EECE.2160: ECE Application Programming

Spring 2017

Exam 2
March 29, 2017

Name: _____

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

For this exam, you may use only one 8.5" x 11" double-sided page of notes. All electronic devices (e.g., calculators, cell phones) are prohibited. If you have a cell 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 is the output of the short code sequence below?

```
char s1[] = "Exam2";
char s2[15];
int i;

strncpy(s2, s1, 10);
printf("%s", s2);

for(i = 0; i < 10; i++)
    if(s2[i] >= '0' && s2[i] <= '9')
        s2[i]++;

strncat(s2, s1, 4);
printf("%s", s2);
```

i. `Exam2Exam3`

ii. `Exam1Exam2`

iii. `Exam2Exam3Exam3`

iv. `Exam2Exam3Exam2`

v. `Exam2Exam3Exam`

1 (continued)

b. What is the output of the short code sequence below? Remember the following function used in Program 6:

`int toupper(int ch)`: If `ch` is a lowercase letter, this function returns the uppercase version. Otherwise, `toupper()` returns the original value of `ch`. For example, `toupper('q')` returns `'Q'`; `toupper('A')` returns `'A'`.

```
char s1[] = "ABCD";
char s2[] = "abcd";
int i;

for(i = 0; i < strlen(s1) - 1; i++)
    s2[i] = toupper(s2[i]);
printf("%s ", s2);

if( strncmp (s1, s2, 3) == 0 )
    printf("s1 == s2");
else
    printf("s1 != s2");
```

i. `ABCD s1 == s2`

ii. `ABCD s1 != s2`

iii. `ABCd s1 == s2`

iv. `ABCd s1 != s2`

v. `abcD s1 != s2`

1 (continued)

c. You are given the structure definition below:

```
typedef struct {
  char name[10];
  int id;
  double salary;
} Employee;
```

Your program declares a single `Employee` variable: `Employee e1;`

Which of the following groups of statements will compile correctly and set the `salary` field inside `e1` to be 75000?

A. `e1.salary = 75000;`

B.
```
Employee *ep = &e1;
ep.salary = 75000;
```

C. `Employee.salary.e1 = 75000;`

D.
```
Employee *ep = &e1;
ep->salary = 75000;
```

i. Only A

ii. Only B

iii. A and D

iv. B and C

v. A, C, and D

1 (continued)

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 assume all appropriate libraries are included.

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[7] = {2, 4, 1};
   int i;

   for (i = 1; i < 6; i++) {
    arr[i] = arr[i-1] + arr[i];
    printf("%d ", arr[i]);
   }
   printf("\n");

   for (i = 0; i < 7; i += 2) {
      arr[i] = arr[i%3] + arr[i/2];
      printf("%d ", arr[i]);
   }
   printf("\n");
   return 0;
}
```

2 (continued)

b. (13 points)

```
int main() {
   int i, j;
   int arr[4][2] = {{7, 5}, {4, 3}, {1, 9}, {12, 2}};

   for (j = 0; j < 2; j++) {
      for (i = 0; i < 4; i++) {
         arr[i][j] = arr[i][j] / 2;
         printf("%d ", arr[i][j]);
      }
      printf("\n");
   }

   j = 0;
   for (i = 0; i < 4; i++) {
      arr[i][j]++;
      for (j = 0; j < 2; j++) {
         if (arr[i][j] % 2)
            arr[i][j]++;
         printf("%d ", arr[i][j]);
      }
   }
   return 0;
}
```

2 (continued)

c. (14 points)

