Computing for Engineers (EE108)

Assignment 3 – array pointers, output parameters, double-clicks

Objectives: This assignment builds on laboratories 6 and 7 and aims to further develop the use of array pointers and output parameters.

Learning outcomes:

- Operate on arrays using pointers and pointer arithmetic
- Detect single and double clicks and various click durations
- Manage a last in first out buffer (based on an array)

Assignment instructions

1. The assignment is for individuals working independently and builds on the lab exercises
2. Create a new sketches as required, naming them as specified in each section.
3. Create a single plain text submission file (.txt) for the assignment using a plain text editor (e.g. NotePad++). Copy all the sketches you write and any other answers required for the assignment into the submission text file. *Name the file "108_A3_firstname_surname.txt", using your actual name. Put your name and assignment number at the top of the submission file also. Clearly label everything in the file. Submissions without names or clearly labelled sections will have to be marked down or (in the worst case) not marked at all.*
4. [redacted] Upload a zip file of your sketches using the same naming convention as the text file, but with a .zip extension.

Marking for lab/assignment

Most of the assignment will be marked in the following lab session. It is essential that you demonstrate all sketches you have completed or partially completed.

For all code sections, marks will be deducted for bad structure, communication and style (e.g. repetitive or highly inefficient code, inappropriate global variables, missing or mismatching comments, poor variable names, bad indentation, etc.), incorrect behaviour, and failure to compile.

General marks will also be lost if the submission document requirements are not followed

NOTE: plagiarism and collusion will not be tolerated. You may be interviewed in detail on any work submitted and any evidence that the work is not your own will have the appropriate consequences.

1 Arrays, pointers, and clicks – save sketch as "A3_ClickArrays"

Background: In this exercise you will work with arrays and array pointers and functions that iterate over arrays using pointers. You will also use a function in the EE108 library that you have not used before to help you detect various click durations and potentially double-clicks. The reason you have not used this function before is because it requires a (pointer) output parameter.

The program has 2 arrays and will operate as follows. Whenever SW1 is normally clicked the two arrays will be printed. When SW1 is long-clicked, array1 will be updated with random values. Whenever SW1 is very-long-clicked, array1 will be updated with zeros. Finally, whenever SW1 is double-clicked, the contents of array1 will be copied into array2. The contents of both arrays must be printed after any of the above changes takes place.

Click guidelines:

- *normal click*: pressed for less than 1 second
- *long click*: pressed for 1-3 seconds
- *very long click*: pressed for more than 3 seconds
- *double-click*: two complete clicks detected in less than 1 second.

There is a function in the EE108 library, readSwitchEventTimes, which is very like the readSwitchEvent function that you have already used. (Refer to the ee108_switches.h header file.) The main differences are as follows:

- readSwitchEventTimes does not distinguish between click types (normal, long) and instead returns SW_CLICK in the case that any click is detected (regardless of duration).
- readSwitchEventTimes has an optional output (unsigned int pointer) parameter.
  - The optional output parameter, pPrevStateDurationMs, receives the duration the switch was in the state up to (but not including) the time readSwitchEventTimes was called. This is explained in the comments in "ee108_switches.h", but essentially, if the switch has just been released, i.e. the return value is SW_CLICK, then prevStateDurationMs will be the duration of the click. (You can use this value to differentiate between normal, long, very long clicks etc. as you wish.)
  - Because the output parameter is optional, you can just pass in NULL if you do not want to use it.

A3_ClickArrays requirements:

- Inside the loop function, declare 2 local arrays with 10 elements each that can remember their values between executions of the loop function. Initialize the elements of the first array in order to the integer values 10-19 inclusive. Initialize the second array elements to be zeros.
- Whenever SW1 is normally clicked (< 1 sec, single click), print the two arrays by calling the `printArray` function for each one. (You must write the printArray function to use an array pointer and pointer arithmetic based on lab 7.)

Hint: You should use readSwitchEventTimes and the pPrevStateDurationMs output parameter to determine the duration of every click and hence to decide whether it is a normal, long, or very-long click.

- Whenever SW1 is long-clicked (1-3 sec), fill the first array with a set of random numbers (each number between 0 and 99 inclusive) and then print the two arrays. You must call a new function that you define to fill the array with random values. Because this assignment focuses on array pointers, your function must use array pointers and pointer arithmetic to achieve its objectives (see lab 7). Use printArray to print the array values.
- Whenever SW1 is very-long clicked (>3 sec), set all the values in the first array to zero and then print both arrays. You must call a new function that you define to set all the array elements to zero. As before, the function must use pointers and pointer arithmetic internally. Use printArray to print the array values.
- Finally, whenever SW2 is normal-clicked, copy all the element values from the first array into the second array and then print both arrays. You must call a new function that you must define to copy values from one array to another. You must use pointers and pointer arithmetic in this function. Use printArray to print the array values.

