CSCI 103L Practice MT1 Solutions
85 minutes

Q1: Short Answer. Fill in or circle the best response
   a) We must declare the __**type**__ of a variable so C/C++ knows how to interpret the
      meaning of the bits of that variable.
   
   b) “Under the hood” (I.e. in computer memory and the processor) all information in a
      digital computer is stored as ___**binary**_____ numbers.
   
   c) ____ True / False: A pointer to a variable of type char is the same size as a pointer to
      a variable of type int.
   
   d) An expression that contains only the name of an array without any brackets, yields:
      _____its starting address (or address of 0\text{th} item)__________.
   
   e) The word ___**scope**__ (one word) refers to the notion that the name of a variable is only
      valid in certain sections of code depending on where it was declared.
   
   f) ______ True / False: Given two C-string declarations: char str1[] = "<some text>" and
      char str2[] = "<other text>", it is possible the comparison str1 == str2 will yield true.
   
   g) The local variables for a function “live” in this area of memory: _**stack**________.
   
   h) Function prototypes make our code more flexible because the order in which we
      __________ (define / execute) functions does not matter. (circle one)
   
   i) Of the 4 properties of a variable: **name, type, location, value**
      which one is not (able to be) chosen by the programmer? _**location**_____.
   
   j) ______ True / False: A char* and an int* are type compatible (i.e. variables of one type can be
      assigned to the other).
   
   k) ______ True / False: Given the 2D array declaration char img[20][20] then img[1][0] and
      img[2][0] are at consecutive addresses in memory.
   
   l) ______ True / False: The keyword continue will start the next iteration of a loop immediately.
m) Given a call to a function: int x = f1(4.25, 'a', 0, "hello"); infer the correct prototype for the function and write it below:

_____int f1(double, char, int, char*); // last one is technically const char*

n) ______ True / False: An if statement requires a corresponding else statement.

Q2: Consider the following code. Trace the execution and calculate the value that would be printed on each line. Any update to a variable is carried through the rest of the program.

Note: Show doubles to whatever accuracy is necessary, dropping trailing 0s. Also, note that printing a boolean value will simply print 1 (for true) or 0 (for false).

// appropriate headers and using statement
int main()
{
    int a = 19, b = 10, c = 10;
    double y = 3.75, z = 1.5;
    (answer here ↓)
    cout << -b/a << endl;
    cout << a++/(b - 4) << endl;
    cout << (int)z + y << endl;
    cout << a % 3 * c / 8 << endl;
    cout << ((a/100 || b/5) && (b%2)) << endl;
    cout << c * z << endl;
}

Q3: (5 pts) Consider the following code fragment, fill in the output in the corresponding blank.

Remember: boolalpha causes ‘cout’ to print ‘true’ or ‘false’.

char c = 'h';
char s[] = "hello";
const char *h = "hello";

cout << boolalpha << (c == s[0]) << endl; ___true___
cout << boolalpha << (h == s) << endl; ___false_________
cout << boolalpha << (*h == "s") << endl; ___true_________
cout << boolalpha << (s[0] == *h) << endl; ___true_________
cout << boolalpha << (*(&c) == h[0]) << endl; ___true_________
Q4 Functions, Pass-by-value & reference. Examine the code below using functions and show what the program would output for the given cout statements.

```cpp
// assume appropriate headers and using statements
void f1(int* a, int b, int *c);
int f2(int* b, int c);

void f1(int* a, int b, int *c)
{
    f2(&a[1], a[0]);
    a[5] = f2(&b, *c);
    a[3] = 18;
    a += 2;
    a[2] = -3;
    cout << "F1: b = " << b << " *c = " << *c << endl;
}

int f2(int *b, int c)
{
    *b *= 3;
    c++;
    return c;
}

int main()
{
    int dat[6] = {1, 1, 10, 10, 40, 40};
    int b = 5, c = 8;
    f1(dat, b, &c);
    cout << "Dat: ";
    for(int i=0; i < 6; i++){
        cout << dat[i] << " ";
    }
    cout << endl << "b= " << b << " c= " << c << endl;
    return 0;
}
```

Program Output:

```
F1: b = 15 *c = 8
Dat: 1 3 10 18 -3 9
b=5 c=8
```
Q5: (10 pts) The program below should read in a character array (not an int) containing at most a 10-digit number and then fill in a new character array that inserts commas in the correct places. So if the user enters 1234567, the function should fill in the output character array with 1,234,567. The smallest number input will be 0 (for which you should just output 0) and the largest number that will be input is 9999999999 (for which you should output 9,999,999,999). Read the comments in main() for more assumptions you can make.

a.) Fill in the sizes of the instr and outstr character arrays in main() using only the size necessary and no more.

b.) Then complete the blanks in the addCommas() function to make the code work correctly. You may not change any of the provided code (you can only fill in the blanks).

