Developing Course Material - Labs

“Scripted” Labs, in particular

Mary Cryan

School of Informatics
University of Edinburgh
Computer Programming - Skills and Concepts

- Programming Language is C
- Designed for non-Informatics students (often Sci& Eng)
- Starts from 0.
- 3 lectures, 1 tutorial, one 2-hour lab each week.
- Assessed by a 3-hour online programming exam.

We re-vamped the course in 2011/12. Main changes were
(i) Switch to an online exam which would test real
programming skills. (ii) Introduce a 2-hour lab each week
where students would work through “scripted” labsheets
under supervision of lecturers and/or demonstrators.
The 2-hour Labs

- Labs of 40-50 students on DICE machines, with 2-3 demonstrators present (often including one lecturer).
- Not “worth” anything in terms of course assessment, but valuable for training students to write code “live”.
- Content closely following material from recent lectures, so aim was
  - To give programming practice, but also . . .
  - To reinforce/illustrate concepts recently taught in class.
- Template files often provided in the early weeks of the labs.
Specific 2-hour Labs

Lab Sheet 2
Simple arithmetic programs: Using the terminal window (and basic Linux), making a `template.c` file, working with variables, simple arithmetic operations, `float` and `int`, using `printf` and `scanf`, compiling and testing, the `if`-statement.

Lab Sheet 3
Programming with iteration: More programming tasks, “program planning” with reference to the “clock” problem, complex boolean conditions, iteration with `for` and `while`, reinforcement of `scanf` and `printf` details.

Lab Sheet 5
Functions, pointers, and some arrays: `rotate` with array as parameter, `swapwrong` and `swap` and testing these two (already seen in class), call by value, pointers, printing from inside a function (to examine behaviour), printing out pointers/addresses.

Developing Course Materials (2017/18) – slide 4
Key points when developing scripts/specifications

- Make the specification **precise** in terms of problem details, function prototypes (when writing code), input/output behaviour (or format of results).

- (for Labs) Remind the student of **key points** they will need to use to solve the problem (*maybe not always for Coursework*).

- Give **example input/outputs** to allow students to test their solutions.

- **Schedule carefully** against lecture content.

- Document structure (writing + “sectioning”), spelling, grammar.
To be improved?

➤ Automated testing: *now done*. There no automated testing-service for labs when I was teaching the course; it was introduced in 2016/17 (definite improvement over students typing in test input).

➤ Too-long descriptions?: labsheets are very well-structured with very clear reminders of what-is-important for each task. However, *often* the students’ eyes skated over these details. Maybe some chopping needs to be done . . .