A3_ClickArrays extra credit:

- Create a new variation of the programme that executes the copy and print functionality (last bullet point above) when a double-click on SW1 is detected instead of when a normal click on SW2 is detected.
    - To detect a double click you need to check if two short (normal) clicks have occurred within 1 sec.
    - Checking for double-clicks will complicate your detection of ordinary normal clicks also, since you cannot be sure that a normal click is not part of a double click until the maximum time between clicks has elapsed. You need to ensure that the first click of a double click doesn't result in the single-click action (i.e. just printing the two arrays) taking place.
    - If you detect a long click or very-long click you don't need to worry about the possibility of a double-click – it only applies to normal/short clicks.

Hint: I recommend that you just use your own static variable to remember the time the last click was detected so that you can measure the time elapsed since the previous click.

✏ Copy the sketch into your answer document.

2 Last in first out (LIFO) buffer – save sketch as “A3_LIFO”

Background: you will develop a program that implements a last-in-first-out buffer (using an array) and you will use this dynamically add (and later remove) values from the end of the array.

A3_LIFO requirements:

- Inside the loop function, declare an int array that will be the LIFO buffer. You will also need a variable to keep track of the number of elements of the LIFO that are used (occupied with valid values).
    - Initially, none of the elements will be “used”.
    - DO NOT use any special element values (e.g. -999) to work out whether or not a particular buffer/array element is unused. All int values are legal so there is no value that is “invalid”. Instead you must use the separate variable you define to keep track of the number of elements in use instead. For example, if the array is of length 10 and your “used” variable indicates that there are 3 elements in use, then you know that there are 7 (10 minus 3) spaces still available in the array. You also know that the next free element is at index 3 (since the 3 elements already occupied are at indexes 0, 1, and 2 respectively).
- Write functions for each of the following operations (use pointers, inout parameters, etc. as needed) and ensure that you use these functions as needed in the remainder of the program
    - Print the number of used slots and the element values in those used slots of the buffer
    - Get the number of used slots in the buffer.
    - Get the number of free slots in the buffer.
    - Add a number to the end of the buffer. To do this, copy the number to the next available slot (if there is one) in the buffer and update the variable that tracks the number of used slots.
    - Remove the most recently added number from the end of the buffer (if there is one) and return it. Get the value at the last used position of the buffer and then update the variable that tracks the number of used slots to indicate that this slot is not used anymore.
- When SW1 is clicked
    - Check if there are unused slots in the buffer
    - If there is space, generate a random number (any integer between 0 and +99 inclusive) and add it to the end of the buffer. Then print the new contents of the buffer.
    - Otherwise (no space in the buffer), print a suitable message
- Whenever SW2 is clicked
    - Check if there are any used slots in the buffer
    - If there are elements in use, remove the last element (the one that had been most recently added) and print its value. Then print the new contents of the buffer.
    - Otherwise (no elements in use), print a suitable message
- The programme output should look something like as follows:

```
Loop 30: [SW1 Click] added to buffer, 7, buffer is now { 7, }, 1 used
Loop 37: [SW1 Click] added to buffer, 49, buffer is now { 7, 49, }, 2 used
Loop 52: [SW1 Click] added to buffer, 73, buffer is now { 7, 49, 73, }, 3 used
Loop 78: [SW1 Click] added to buffer, 58, buffer is now { 7, 49, 73, 58 }, 4 used
Loop 91: [SW1 Click] Cannot add to buffer -- buffer full
Loop 114: [SW2 click] removed from buffer, 58, buffer is now { 7, 49, 53, }, 3 used
Loop 130: [SW2 click] removed from buffer, 73, buffer is now { 7, 49, }, 2 used
Loop 144: [SW2 click] removed from buffer, 49, buffer is now { 7, }, 1 used
Loop 156: [SW2 click] removed from buffer, 7, buffer is now {  }, 0 used
Loop 175: [SW2 click]  Cannot remove from buffer -- buffer empty
```

Hint: Develop the programme in stages. Consider starting with just detecting the click and long-click correctly and printing out which one was detected. Then keep track of the number of used elements increasing for each click (up to the size of buffer) and decreasing for each long-click (down to zero). Finally add support for generating random numbers and actually adding numbers to or removing numbers from the buffer array and printing the (valid in-use) contents of the buffer.

✏ Copy the sketch into your answer document.

3 Reflection

Q1: What did you learn in the associated lab and assignment?

Q2: What was the most important thing you learned in the lab/assignment?

Q3: What was the hardest aspect of this lab/assignment and why?