Introduction to Computing Test 2

Question 1. (15 points) Complete the tracing of the following code to show the expected output and the run-time stack as the program executes. Recall that a function's call-frame contains the return address, formal parameter(s), and local variable(s).

```cpp
#include <iostream>
using namespace std;

int doSomething(int a, int & b);

int main() {
    int x = 5, y = 7, z;
    z = doSomething(x, y);
    cout << "x = " << x << " y = " << y
         << " z = " << z << endl;
} // end main

int doSomething(int a, int & b) {
    int x = 3;
    x = 2 * a;
    b = x + b;
    return x;
} // end doSomething
```

**Expected Output**

```
x = 5  y = 17  z = 10
```

Question 2. (10 points) Use the gradebook example discussed in class to describe what is meant by the term “parallel arrays”.

The same index value across multiple arrays is used to relate information about the same entity. For example, row 0 in `studentNames` and row 0 in `scores` both refer to Doe, Jane's information.
Question 3. (10 points) All simple sorts consist of two nested loops. The role of each loop is:
- outer loop - keeps track of the dividing line between the sorted part and unsorted part of the array
- inner loop - extends the size of the sorted part by one element

For each simple sort listed below, describe in English how the inner loop extends the size of the sorted part by one element. Assume that all sorts are sorting from smallest to largest elements (ascending order).

a) bubble sort:

The inner-loop scans the unsorted part from 1-to-N, comparing adjacent elements. If they are out-of-order, then it swaps them. This causes the largest element in the unsorted part to "bubble" to the end of the unsorted part.

b) selection sort:

The inner-loop scans the unsorted part to find the minimum element in it. After the inner-loop, the min. element is swapped with the first unsorted element.

Question 4. (40 points) Complete the following program by (1) writing the calls in the main to both functions, and (2) writing both of the functions: FillArray and FindFirstAndCount. These functions’ prototypes are shown below, and they should behave as follows:

FillArray - reads values interactively into the “numbers” array until the specified sentinelValue is read. After the function, “elementCount” should reflect the number of values read into “numbers.”

FindAll - scans the array “numbers” looking for all occurrences of “target”. The index locations of all occurrence of “target” is returned in “foundIndexes”, and “foundCount” returns a count of the number of times “target” occurs in the array “numbers”.

```cpp
#include <iostream>
using namespace std;

const int SIZE = 100;  // maximum number of elements in array

// function prototypes
void FillArray(double numbers[], int &elementCount, double sentinelValue);
void FindAll(double numbers[], int elementCount, double target, int foundIndexes[],
            int &foundCount);

int main () {
    int countOfNumbers, targetCount, targetLocations[SIZE];
    double numbers[SIZE], sentinel = -999, target = 100;

    // COMPLETE THE CALLS TO THE FUNCTIONS
    FillArray(numbers, countOfNumbers, sentinel);
    FindAll(numbers, countOfNumbers, target, targetLocations, targetCount);

    if (targetCount == 0) {
        cout << "The target " << target << " was not found." << endl;
    } else {
        cout << "The target " << target << " was found " << targetCount
            << " times with the first occurrence at " << targetLocations[0] << endl;
    } // end if
} // end main
```
// WRITE THE CODE FOR THE FUNCTIONS FillArray and FindAll BELOW
void FillArray(double numbers[], int & elementCount, double sentinelValue) {
    do {
        cout << "Enter a value (or " << sentinelValue
        << " to quit) : "
        << endl;
        cin >> value;
        if (value == sentinelValue) {
            return;
        }
        numbers[elementCount] = value;
        ++elementCount;
    } while (true);
} // end FillArray

void FindAll(double numbers[], int elementCount, double target, int foundIndexes[],
             int & foundCount) {
    foundCount = 0;
    for (int testIndex = 0; testIndex < elementCount; testIndex++) {
        if (target == numbers[testIndex]) {
            foundIndexes[foundCount] = testIndex;
            ++foundCount;
        }
    } // end for
} // end FindAll
Question 5. (10 points) Suppose you have a sorted array containing 1,000 elements.
a) In the worst case, how many comparisons between the target and an array element would be performed in an **unsuccessful** binary search? 

\[ 10 \]

b) In the worst case, how many comparisons between the target and an array element would be performed in an **unsuccessful linear search** (assume the linear search algorithm does not expect the array is sorted)? 

\[ 1000 \]

Question 6. (15 points) Below is the textbook's code for a **binary search** on a sorted array.

```c
int binarySearch(int array[], int size, int value) {
    int first = 0,  // First array element
        last = size - 1,  // Last array element
        middle,  // Mid point of search
        position = -1;  // Position of search value
    bool found = false;  // Flag

    while (!found && first <= last) {
        middle = (first + last) / 2;  // Calculate mid point
        if (array[middle] == value) {  // If value is found at mid
            found = true;
            position = middle;
        } else if (array[middle] > value) {  // If value is in lower half
            last = middle - 1;
        } else {  // If value is in upper half
            first = middle + 1;
        }
    } // end if
    } // end while
return position;
} // end binarySearch
```

Trace the `binarySearch` code using the following actual parameters by showing the changes to first, last, middle, position, and found.

<table>
<thead>
<tr>
<th>first</th>
<th>last</th>
<th>middle</th>
<th>position</th>
<th>found</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>3</td>
<td></td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
<td></td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>array: 2 3 4 5 7 8 9</th>
<th>(MAX-1)</th>
<th>size: 7</th>
<th>value: 7</th>
</tr>
</thead>
</table>

15

4

25