When working with collections of numbers, it is common to use an array to hold these numbers. An array is simply that a set of numbers that have the same name and are accessed by an index. In code it is declared as

```
#define COMBO_DIGITS 4
int Combination[COMBO_DIGITS];
```

Now it is not necessary to use a defined constant to set the size of an array, you could just have easily written this as

```
int Combination[4];
```

However if you want to change the number of digits used, it is can be hard to hunt down all the places that need to be changed. If a define value is used throughout the code, changing it in one place changes all the code.

Initialization of an array can be done at its declaration. Again it might look like

```
int Combination[COMBO_DIGITS] = {1, 2, 3, 4};
```

Which will start the array off as 1, 2, 3 and 4. If you have a large array and want to start it off as being all zeros, it can be impractical to write a long string of 0's. The following short hand works to do this.

```
int Combination[COMBO_DIGITS] = {0};
```

Now when handling an array in the program, the individual elements in the array can be accessed use the [] operator such as

```
Combination[0] = 1;
```

Now it should be noted that for the array of four elements the indices are 0, 1, 2 and 3. This fact will cause more confusion than one would expect. Consider if we were going to force an array, call it `Code[COMBO_DIGITS]`, to all zeros. The {} operator only works at initialization, so

```
Code = {0, 0, 0, 0}; // This won't work.
```

will not work. Rather the code will need to loop through the various elements setting each

```
int i;
for( i = 0; i < COMB_DIGITS; i++ )
     Code[i] = 0;
```

Note this is one time when a loop is appropriate in an embedded program. Now to compare two arrays, again you will need to go through them element by element and test each pair of elements. As an example, consider the code below that tests if two arrays are the same.

```
int TestArraysEqual( int Combo[], int Code[], int LengthOfCodes )
{
   for(int i = 0;             // Start at beginning
         i < LengthOfCodes    // Continue if still in array bounds
         i++)                 // Move to next index.
   {
     if( Combo[i] != Code[i] ) // If the i'th elements don't match
          return 0;            // Return 0 indicating that arrays are not equal.
                               // Which also leaves the function, but we now know
                               // that the arrays are not equal.
   }
   // if we go all the way through the array and none of the elements
   // are not equal
   return 1; // Return 1 to indicate the two arrays are equal.
}

...

// In main program a set of arrays are declared.
#define LENGTH_OF_DATA 10
int Combination[LENGTH_OF_DATA], Code[LENGTH_OF_DATA];

...

   // Then when we want to compare the two arrays.
   if( TestArraysEqual( Combination, Code,  LENGTH_OF_DATA ) )
   {
      // Code for case of the two arrays being equal.

   }
```

If a 1 is returned, at the end of the "for" loop in the function `TestArraysEqual`, the result is true. However it even one of the elements in the two arrays are not equal a 0 is returned and that evaulates as a false.

Two things need to be explained about this code. As we go through the arrays, we simply return a false if any elements are not equal, since we can stop looping once we find a pair of elements that don't match. Second, it is common to need an integer variable for a short loop such as this one, and rather than having to declare this integer at the top of the program or function, a short term variable can be declared inside the loop. Here "i" is created at the start of the "for" loop and used primarily in this loop.

It should be noted that the compiler will accept a statement like

```
ArraysEqual = ( Combination == Code );
```

But this test will return a false regardless of the values in the arrays. The explanation of this will be described in class, but note the details of this can be confusing and is beyond what we want to discuss here.