CMSC110 Introduction to Computing

Lab#9: Array Calisthenics
Week of October 31, 2016

1. The following function finds the smallest element of an array of `floats`:

```
float findMin(float[] numbers)
{
  float min = numbers[0];

  // Now, compare min with every element in numbers,
  // replacing min with any that are found to be smaller
  for (int i=0; i < numbers.length; i++) {
     if (numbers[i] < min) {
        min = numbers[i];
     }
  }

  return min;
} // findMin
```

a. What can go wrong with this function? Is there ever a time it will crash? (*Hint*: Yes!)

b. Put this function into a Processing program.

c. In Processing, write some code (in `setup()`) that demonstrates that this function works. Your code will have to create an array of `floats`, put some numbers in it, and then print out the result of calling `findMin`.

2. Write a new function `float findMax(float[] numbers)` that finds the maximum element in an array. Use `findMin` as a template. Make sure to test your function (by calling it from `setup()` and printing out the result).

3. Write a new function `int findMaxIndex(float[] numbers)` that finds the *index* of the maximum number in an array. For example, if we have an array `xs` that contains `{2.3, 8.1, 4, 6.28}`, then `findMaxIndex(xs)` will return `1` because the maximum element, 8.1, is at index 1 in the array. (Remember that indexing starts at 0.) Make sure to test your function.

4. Write the function `float sum(float[] data)` that returns the sum of all the elements in an array. For example, if we have array `xs` that contains `{2.3, 8.1, 4, 6.28}`, then `sum(xs)` will return `20.68`. Make sure to test your function.

7. How to swap the contents of two integers? While not quite an array operation, it is needed in order to do some later array operations.

```
float a = 11, b = 5;
// Swap the contents of a and b
float temp = a;  // save contents of a in temp
a = b;           // copy contents of b in a
b = temp;  // restore contents of a from temp
// Now contents of a and b are swapped
```

As you can see, you need a third variable, `temp`, to hold the contents of one of the variables in order to swap.

Copy this code into Processing and test it to make sure it works as expected.

8. How to reverse the contents of an array `numbers`? After the code below is carried out, it will contain the elements in reverse order, i.e.

```
// Assume, numbers = [4, 7, 2, 1, 5, 2, 1, 9]
for (int i=0; i < numbers.length/2; i++) {
   float temp = numbers[i];
   numbers[i] = numbers[numbers.length-1-i];
   numbers[numbers.length-1-i] = temp;
}
// Now numbers = [9, 1, 2, 5, 1, 2, 7, 4]
```

9. Next, use the code above to write a function `void reverse(float[] data)` that reverses a given array. Test your function.

10. Consider the code below:

```
float[] numbers = new float[15];
int n;
…
// numbers now contains [5, 2, 6, 8, 6, 5, 3, 2, 0, 0, 0, 0, 0, 0, 0]
// And, n = 8 indicating the number of elements set in numbers
```

Write command(s) to add the number 11 after these numbers (that is, at index `n`). Remember to update the value of `n`.

11. Given the array, `numbers` in (10) above, and `n=8`, and the array `moreNumbers` as shown below:

```
float[] moreNumbers = new float[15];
int m;
…
// moreNumbers contains [11, 12, 13, 14, 0,0,0,0,0,0,0,0,0,0,0], m = 4
```

Write command(s) to append the contents of `moreNumbers` to the end of `numbers`. That is, after the command(s) are carried out, `numbers` will contain `[5, 2, 6, 8, 6, 5, 3, 2, 11, 12, 13, 14, 0, 0, 0]`, and `n = 12`. Both `moreNumbers` and `m` will remain unchanged.

12. Write a function called `append()` defined as follows. Make sure to test it!

```
void append(float[] a, int na, float[] b, int nb) {
// Appends the contents b[0..nb-1] to a[]
// If there isn't room for appending, do nothing
```

13. Write a function `accumulate()` defined as follows. Make sure to test it!

```
float[] accumulate(float[] a)
// Creates and returns an array as follows:
// Suppose a = [1, 2, 3, 4, 5]
// accumulate(a) will return the array [1, 3, 6, 10, 15],
// where each element is the sum of all previous elements in
// the original array
```

14. Write a function `sieve()` defined as follows. Make sure to test it!

```
void sieve(boolean[] a, int n)
// Sets the value of all indexes in a that are multiples of n to false (but leaving
// element n alone)
// E.g. Suppose a=[false, true, true, true, true, true, true, true, true, true, true]
// After the call, sieve(a, 2):
// a = [false, true, true, true, false, true, false, true, false, true, false]
// Further, after the call, sieve(a, 3):
// a = [false, true, true, true, false, true, false, true, false, false, false]
```

15. [Ultimate Challenge] Consider the commands below:

```
boolean[] numbers = new boolean[100];
for (int i=0; i < numbers.length; i++)
   numbers[i] = true;
numbers[0] = false;

for (int n = 2; n < numbers.length/2; n++) {
    if (numbers[n]) {
      sieve(numbers, n);
    }
}
```

What can you say about the state of the `numbers` array?

[Hint] Print out the indices of all elements in `numbers` that are **`true`**.