Spring 2014 Name: Mark Fienup

C/C++ Programming 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).

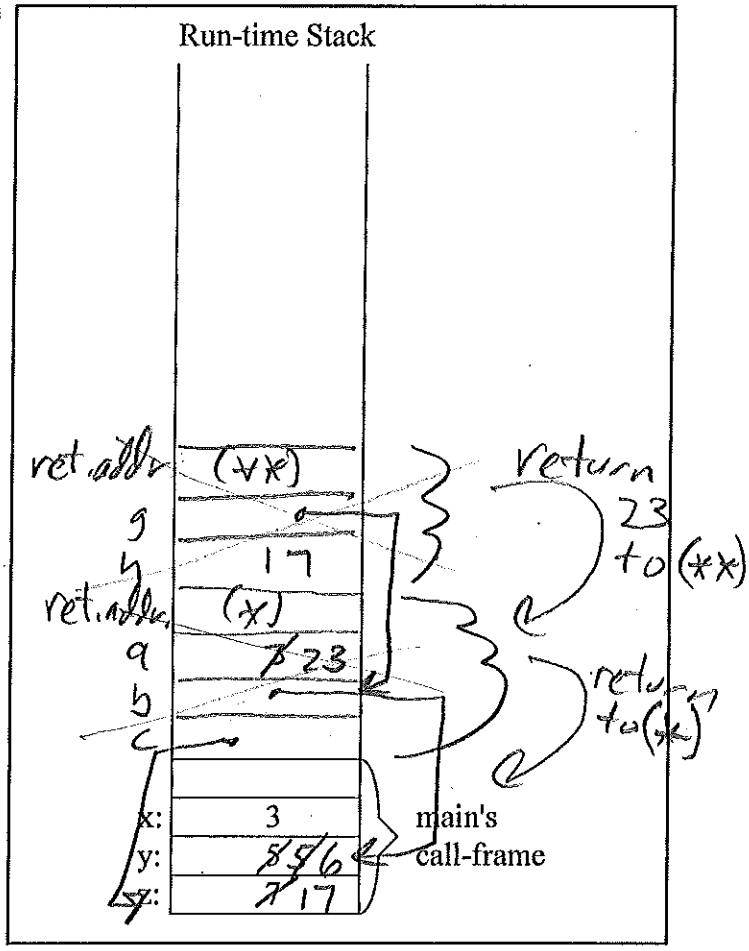
```
#include <iostream>
using namespace std;

void doSomething(int a, int & b, int & c);
int doMore(int & g, int h);

int main() {
   int x=3, y=5, z=7;

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

void doSomething(int a, int & b, int & c) {

   b = a + 2;
   c = c + 10;
   a = doMore(b, c);
   cout << "a: "<< a << "b: "<< b << endl;
} // end doSomething

int doMore(int & g, int h) {
   g = 6;
   return g + h;
} // end doMore
```

Continue trace from here

(*)

(**)

(***)



Expected Output

9% 23b% 6 \n
X= 3 Y = 6 Z= 17 \n

Question 2. (10 points) Explain why you should design a program by splitting it up into functions before writing the program.

If you design first, you will make fewer mistakes. Splitting the larger problems into small functions makes the solution easier to figure out.

Question 3. (10 points) Suppose you have a sorted array containing 1,000 elements which you would like to search for a specified target value.

a) In the worst case, how many comparisons between the target and an array element would be performed in an **successful binary search**? 9 or 10

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

1,000

Question 4. (15 points) Below is the textbook's code for a *binary search* on a sorted array.

```
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 {
         first = middle + 1;           // If value is in upper half
      } // 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.

	0	1	2	3	4	5	6	7	8	9	(MAX-1)
array:	2	3	4	5	7	8	9	11	14	17	

size: 10 value: 9

first	last	middle	position	found
~~0~~	9	~~4~~	-1	false
~~5~~		~~7~~		
		6		
		5		
6				
6		6		true

Question 5. (25 points) Suppose we have:
quizScores - an array of doubles containing quiz scores,
quizMaxPossibles - a parallel array containing the maximum possible for each quiz, and
quizCount - an integer specifying the number of quizzes in the array.
For example:

	0	1	2	3	4	5	6	7	8	9	(MAX-1)
quizScores:	7.0	5.0	3.0	6.0	4.0	2.0	0.0	3.0	5.0	7.0	

quizCount: 10

	0	1	2	3	4	5	6	7	8	9	
quizMaxPossibles:	7.0	6.0	5.0	6.0	5.0	5.0	5.0	5.0	6.0	9.0	

Write C++ code to:
- calculate the total of all quizzes
- calculate the maximum total over all quizzes
- calculate the percentage correct over all the quiz scores
  (i.e., 100 * (total of all quizzes) / (max. total over all quizzes))

