University of Stirling

Computing Science and Mathematics

CSCU9A2 Tutorial 7 – sample answers Spring 2017

(for week starting 20th March)

1. Besides thinking about what is directly relevant to the topic, it is really necessary to go through all the list of things in the "Know Thy User" lecture notes, and check to see whether any of them apply. Hopefully the main point to get out of this is to see that if you do go through the exercise, you do actually get a lot of useful design information, which you can then use when you're designing the thing (i.e. this exercise is the first part of the design phase!)

   Here is the *kind* of outcome that should come from the analysis:

   (a) The points you might make about the visitor to such a web site is that they are probably interested in wool sweaters and/or the Isle of Skye (since there must be some reason why they're visiting it in the first place!).

   Typically the people who buy from such places are tourists to Skye, so the site should be aimed at that sort of tourist market.

   Age? Mostly adults. Affect the design? Yes, adults have more sophisticated tastes so you'd aim the site generally at adults, not a "Disney" version!

   Gender? Could easily be either. Doesn't affect the design more than to make it gender-neutral.

   Education/training/experience? Just what is typical of web users.

   Motivation/goals? They might want to buy a jumper, they might want to do so online, or they might want to do so in person, they might want to visit Skye.

   Language/cultural background/nationality? Despite the Hebrides also having Gaelic as a language used, that's not so likely to be important here as it's the tourist market being targetted. The site should definitely be in English, but could also consider having it translated into other languages (depending on which languages are most common amongst those visiting Skye)

   Disabled? Yes, certainly a visitor could be disabled, visually, aurally, physically or mentally, they all might want a woolly jumper! Effects on design should include good accessibility (readable in a text-only browser, no sound effects that can't be missed, etc.)

   (b) Generally, you can say that they are probably going to be shoppers in the mall. Can't even say that for certain (e.g. security guards, groups of bored teenagers hanging around), though.

   Age? All ages. This might have influenced the design by making it a touch screen in the first place. With children running around, it's a good idea if the physical equipment isn't easily breakable. Gender? Mixed.

   Education/training? Varied. So must be suitable for people who haven't used computers, who aren't very familiar with technology.

Motivation/goals? They're probably motivated by wanting to learn more about the shops, maybe they want to get to a particular shop or know which shop sells a certain item.

Language/cultural background/nationality? Vast majority Scottish, English-spoken. If the mall wanted to attract tourists then it could certainly cater to other languages too.

Experience? Familiarity with shopping, probably, but not necessarily that mall - good idea to include a map of the mall?

Left/right handed? Most will be right handed. Although the layout might be slightly more convenient for a right-handed person (e.g. by having buttons on the rhs of the display), it shouldn't be overly so.

Disabled? (visual, hearing, physical, mental) Certainly. Disabled people shop too. So the touchscreen shouldn't be too high up so that people in wheelchairs can use it, it shouldn't overly rely on voice (so it won't be a problem for deaf people), it should also be accessible to people with visual problems, e.g. large size display, no colour-blind problematic colour choices, audio features, maybe braille tells you how to operate the features.

(c) Age? Mostly 5-7 year olds (though adults will probably be users in so far as helping the child to use it), although there could be older people with lower learning/ability levels using it.

Gender? either

Training/Experience? 5-7 year olds may or may not have had much experience with a computer. They may, for example, know how to use the mouse, but not how to type. So in the design, be careful not to use much that requires typing, or minimize it (unless it's a puzzle about typing! e.g. find-the-letter)

Motivation/goals? Probably to have fun. Probably to solve the puzzle. So it's got to be fun to do, and the puzzle has to be of the right level of difficulty, not too easy, not too hard.

Language/cultural background/nationality? English/British, unless the company is aiming for an overseas market.

Left/right handed? Could be either. Designer should not make it difficult to operate with either preferred hand.

Disabled? (visual, hearing, physical, mental) Yes, could be. If it's a visual puzzle, then the blind aren't going to use it, but the partially sighted/colour blind could, so keep things big on the screen, choose good colour choices, have sound feedback to help. Deaf children might use it though, so don't rely on sound being heard. Also disabled children might want to use it, so try and make operation as simple as possible, and as easy as possible e.g. for any child who has difficulty using a keyboard or fine motor control with a mouse.

2. *Note: This is the partitioning step in Quicksort. Quicksort has been in a practical, but has not been discussed in lectures.*

   (a) A reasonably challenging exercise tracing another array manipulation algorithm!

   `pivot` simply holds one value, a copy of either the middle index element of the array, or the one just below the middle. In this case element index 2: `21`. It never changes (no assignments to it in the code). `pivot` is `21`

   `i` and `j` hold `int`s that, in effect, point to elements of the array. In the diagrams below, the boxes for the memory location for `i` and `j` are positioned below the indicated array element to emphasise this.

   Tracing table on next page...

   (b) This has rearranged the values in the array so that everything below (to the left of) the pivot is <= the pivot, and everything above (to the right of) the pivot is >= the pivot – so the two partitions are numerically non-overlapping sets of values (except for maybe duplicated pivot values). [Quicksort then recursively sorts each "half" – though, as here, they might not be exact halves.]

   This algorithm is O(n) – it jumps around, but only visits each value once or twice.

   (c) If the pivot is the median value, then the two "halves" will be as close as possible to being an equal split of the work in the two recursive calls – which is when the divide-and-conquer approach works best. [If we used a fixed way to pick a pivot, eg first or middle, then it is *possible* that it will turn out by accident that *all* the other values *always* fall into one half or the other. QS then degenerates to its worst case, which is $O(n^2)$ ☹. Merge sort does *not* have this weakness, but does require additional storage whereas QS rearranges in-place.]

   Algorithm to find the median of an array.... for discussion: how to do it, and what its performance is!

		Array index									
(Line no) **Note**	**Test result**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	
Initially		4 i = 0	15	99	6	21	12	88	67	3 j = 8	
(3) At while loop: 0 <= 8	true										
(5) At while loop: 4 < 21	true										Scan for value >= pivot
(7) Adjust i			i = 1								
(5) At while loop: 15 < 21	true										
(7) Adjust i				i = 2							
(5) At while loop: 99 < 21	false										
(9) At while loop: 3 > 21	false										Scan for <= pivot
(13) if: 2 <= 8	true										
(14-17) Swap elements		4	15	3 i = 2	6	21	12	88	67	99 j = 8	Swap misplaced values
(18-19) Adjust i, j					i = 3				j = 7		
(3) At while loop: 3 <= 7	true										Scan for value >= pivot
(5) At while loop: 6 < 21	true										
(7) Adjust i						i = 4					
(5) At while loop: 21 < 21	false										
(9) At while loop: 67 > 21	true										Scan for value <= pivot
(11) Adjust j								j = 6			
(9) At while loop: 88 > 21	true										
(11) Adjust j							j = 5				
(9) At while loop: 12 > 21	false										
(13) if: 4 <= 5	true										
(14-17) Swap elements		4	15	3	6	12 i = 4	21 j = 5	88	67	99	Swap misplaced values
(18-19) Adjust i, j						j = 4	i = 5				
(3) At while loop: 5 <= 4	false										
Exit main loop and finish											

SBJ/DEC March 2017