EECE.2160: ECE Application Programming

Spring 2016

Exam 2

March 30, 2016

Name: ____

Section (circle 1): **201** *(8-8:50, P. Li)* **202** *(12-12:50, M. Geiger)*

For this exam, you may use only one 8.5" x 11" double-sided page of notes. All electronic devices (e.g., calculators, cellular phones, PDAs) are prohibited. If you have a cellular phone, please turn it off prior to the start of the exam to avoid distracting other students.

The exam contains 3 questions for a total of 100 points. Please answer the questions in the spaces provided. If you need additional space, use the back of the page on which the question is written and clearly indicate that you have done so.

Please read each question carefully before you answer. In particular, note that:

- Question 3 has three parts, but you are only required to complete two of the three parts.
  - You may complete all three parts for up to 10 points of extra credit. If you do so, **please clearly indicate which part is the extra one—we will assume it is part (c) if you mark none of them.**
- For each part of Question 3, you must complete a short function. We have provided comments to describe what your function should do and written some of the code for you.
  - Note that each function contains both lines that are partially written (for example, a `printf()` call missing the format string and expressions) and blank spaces in which you must write additional code. **You must write all code required to make each function work as described—do not simply fill in the blank lines.**
  - Each test case is an example of how the function should behave in one specific case—**it does not cover all possible results of running that function.**
- You can solve each part of Question 3 using only the variables that have been declared, but you may declare and use other variables if you want.

You will have 50 minutes to complete this exam.

Q1: Multiple choice	/ 20
Q2: Arrays	/ 40
Q3: Functions	/ 40
TOTAL SCORE	/ 100
EXTRA CREDIT	/ 10

1. (20 points, 5 points per part) ***Multiple choice***
For each of the multiple choice questions below, clearly indicate your response by circling or underlining the one choice you think best answers the question.

a. What will the short code sequence below print?

```
int i;
for (i = 30; i > 0; i /= 2) {
   printf("%d ", i);
   i -= 2;
}
```

i. `30 15 7 3 1`

ii. `30 15 7 3 1 0`

iii. `30 14 6 2`

iv. `30 14 6 2 0`

v. `14 6 2 0`

b. Which of the following statements about one-dimensional arrays as function arguments are true?

A. If the array size is specified in the function prototype or definition (for example, `int f(int arr[10])`), then that function will only work with arrays of that size.

B. Arrays are always passed by address to functions.

C. If a function takes an array as an argument, it should also take the size of the array as an argument. The function can use that second argument to ensure array accesses are inside the bounds of the array.

D. If an array is passed to a function, the function cannot change any of the values stored in that array.

i. Only A

ii. Only B

iii. A and D

iv. B and C

v. All of the above (A, B, C, and D)

1 (continued)

c. Which of the following function prototypes correctly specifies a two-dimensional array as an argument?

i. `void f1(int arr[0][0], int n);`

ii. `int f2(double arr[][5], int n);`

iii. `double f3(double arr[][], int r, int c);`

iv. `void f4(int arr[10][], int c);`

v. None of the above

d. Which of the following statements accurately reflect your opinion(s)? Circle all that apply (but please don't waste too much time on this "question")!

i. "I think the most recent programming assignments are still pretty easy."

ii. "I think the programming assignments have gotten to be too difficult."

iii. "I think the programming assignments have gotten harder, but are still fair."

iv. "Is the semester over yet?"

2. (40 points) ***Arrays***
For each short program shown below, list the output exactly as it will appear on the screen. Be sure to clearly indicate spaces between characters when necessary.

You may use the available space to show your work as well as the output; just be sure to clearly mark where you show the output so that I can easily recognize your final answer.

a. (12 points)

```
int main() {
   int arr[10] = {3, 9, 4, 5, -3, 7, -1, 10, -4, 2};
   int i;

   for (i = 9; i > 0; i -= 3)
      printf("%d %d %d\n", arr[i], arr[i-1], arr[i-2]);

   for (i = 0; i < 10; i += arr[i])
      printf("%d\n", arr[i]);

   return 0;
}
```

2 (continued)

b. (14 points)

Please note that both printf() calls that print values from the array use a precision of 1.

```
int main() {
   double mat[2][5] = { {1.1, 2.2, 3.3, 4.4, 5.5},
                        {6.6, 7.7, 8.8, 9.9, 0.0} };
   int i, j;

   for (i = 0; i < 2; i++) {
      for (j = 4; j >= 0; j--) {
         printf("%.1lf ", mat[i][j]);    // Print with precision 1
      }
      printf("\n");
   }

   for (i = 9; i >= 0; i -= 2)
      printf("%.1lf\n", mat[i/5][i/2]); // Print with precision 1

   return 0;
}
```

