EECE.2160: ECE Application Programming

Spring 2017

Exam 2 Solution

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);   Copies s1 ("Exam2") to s2
printf("%s", s2);      and prints it

for(i = 0; i < 10; i++)                      Check all characters in
    if(s2[i] >= '0' && s2[i] <= '9')         s2; if character is a
        s2[i]++;                             digit, increment it
                                             (changes '2' to '3')

strncat(s2, s1, 4);    Concatenate 1st 4 chars of s1 ("Exam")
printf("%s", s2);      to s2 (now "Exam3Exam") and print it
```

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++)    Changes 1st 3 chars of s2
    s2[i] = toupper(s2[i]);             to uppercase letters
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. ***<u>A and D</u>***

iv. B and C

v. A, C, and D

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?"

<u>All of the above are "correct."</u>

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};     arr[7] = {2, 4, 1, 0, 0, 0, 0}
   int i;

   for (i = 1; i < 6; i++) {       Add consecutive elements
    arr[i] = arr[i-1] + arr[i]; arr[1] = arr[0]+arr[1] = 2+4 = 6
    printf("%d ", arr[i]);      arr[2] = arr[1]+arr[2] = 6+1 = 7
   }                            arr[3] = arr[2]+arr[3] = 7+0 = 7
   printf("\n");                arr[4] = arr[3]+arr[4] = 7+0 = 7
                                arr[5] = arr[4]+arr[5] = 7+0 = 7

   for (i = 0; i < 7; i += 2) {    Modify arr[i] (i = 0,2,4,6)
     arr[i] = arr[i%3] + arr[i/2];
     printf("%d ", arr[i]);    arr[0] = arr[0]+arr[0] = 2+2 = 4
   }                           arr[2] = arr[2]+arr[1] = 7+6 = 13
   printf("\n");               arr[4] = arr[1]+arr[2] = 13+6 = 19
   return 0;                   arr[6] = arr[0]+arr[3] = 4+7 = 11
}
```

OUTPUT:

```
6 7 7 7 7
4 13 19 11
```

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++) {          Print array column-by-column
      for (i = 0; i < 4; i++) {       after dividing value by 2
         arr[i][j] = arr[i][j] / 2;   Column 0: 3 2 0 6
         printf("%d ", arr[i][j]);    Column 1: 2 1 4 1
      }
      printf("\n");
   }

   j = 0;
   for (i = 0; i < 4; i++) {
      arr[i][j]++;                    ← OUT OF BOUNDS WHEN i > 0
      for (j = 0; j < 2; j++) {            (since j == 2)
         if (arr[i][j] % 2)           ← If arr[i][j] is odd,
            arr[i][j]++;                   increment it
         printf("%d ", arr[i][j]);    Prints every value in the
      }                               array on a single row
   }
   return 0;
}
```

The second set of loops actually goes outside the array bounds—the statement marked OUT OF BOUNDS above goes past the end of rows 1, 2, and 3.

The solution "assumes" that going past the end of a row in a 2-D array accesses the first element in the next row (at least, that's what happened when I ran the program on my computer). Therefore, arr[2][0] goes from 0 to 1 (and later from 1 to 2), and arr[3][0] goes from 6 to 7 (and later from 7 to 8).

When this exam was graded, I threw out this problem and gave everyone full credit, but the explanation above hopefully helps you understand how the output below was generated.

OUTPUT:

```
3 2 0 6
2 1 4 1
4 2 2 2 2 4 8 2
```

2 (continued)

c. (14 points)

```
void f(double arr[], int n, int step) {
   double temp;
   int i;
   if (n <= step) {                         Function does nothing
      printf("Array not changed.\n");       if array is too small to
      return;                               have values separated by
   }                                        "step" positions
   else {
      for (i = 0; i < n - step; i++) {      Swaps array elements
         temp = arr[i];                      that are separated by
         arr[i] = arr[i + step];             "step" positions. Has
         arr[i + step] = temp;               the effect of moving
      }                                      that number of values
      for (i = 0; i < n; i ++) {             from the beginning of
         printf("%0.lf ", arr[i]);           the array to the end.
      }                                     Array is printed once
      printf ("\n");                         swaps are done
   }
}

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

   f(myArr, 10, 2);  Moves 1st 2 values (1, 2) to positions 8 & 9
   f(myArr, 4 , 5);  Does nothing—2nd argument greater than 1st
   f(myArr, 6 , 3);  Moves 1st 3 values (3, 4, 5) to positions 3-5
   return 0;
}
```

OUTPUT:

```
3 4 5 6 7 8 9 10 1 2
Array not changed.
6 7 8 3 4 5
```

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

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)
```

Students were required to write bold, underlined, italicized code.

```
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
   for (i = len; i > 0; i--) {       // Fixed "i++" typo

      // Inner loop controls "columns" of output
      for (j = 1; j <= len; j++)

         // Test to see if current column is on border of triangle
         // If so, print star
         if ((i == 1) ||              // Bottom row
             (j == len) ||            // Right side of triangle
             (j == i))                // Hypotenuse of triangle
            printf("*");

         // Otherwise, print space
         else
            printf(" ");
      } // End inner loop

      printf("\n");
   }  // 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!

Students were required to write bold, underlined, italicized code.

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

   // Initialize variables as needed
   sum = 1;

   // Return 0 for any value too small to be a perfect number
   if (num < 6)
      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
   for (i = 2; i <= num / 2; i++) {
      if (num % i == 0)
         sum = sum + i;
   }

   // If number is perfect, return 1; otherwise, return 0
   if (sum == num)
      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)

Students were required to write bold, underlined, italicized code.

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

   // Calculate average
   sum = 0;

   for (i = 0; i < n; i++) {
      sum += list[i];
   }
   *avg = sum / n;

   // Calculate standard deviation
   sum = 0;
   for (i = 0; i < n; i++) {
      sum += (list[i] - *avg) * (list[i] - *avg);
   }
   *sd = sqrt(sum / n);
}
```