Homework 2

Due Monday February 1 by 1:30PM. Remember you can turn it in late with a 10% per day late penalty.

In this assignment you will write a few different C functions and call them from your **`main`** function. Create a directory named **`hw2`** in your personal course repo. Note the spelling. Make sure the directory is named **`hw2`**and not **`HW1`** or **`Hw2`**, etc. Make sure all of the files outlined below are in your **`hw2`** directory and updated in your personal course repo by the due date and time.

1. [10 points] Write a C function **`first_nonrepeat`** that takes a string and returns the index of the first non-repeated chracter in the string. Put this function in a file **`first_nonrepeat.c`**. For instance, the index of the first nonrepeated character in "total" is 1 (because it is the index of "o") and the index of the first non-repeated character in "teeter" is is 5 (because the index of "r" is 5). If the string does not have any non-repeated characters (such as "meme" and "unprosperousness") the function should return -1.

   Hint: I can think of two obvious ways to write this function. One is a brute force way that looks at the characters in the input string multiple times. The other way only loops through the string at most twice and uses the fact that characters in C are really 8 bit integers and you could use a character as an index into an array that kept track of character frequencies. (How big does the array need to be?). Partial credit for the brute force technique and full credit for using the characters as an index into an arry.

   ```
   printf("%d\n", first_nonrepeat("total"));            // prints 1
   printf("%d\n", first_nonrepeat("teeter"));           // prints 5
   printf("%d\n", first_nonrepeat("unprosperousness")); // prints -1
   printf("%d\n", first_nonrepeat("palatial"));         // prints 0
   ```

2. [10 points] Write a C function **`longest_distinct`** that takes an array of integers and returns the length of the longest subarray where all of the elements are distinct. Put this in a file named **`longest_distinct.c`.** For example, if given the C array 5, 1, 3, 5, 2, 3, 4, 1 the function would return 5 because the subarray 5, 2, 3, 4, 1 is the longest subarray where all of the elements are different.

   Try this on the arrays

   ```
   1, 1, 1, 1, 1, 1        // answer is 1
   1, 1, 2, 3, 4, 5, 3     // The answer is 5
   129, 990000000, 12345, 99   // answer is 4
   ```

   In C, when passing an array as an argument you usually need to also pass the integer length of the array. That is because in C, arrays do not know how big they are (they do not have a **`size`** attribute or **`length`** method). So the type declaration for the **`longest_distinct`** function should be …

   ```
   int longest_distinct(int vec[], int n) { … }
   ```

Homework 2

3. [10 points] Write a function **`mirror`** (in the file **`mirror.c`** of course) that takes an unsigned char **`n`** (an 8 bit unsigned an integer) and returns a 1 (true) if the bits of **`n`** read the same forward and backwards. For example, 195 in base ten is 11000011 in binary. So **`mirror`** would return 1. The call **`mirror(1)`** would return 0 because the reverse of 1 (1 is really 0b00000001) is an 8 bit integer is 10000000. Use bit operations to write this function. Don't tryto convert things to strings and read the string for4ward and backwards.

```
printf("%d\n", mirror(195));  // prints 1
printf("%d\n", mirror(1));    // prints 0
printf("%d\n", mirror(255));  // prints 1
printf("%d\n", mirror(165));  // prints 1
```

   Hint: Don't overthink this problem and make it more complicated than it needs to be. Think about our bit operatations `&, |, <<, >>`.

4. Write a function named **`main`** in a file named **`main.c`** and calls each of your functions above.