CMPS 1600 Introduction to Computer Science II – Spring 14

3/15/14

7. Homework

Programming portion due **Thursday 3/20/14** at 11:55pm on Blackboard.
Written portion due the next day at the beginning of class.

Please submit a `.c` file for each programming question to Blackboard.
In order to receive any credit for the programming portions, you are required to thoroughly comment and test your code.

1. **Counting Sort (10 points)**
Consider an array of non-negative integers (i.e., numbers greater or equal to zero) that may contain the same integer values multiple times. An example input is: `3, 2, 6, 4, 3, 2, 3, 4, 6, 3, 4, 3, 5, 2, 6`. Instead of using a comparison-based sorting algorithm like merge sort or selection sort, you will implement *counting sort* which uses counters to keep track of the multiplicity of each number in the array. For the example input above, the basic idea is to count how many `2`s, `3`s, `4`s, `5`s, `6`s are in the array, and then simply write that many numbers into the output. In our example there are three `2`s, five `3`s, three `4`s, one `5`, and three `6`s. Therefore the output is `2,2,2, 3,3,3,3,3, 4,4,4, 5, 6,6,6`.

   (a) (8 points) Write C-code to implement counting sort. Your code should minimally contain:

   - The `printArray` function that we covered in `arrays.c` in class.
   - A function `maximum` that has two arguments: (1) an integer array or pointer, and (2) an integer storing the length of the array. `maximum` should return the maximum number in the array.
   - A function `countingSort` that has two arguments: (1) an integer array or pointer, and (2) an integer storing the length of the array. `countingSort` should perform the counting sort algorithm in the following steps: (1) Compute the maximum of the numbers in the array using your `maximum` function. (2) Create a new counter array that is one larger than the maximum number you just computed. [Remember, all numbers in the input array are greater or equal to zero.] Then use this counter array to count all the numbers in your input array. For the example above, your counter array should be:

	0	1	2	3	4	5	6
counter array:	0	0	3	5	3	1	3

     (3) Finally, override the input array with the sorted numbers by looping through your counter array and for each non-zero counter adding that many numbers to the array. Thus, the sorted output will be stored in the input array.
   - A `main` function in which you declare some test arrays (among them the above example array) and then call the `countingSort` function. Use `printArray` to print the array before and after; you can also use `printArray` to print the counter array along the way to debug your code.

Flip over to back page ⟹

(b) (1 point) The runtime of counting sort should be $O(n + k)$, where n is the size of the input array and k is the maximum of all the numbers in the array. Argue why your code has this runtime. *(Hint: You may encounter nested loops that overall still have linear runtime. Why?)*

(c) (1 point) `countingSort` overrides the input array with the sorted numbers. Justify why this kind of modification that has been performed inside the `countingSort` function is visible in the `main` function.

2. **Arrays and Pointers (10 points)**
Consider the following code which is given in `hw7-arrays.c`:

```
int A[10]={3, 6, 9, 1, 2, 8, 15, 4, 12, 5};
int* B = malloc(10*sizeof(int));
int* C = malloc(10*sizeof(int));
int i=0;
for(i=0; i<10; i++){
  B[i]=i;
  C[i]=2*i;
}
printf("A: "); printArray(A,10);
printf("B: "); printArray(B,10);
printf("C: "); printArray(C,10);
```

(a) (2 points) Add print statements to `hw7-arrays.c` to print the addresses and contents of all variables in the above code. Use `%u` in the `printf` function to print the addresses as unsigned integers. Run your code on your computer.

(b) (6 points) On paper, draw the memory with all the addresses and contents of all variables. Distinguish between the stack (used for variables) and the heap (used for dynamically allocated memory). Write the addresses next to the stack and heap depictions. This should look similar to the pictures we drew in class. Make sure to show all contents of all arrays.

(c) (1 point) What value is the difference between two consecutive addresses in the arrays? Why is it this value?

(d) (1 point) Assume that after the code above we wanted to call a function `arraySwap( ?   , ?   )`, which should swap `B` and `C`. That is, after swapping the arrays, `printArray(B,10)` should print the values that used to be stored in `C`, and vice versa.

What should the question marks in `arraySwap( ?   , ?   )` be replaced with, in order to achieve the desired result? Justify your answer shortly.

(e) (2 extra credit points) Write the function `arraySwap`.