2 (continued)

c. (14 points)

```
void f(int arr[], int st, int g, int inc) {
   int i;
   for (i = st; i + g < 8; i += inc)
      arr[i+g] = arr[i] + arr[i+g];
}

int main() {
   int vals[8] = {12, 15, 13, 3, 1};
   int i;

   for (i = 0; i < 8; i++)
      printf("%d ", vals[i]);
   printf("\n");

   f(vals, 0, 3, 2);
   for (i = 0; i < 8; i++)
      printf("%d ", vals[i]);
   printf("\n");

   f(vals, 4, -2, 3);
   for (i = 0; i < 8; i++)
      printf("%d ", vals[i]);
   printf("\n");

   return 0;
}
```

3. (40 points, 20 per part) ***Functions***

For each part of this problem, you are given a short program to complete. **CHOOSE ANY TWO OF THE THREE PARTS** and fill in the spaces provided with appropriate code. **You may complete all three parts for up to 10 points of extra credit, but must clearly indicate which part is the extra one—I will assume it is part (c) if you mark none of them.**

Remember, you must write all code required to make each function work as described—**do not assume you can simply fill in the blank lines and get full credit.** Also, remember that each example provided is only applicable in one specific case—**it does not cover all possible results of using that function.**

a. `void emptyBox(int w, int h);`

This function prints a box with width `w` and height `h`, where the outer edge of the box is represented by a series of `*` characters, and the inner part of the box contains only spaces. For example, `emptyBox(6, 3)` would print the following box:

```
******
*    *
******
```

```
void emptyBox(int w, int h) {
  int i, j;    // Loop indexes

  // Go through all possible box coordinates, row by row
  // Be sure that each row of output starts on a new line

  _________ (________________________________________________) {

    _________ (_____________________________________________) {

      // Determine appropriate output character


        printf("*");


        printf(" ");
    }

  }

}
```

3 (continued)
b. `int persistence(int v);`

If you multiplying the digits of an integer and continuing that process with each resulting product, the sequence of products always arrives at a single digit number. For example, starting with 715 yields the sequence 715 → 7 x 1 x 5 = 35 → 3 x 5 = 15 → 1 x 5 = 5.

The number of times products need to be calculated to reach a single digit is called the persistence of that integer (for example, the persistence of 715 is 3; the persistence of 9 is 0).

This function calculates and returns the persistence of an integer, `v`, by the process described in the comments below. Note: modulus and division operations can help you isolate and remove digits, starting with the lowest digit. (For example, using 715, you first isolate the 5, leaving 71. Then, isolate the 1, leaving 7. Then, isolate the 7, leaving 0.)

```
int persistence(int v) {
   int dig;     // Current digit
   int tot;     // Running total of multiplied values
   int p = 0;   // Persistence

   // Loop as long as v has >= 2 digits

   while (____________________________________________________) {

      // Reinitialize tot each time through this loop

      tot = _____________;

      // Isolate lowest digit of v, multiply tot by that digit, &
      //  remove that digit from v. Repeat until no digits remain

      while (________________________________________________) {





      }

      // Update persistence (to count this loop iteration)
      //   and v (to be the result of multiplying repeatedly)




   }

   return p;    // Return persistence
}
```

3 (continued)

c. `double winPctStreak(char res[], int ng, int *streak);`

Each entry in the character array `res[]`, which contains `ng` entries, represents the result of a game—`'W'` or `'w'` for a win, `'L'` or `'l'` (lowercase L) for a loss. The function should go through the entire array and calculate:

- The winning percentage—the percentage of `'W'` or `'w'` entries, which is between 0 and 100. This value should be returned at the end of the function.
- The longest winning streak (i.e., sequence of consecutive `'W'` or `'w'` entries). The argument `streak` points to the location where this value should be stored. Assume the value at this location is not initialized when the function is called.

For example, given `char sch[] = {'W','l','w','W','W','l','w','W'}`and `int str`, `winPctStreak(sch, 8, &str)` returns 75.0 and sets `str = 3`.

```
double winPctStreak(char res[], int ng, int *streak) {
   int i;              // Loop index
   double wins;        // Number of wins
   int cur;            // Current streak

   // Initialize variables as needed



   // Loop to go through res array

   _________ (________________________________________________) {

      // Count each win and update streak variables

      if (____________________________________________________) {






      }

      // For each loss, reset current streak
      else


   }

   // Calculate and return win percentage


}
```