WORKSHEET 5b

For this worksheet you are expected to work individually to complete all of the following exercises. The main outcome of this tutorial session is to ensure that you can dereference a pointer to access the value stored a given address, that you can find the address of a given variable, that you can understand pointers to pointers and that you can use pass-by-reference parameters.

This week's tutorial task is shown in red below.

EXERCISE 1: WHAT IS YOUR MEMORY LIKE?

Difficult Level: ★ **Duration: 15mins**

Write a program that asks the user to type a single char, an integer value, a float value and a string. The program should then print out the address of each variable in hexadecimal format.

EXERCISE 2: ALL POINTERS ARE COVERED

Difficult Level: ★★ **Duration: 20mins**

Write a program to print out the address of an array of the three strings and the address of each string in the array. You should use the following declaration in your code to represent the array of strings:

```
char* argv[] = {"Hello", "World", "!!!"};
```

Hint: you will need to use the special format specifier %p with `printf()` to print out the requested addresses.

EXERCISE 3: OUT ON THE RANGE

Difficult Level: ★★★ **Duration: 25mins**

Write a program that prints the minimum and maximum values in a randomly generated array of 20 values using a function called `maxmin`. The function has the following prototype:

```
void maxmin(int x, int *pmax, int *pmin);
```

This function should compare `x` with the given maximum and minimum values pointed to by `pmax` and `pmin` respectively. Your function definition should then update the maximum and minimum values as appropriate.

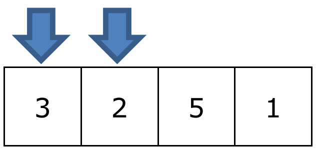
Hint: the initial values of the variables pointed to by `*pmax` and `*pmin` should be chosen carefully, i.e. do not use the value 0 for both.

EXERCISE 4: BLOWING A BUBBLE

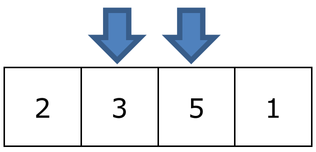
Difficult Level: ★★★★ **Duration: 1hr**

One of the most basic algorithms that can be used to sort numerical values is called a Bubble Sort. This algorithm works by swapping adjacent array values if the two values are deemed to be in the wrong order - assuming the array can be ordered in ascending or descending order. The algorithm must scan the whole array (n – 1) times where n is the size of the array making changes where the order of two adjacent values is wrong.

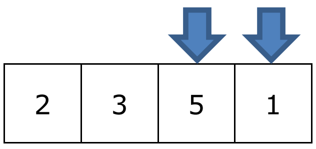
Below is a graphical example of a single scan of an array of 4 items that is being sorted in ascending order. The first two values are in the wrong order because 3 is larger than 2. These items are swapped.



The next two values are in the correct order (i.e. 3 is smaller than 5) and so these items are not swapped.



The final two values are in the wrong order (i.e. 5 is larger than 1) and so they are swapped.



After a single scan the array looks like this:

2	3	1	5

Clearly, this array is still not in ascending order. In fact, to be sure that it is in the correct order it must be scanned in this manner 3 times (i.e. n – 1 where n = 4). The resulting array after 3 scans is shown below:

1	2	3	5

Write a program that sorts an array of 20 random integer values using the Bubble Sort algorithm described above.

Hint: you should make use of the `swap()` function shown in the lecture slides to swap values in the array.