```
int index;
double total, totalPossible, percentCorrect;

total = 0.0;
totalPossible = 0.0;
for (index=0; index < quizCount; index++) {
    total += quizScores[index];
    totalPossible += quizMaxPossibles[index];
} //end for
percentCorrect = 100 * total / totalPossible;
```

Question 6. (10 points) Consider the following two-dimensional array definition:

```
double salesTable[8][10];
```

2 a) How many rows does the array have? 8

2 b) How many columns does the array have? 10

3 c) How many elements can be stored in the array? 80

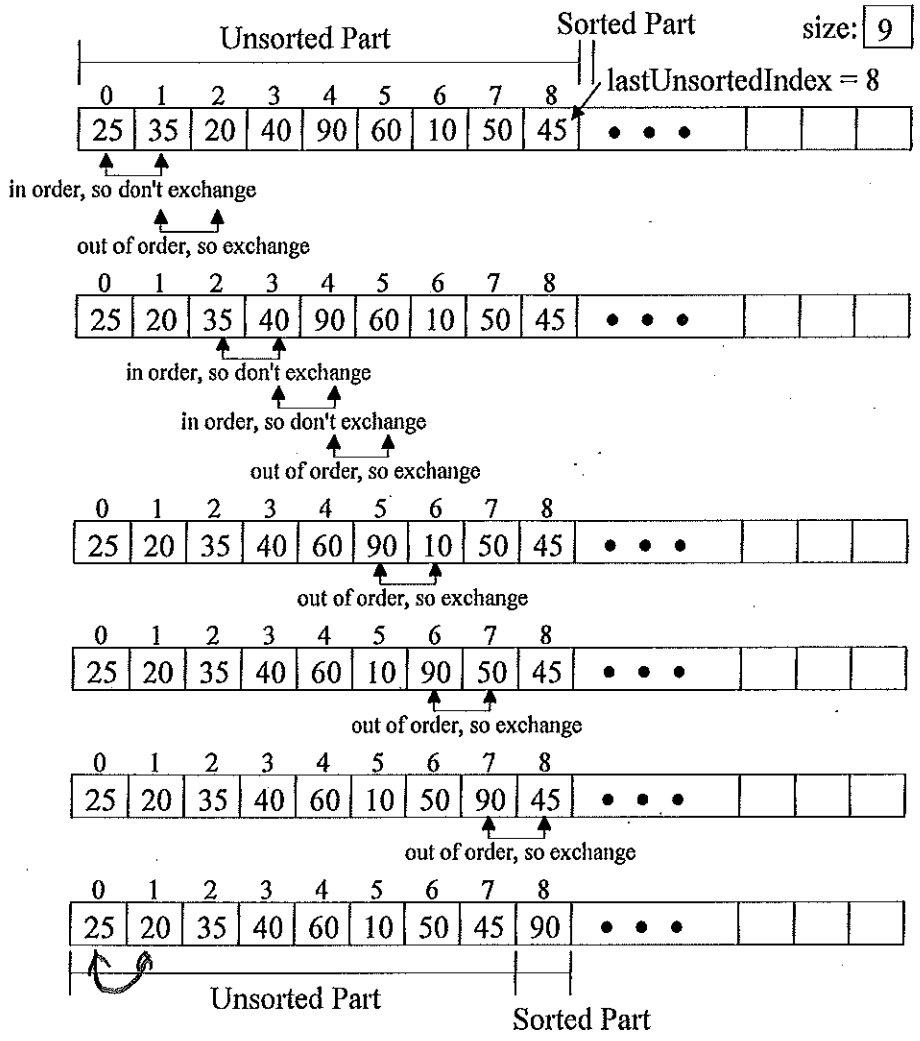
3 d) Write an assignment statement to store 10.3 in the last column of the first row in the array.

```
salesTable[0][9] = 10.3;
```

35

Question 7. (15 points) Recall *Bubble sort* is an example of a simple sort. It's outer-loop keeps track of the dividing line between the unsorted and sorted part of the array. Bubble sort's inner loop scans the unsorted part of the array comparing adjacent items. If it finds adjacent items out of order, then it exchanges them. This causes the largest item to "bubble" up to the "top" of the unsorted part of the array.

At the start of the first iteration of the outer loop, the initial array is completely unsorted:
The inner loop scans the unsorted part by comparing adjacent items and exchanging them if out of order.



After the inner loop's first execution, the sorted part has grow by one is size.

As done above, show the comparisons and exchanges for the inner loop's second execution.