// assume appropriate headers including <cstring> and using statements
// assume an appropriate prototype for addCommas
int main()
{

    char instr[11], outstr[14]; // Fill in the array sizes
    cout << "Enter a number between 0 and 9999999999: " << endl;

    // Assume the user will not input a number with leading zeros
    // (i.e. the user will only enter 0 not 0000000000 and 123456 not 00123456).
    // Further assume the user never types non-digits like letters or punctuation.

    cin >> instr;
    addCommas(instr, outstr);
    cout << "With commas: " << outstr << endl;
    return 0;
}

void addCommas(__char*_____ is , __char* ___ os )
{
    int len = __strlen(is)__;
    int k = 0;
    for(int i = 0; i < len; i++){

        os[k] = is[i];
        __k++________

        if( ( (__len-i____________) % 3 == ___1____ ) &&
            ( (__len-i _________) > 3 ) ) {
            // add a statement or two here in the if statement
            os[k] = ";
            k++;
        }
    }

    _os[k] = 0; // or \0"_________ // use only if you need additional code here
}
Q6

Consider the program shown below. Given the assumed starting locations of the arrays in memory, show the contents of each pointer on the appropriate lines. At the end of the program show the final contents of the arrays.

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

int main()
{
    // Assume x starts at address 200
    double x[] = {2.2, 4.4, 6.6, 8.8, 11.0};
    // Assume str starts at address 250
    char str[] = "103 170";

    // Show the pointer contents
    // (i.e. the address stored in p or r)

double* p = x+4;    // 6.1) p = __232___
char* r = &str[2];  // 6.2) r = __252___

*p += 1;            // 6.3) p = __232___
p -= 2;              // 6.4) p = __216___
*(p-1) -= 2;         // 6.5) p = __216___

*(r+2) = '2';        // 6.6) r = __252___
r = str+1;           // 6.7) r = __251___
r += 3;              // 6.8) r = __254___

// 6.9) Show the final contents of the x array. 2.2 2.4 6.6 8.8 12.0
// 6.10) Show the final contents of the str array. "103 270"
return 0;
```
Q7 Examine the eight 256 row x 256 column image options and the **two** code snippets below. The array indexing and value of colors like black, white, and shades of gray are the **SAME** as in your lab 5. For the two code snippets, identify the image (by number) that will be produced by that code. We had our word processor draw a black border around each image to make it easier to identify the boundaries.

<table>
<thead>
<tr>
<th>Image 1</th>
<th>Image 2</th>
<th>Image 3</th>
<th>Image 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image 1" /></td>
<td><img src="image2.png" alt="Image 2" /></td>
<td><img src="image3.png" alt="Image 3" /></td>
<td><img src="image4.png" alt="Image 4" /></td>
</tr>
<tr>
<td>Image 5</td>
<td>Image 6</td>
<td>Image 7</td>
<td>Image 8</td>
</tr>
<tr>
<td><img src="image5.png" alt="Image 5" /></td>
<td><img src="image6.png" alt="Image 6" /></td>
<td><img src="image7.png" alt="Image 7" /></td>
<td><img src="image8.png" alt="Image 8" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Corresponding Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.) for(int i=0; i &lt; SIZE; i++){ for(int j=0; j &lt; SIZE; j++){ img[i][j] = ((i%32)/16)*255; } }</td>
<td>5</td>
</tr>
</tbody>
</table>

**Room for scratch work:**

This code runs in \(O(n^2)\) (where \(n = SIZE\)), because for each **ONE** of the SIZE values \(i\) takes on (e.g. 0 to 255), \(j\) will iterate through all SIZE (256) values. This leads to SIZE*SIZE steps of work.
Q8) Write a function, `evenDigits` that takes a single integer, `x`, and counts how many of the digits in the number are **even** and how many are **odd**.

The function should **output** the count of **even** digits in the number AND a **boolean** that is true if the number of *even* digits is larger than the number of *odd* digits, and false otherwise.

**Example 1:** If the user inputs `123` then there is only 1 even digit (i.e. 2) and we would also output false since there are NOT more even digits than odd.

**Example 2:** If the user inputs `98864201` then there are 7 even digits and we would also output true since there are more even digits than odd.

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

// Complete the prototype for your function. Either the Boolean
// of the integer representing the number of even digits needs
// to be passed by reference. You can choose either. We choose `neven`.
bool evenDigits( int num, int* neven );

int main()
{
    int num;
    bool moreEven = false;
    int numEvenDigs = 0;
    // Get the integer
    cin >> num;
    // Add 1-4 lines of code to call evenDigits and
    // update moreEven and numEvenDigs
    moreEven = evenDigits(num, &numEvenDigs);

    // Print results
    cout << "Number of even digits: " << numEvenDigs << endl;
    cout << "More even digits than odd: " << boolalpha << moreEven << endl;
    return 0;
}

// Write the definition of your function here
bool evenDigits( int num, int* neven )
{
    int nodd = 0;
    while(num != 0){
```
if( num % 2 == 0){  // same as (num % 10) % 2
    (*neven)++;
} else{
    nodd++;
}
num /= 10;
return *neven > nodd;