```
void f(double arr[], int n, int step) {
   double temp;
   int i;
   if (n <= step) {
      printf("Array not changed.\n");
      return;
   }
   else {
      for (i = 0; i < n - step; i++) {
         temp = arr[i];
         arr[i] = arr[i + step];
         arr[i + step] = temp;
      }
      for (i = 0; i < n; i ++) {
         printf("%0.lf ", arr[i]);
      }
      printf ("\n");
   }
}

int main () {
   double myArr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

   f(myArr, 10, 2);
   f(myArr, 4 , 5);
   f(myArr, 6 , 3);
   return 0;
}
```

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

For each part of this problem, you are given a short function 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.**

In order to allow plenty of space to solve each problem, this page is essentially just a "cover sheet" for Question 3—**the actual problems start on the next page**.

3 (continued)

a. `void printTriangle(int len);`

This function prints a right triangle with two equal sides of length `len`. The outer edge of the triangle is represented by a series of `*` characters, and the inner part of the box contains only spaces. The triangle should be printed so that the hypotenuse (the side opposite the right angle) goes from the top right corner to the bottom left corner. For example, `printTriangle(4)` would print the following:

```
   *      (three spaces followed by a star)
  **      (two spaces followed by two stars)
 * *      (one space, one star, one space, one star)
****      (no spaces, four stars)
```

```
void printTriangle(int len) {
   int i, j, k;      // Loop indexes (you may not need all 3)

   // Outer loop controls "rows" of output--make sure each "row"
   //   starts on a new line

   _________ (_____________________________________________) {

      // Inner loop controls "columns" of output

      _________ (_____________________________________________)


         // Test to see if current column is on border of triangle
         // If so, print star

         _________ (_____________________________________________

                   _____________________________________________)

            printf("*");

         // Otherwise, print space



            printf(" ");


      } // End inner loop


   } // End outer loop--remember, each "row" starts on new line

}
```

3 (continued)
b. `int isPerfect(int num);`

A perfect number is a positive integer that equals the sum of its positive factors (the numbers that evenly divide into it), excluding the number itself. For example, the lowest perfect number is 6, because the factors of 6 are 1, 2, and 3, and 1 + 2 + 3 = 6.

This function takes a single argument, `num`, and returns 1 if `num` is a perfect number and 0 otherwise. To determine if a number is perfect, find all of its factors, add them together, and check if the sum of the factors matches the original number.

Note: For full credit, your loop that finds all factors should not test any values that cannot possibly be factors of `num`. Hint: think about the smallest possible factors of any number and what the corresponding largest factors can be. Remember to exclude the original number itself!

```
int isPerfect(int num) {
   int i;       // Loop index
   int sum;     // Running total of factors

   // Initialize variables as needed



   // Return 0 for any value too small to be a perfect number

   if (_____________________________________________________________)
      return 0;

   // Loop to find all factors of num and add them up
   // Remember, for full credit, this loop shouldn't test any
   //   value that can't possibly be a factor of num

   _________ (_____________________________________________________) {









   }

   // If number is perfect, return 1; otherwise, return 0

   if (_____________________________________________________________)
      return 1;
   else
      return 0;
}
```

3 (continued)

c. `void avgSD(double list[], int n, double *avg, double *sd);`

Given an array, `list[]`, containing `n` double-precision values, calculate the average and standard deviation of the array and store them at the addresses pointed to by `avg` and `sd`, respectively. To calculate the standard deviation of a list of values, follow the steps below, using {7, 5, 3, 1} as an example list:

- Calculate the average of all values (which is 4 for the example values)
- Go through the list and add up the squares of the differences between each value and the average (for this example, $(7 - 4)^2 + (5 - 4)^2 + (3 - 4)^2 + (1 - 4)^2 = 9 + 1 + 1 + 9 = 20$)
- Divide by the number of values (for this example, 20 / 4 = 5)
- The standard deviation is the square root of that result, which you can find using the `sqrt()` function from the math library (for this example, `sqrt(5)` ≈ 2.236)

```
void avgSD(double list[], int n, double *avg, double *sd) {
   double sum;       // Running total
   int i;            // Loop index

   // Calculate average


   for (_____________________________________________________________) {



   }



   // Calculate standard deviation


   for (_____________________________________________________________) {



   }


}
```