EECE.2160: ECE Application Programming

Fall 2016

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. Which values listed below for the strings `s1` and `s2` will cause the function `strncmp(s1, s2, 3)` to return a nonzero value?

A. `s1 = "string", s2 = "strike"`

B. `s1 = "String", s2 = "string"`

C. `s1 = "Test", s2 = "Test"`

D. `s1 = "Test", s2 = "Exam"`

i. Only C

ii. Only D

iii. A and C

iv. B and D

v. A, B, and D

1 (continued)

b. What will the short code sequence below print?

```
int i;
char s1[50] = "E";
char s2[50] = "C";
for (i = 0; i < 4; i++) {
   if (i % 2 == 0)
      strcat(s1, s2);
   else
      strcat(s2, s1);
}
printf("%s %s\n", s1, s2);
```

i. `EC CE`

ii. `EC CEC`

iii. `ECC CEE`

iv. `ECCEC CEC`

v. ECCEC CECECCEC

c. What does the following short code sequence print?

```
char s1[20];
char s2[20];

strcpy(s1, "Exam 2");
strncpy(s2, s1, 2);
s2[2] = '\0';
s2[3] = 'A';
printf("%s %s\n", s1, s2);
```

i. `Exam 2 Exam 2`

ii. Exam 2 Ex

iii. `Exam 2 ExA`

iv. `Ex Exam 2`

v. None of the above

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?"

All of the above are "correct."

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. (13 points)

```
int main() {
   double list[9] = {1.23, 4.56, 7.89, 10.23, 45.67, 89.01};
   int i;

   for (i = 8; i >= 0; i -= 2) {
      printf("%.2lf\n", list[i]);
      if (list[i] == 0)
         list[i] = i / 4.0;
   }

   Loop below prints values starting (almost) in middle of array
     and working way out to ends:
      • When i = 1, prints list[3] and list[5]
      • When i = 2, prints list[2] and list[6]
      • When i = 3, prints list[1] and list[7]
      • When i = 4, prints list[0] and list[8]
   for (i = 1; i <= 4; i++)
      printf("%.2lf %.2lf\n", list[4-i], list[4+i]);

   return 0;
}
```

Prints list[8], list[6], list[4], list[2], & list[0]

Affects list[8] and list[6], since both are 0 (declaration of list[] only contains 6 initial values, so last 3 elements are 0).

Sets list[8] = 8 / 4.0 = 2.0 and list[6] = 6 / 4.0 = 1.5

OUTPUT:

```
0.00
0.00
45.67
7.89
1.23
10.23 89.01
7.89 1.50
4.56 0.00
1.23 2.00
```

2 (continued)
b. (13 points)

```
int main() {
   int arr[3][3] = { {2, 0, 1}, {1, -5, 3}, {2, 15, -4} };
   int i, j;
   int r, c;

   for (i = 0; i < 3; i++) {
      for (j = 2; j >= 0; j--) {
         printf("%d ", arr[i][j]);
      }
      printf("\n");
   }

   for (i = 0; i < 3; i++) {
      r = arr[i][0];
      c = arr[0][i];
      printf("%d\n", arr[r][c]);
   }

   return 0;
}
```

Prints array contents with rows in order but columns backwards

Prints 3 elements from array, using first column of array for row numbers and first row of array for column numbers:

- ***i = 0 → r = c = arr[0][0] = 2 → print arr[2][2] = -4***
- ***i = 1 → r = arr[1][0] = 1, c = arr[0][1] = 0***
  → print arr[1][0] = 1
- ***i = 2 → r = arr[2][0] = 2, c = arr[0][2] = 1***
  → print arr[2][1] = 15

OUTPUT:

```
1 0 2
3 -5 1
-4 15 2
-4
1
15
```

2 (continued)

c. (14 points)

```
Function uses values in arr2[] as indexes into arr1[] if index
  values fall within array bounds for arr1[] (0 <= index < n1).
If arr2[i] is valid index into arr1[], add i to arr1[] value.
void f(int arr1[], int n1, int arr2[], int n2) {
   int i;
   for (i = 0; i < n2; i++) {
      if ((arr2[i] >= 0) && (arr2[i] < n1))
         arr1[arr2[i]] += i;
   }
}

int main() {
   int list1[] = {100, 8, 5, 3, -11, 2, 0, 1, 4};
   int list2[] = {8, 6, 7, 5, 3, 0, 9};
   int i;

   f(list1, 9, list2, 7);      list1[8] = 4 + 0 = 4
   for (i = 0; i < 9; i++)     list1[6] = 0 + 1 = 1
      printf("%d ", list1[i]); list1[7] = 1 + 2 = 3
   printf("\n");               list1[5] = 2 + 3 = 5
                               list1[3] = 3 + 4 = 7
                               list1[0] = 100 + 5 = 105
                               list1[9] does not exist

   Remember, list1 is changed by the first call to f(), so now:
     list1[] = {105, 8, 5, 7, -11, 5, 1, 3, 4}
   f(list2, 7, list1, 9);      list2[105] does not exist
   for (i = 0; i < 7; i++)     list2[8] does not exist
      printf("%d ", list2[i]); list2[5] = 0 + 2 = 2
   printf("\n");               list2[7] does not exist
                               list2[-11] does not exist
                               list2[5] = 2 + 5 = 7
                               list2[1] = 6 + 6 = 12
                               list2[3] = 5 + 7 = 12
                               list2[4] = 3 + 8 = 11

   return 0;
}
```

OUTPUT:

```
105 8 5 7 -11 5 1 3 4
8 12 7 12 11 7 9
```

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);
}
```