesn't seem so daunting.
 - Read through all of the sources available to you, this time looking for a specific solution to reading a single line

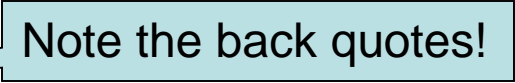
Reading with a Mission

- Re-read technical reference documentation once you have a compact problem to solve
 - You'll find that you'll pay closer attention
 - Man pages finally have a use!
 - You'll remember more of what you read
 - You'll find the answers to your problem
 - Before, you didn't have a goal, so your brain didn't bother to internalize or remember the information

Back to the Example

- You figure out how to read a single line
 - `read X`
- You test your knowledge
 - Write a tiny shell script to test it
 - You also figure out how to print the line to the screen to test it.
- Next Step:
 - Use your line read routine
 - Sum the numbers on the line
- Try handling the simple case first
 - Add two numbers on the line
- Decompose
 - Forget the line, how do I add two numbers?
 - Then worry about getting those numbers from the line

The Example

- You read the docs, and figure out how to add two numbers and store them in a variable
 - You find it faster this time:
 - `sum=`expr 3 + 4``
- 
- Note the back quotes!
- Now, how to take those numbers from the line you read
 - A new problem!
 - `read X` reads the entire line into a variable - how do you break it up?
 - Back to the docs
 - `read X1 X2` will work
 - Success again!
 - `read X1 X2`
 - `sum=`expr $X1 + $X2``
 - `echo $sum`

The Example

- Now a new challenge: what if you don't know how many numbers are on a line?
 - Back to the documentation
 - But this time, you are stumped.
 - You ask a friend
- How else can we do things with a non-static number of elements?

Problem Solving Summary

- Break down the problem until you have a problem that looks solvable
- Then hit the docs hard to find the solution
 - You will learn much more from the docs, if you are reading with a well-defined problem
- Solve the tiny problem
 - Make sure that you write the code!
 - Feel good about the success – have a donut:
- Now work upwards
 - Integrate your solution into the (slightly) larger problem.
- Repeat!



Tricky bits

- Each process has it's own identification number called the process id, or `pid`
- When you initiate a shell script, a pid is generated for the new instance of the called shell
 - `% /bin/sh ./myRadShellScript`
- What is the pid of this shell?
 - the variable `$` holds the pid of the current process in a shell script
 - `$$`
 - In C, you call a function
 - `int getpid()`

Tricky bits

- Space nastiness:

```
% expr 5 \* \ ( 4 + 2 \)
```

```
30
```

```
% expr 5 \* \ (4 + 2\)
```

```
expr: non-numeric argument
```

2>&1 in the bash shell

- Direct stderr to the same place as stdout
 - `% /bin/sh ./stats -rows test_file 2>&1`
- Direct only stderr to errorFile
 - `% /bin/sh ./stats -rows test_file 2> errors`
- Append both stdout and stderr to file: output
 - `% /bin/sh ./stats -rows test_file >> output 2>&1`