16.216: ECE Application Programming

Fall 2014

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. Given strings `s1 = "Exam 1"`, `s2 = "Exam 2"`, and `s3 = "Exactly"`, which of the following calls to string comparison functions will return the value 0?

A. `strcmp(s1, s2);`
B. `strncmp(s1, s2, 5);`
C. `strcmp(s1, s3);`
D. `strncmp(s1, s3, 3);`

i. Only A

ii. Only D

iii. A and C

iv. B and D

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

b. Given the short code sequence below:

```
int i;
char str[20] = "x";
for (i = 0; i < 4; i++)
   strcat(str, "ox");
printf("Length of string = %d\n", strlen(str));
```

What will this program print?

i. `Length of string = 1`

ii. `Length of string = 3`

iii. `Length of string = 5`

iv. ***`Length of string = 9`***

v. `Length of string = 20`

c. Given the code sequence below:

```
char s1[20];
char s2[20];
strcpy(s1, "Summer");
s1[1] = 'i';
strncpy(s2, s1, 4);
s2[3] = '\0';
printf("%s  %s\n", s1, s2);
```

What will this program print?

i. `Summer  Simmer`

ii. `Simmer  Simmer`

iii. `Summer  Simm`

iv. `Simmer  Simm`

v. ***`Simmer  Sim`***

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. (12 points)

```
void main() {
   int i;
   double list[6] = { 1.2, 3.4, 5.6, 7.8, 9.0 };
```

Not shown: list[5] = 0.0

Each iteration of this first loop prints two values that are equidistant from the start and end of the array (the first and last values; the second and second to last values, etc.) This loop accounts for the first three lines of output below.

```
   // Each value is printed with precision of one
   for (i = 0; i < 3; i++) {
      printf("%.1lf %.1lf\n", list[i], list[5 - i]);
   }
```

The second loop starts with the last value in the loop, and each iteration prints the difference between an array value and the value directly before it. This loop accounts for the last five lines of output.

```
   for (i = 5; i > 0; i--) {
      printf("%.1lf\n", list[i] - list[i - 1]);
   }
}
```

OUTPUT:

```
1.2 0.0
3.4 9.0
5.6 7.8
-9.0
1.2
2.2
2.2
2.2
```

2 (continued)

b. (14 points)

```
void main() {
   int i, j;
   int arr[4][2];

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

   for (i = 7; i >= 0; i--) {
      printf("%d ", arr[i % 4][i % 2]);
   }
}
```

Initialize the array so that each element is the sum of its row and column numbers, and print the array row by row (first 4 output lines)

Generate a single output line showing the following elements: arr[3][1], arr[2][0], arr[1][1], arr[0][0] Each element is shown twice because the results of the modulus operations are exactly the same in the last 4 iterations as they are in the first 4.

OUTPUT:

```
0 1
1 2
2 3
3 4
4 2 2 0 4 2 2 0
```

2 (continued)

c. (14 points)

```
void f(int arr[], int n) {
   int i;
   for (i = 0; i < n / 2; i++) {
      arr[n - 1 - i] += arr[i];
   }
}
```

Function goes through n/2 elements of arr, changing the highest elements by adding the lowest elements. Highest element changed depends on value of n.

Note that highest element accessed by function isn't actually changed because list[0], which is always added to list[n-1], is always 0.

```
void main() {
   int i;
   int list[] = { 0, 2, 4, 6, 8, 9, 7, 5, 3, 1 };

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

   f(list, 6);
   for (i = 0; i < 10; i++)
      printf("%d ", list[i]);
   printf("\n");

   f(list, 2);
   for (i = 0; i < 10; i++)
      printf("%d ", list[i]);
   printf("\n");
}
```

Changes last 5 elements (n = 10 → n/2 = 5)

Changes elements 3-5 (n = 6 → n/2 = 3)

Changes element 1 (n = 2 → n/2 = 1)

OUTPUT:

```
0 2 4 6 8 17 13 9 5 1
0 2 4 10 10 17 13 9 5 1
0 2 4 10 10 17 13 9 5 1
```

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 splitDay(int doy, int *m, int *dom);`

Given an integer, `doy`, representing the day within a year of 365 days, determine the appropriate month and day and store them in the variables pointed to by `m` and `dom`, respectively. For example, calling `splitDay(10, &month, &day)` would set `month = 1` and `day = 10`, since the 10th day of the year is January 10; `splitDay(360, &month, &day)` would set `month = 12` and `day = 26`, as the 360th day of the year is December 26.

Students were responsible for writing bold, underlined, italicized code.

```
void splitDay(int doy, int *m, int *dom) {
   int daysPerMo[] = { 31, 28, 31, 30, 31, 30,     // Days in
                       31, 31, 30, 31, 30, 31 };  //  each month
   int sum;     // Sum of days after a certain number of months

   // Initialize variables
   *m = 0;
   sum = 0;

   // Find the month by adding the number of days in all previous
   //   months--when you've added too many days, the previous
   //   month is the correct one
   while (sum + daysPerMo[*m] < doy) {
      sum += daysPerMo[*m];
      (*m)++;
   }

   // Finalize month and day values--day of month is what's left
   //   in the partial month after you've added all prior months
   (*m)++;
   *dom = doy - sum;
}
```

3 (continued)

b. `int countFib(int lo, int hi);`

This function counts the number of values in the Fibonacci sequence that fall between the endpoints lo and hi, including the endpoints. The first two Fibonacci numbers are 0 and 1, and every subsequent value in the sequence is the sum of the two numbers that precede it: **0, 1, 1, 2, 3, 5, 8, 13, 21, 34,** and so on. Therefore, for example, `countFib(1,10)` would return 6 and `countFib(1, 35)` would return 9.

Students were responsible for writing bold, underlined, italicized code.

```
int countFib(int lo, int hi) {
   int num1, num2;   // Numbers in Fibonacci sequence
   int count;        // Counter

   // Initialize variables
   num1 = 0;
   num2 = 1;
   count = 0;

   // Special case: if lo is 0 or negative, count one more value
   if (lo < 1)
      count = 1;

   // Go through Fibonacci values below low endpoint before
   //   starting count
   while (num2 < lo) {
      num2 = num1 + num2;
      num1 = num2 - num1;
   }

   // Now generate and count Fibonacci values between endpoints
   while (num2 <= hi) {
      num2 = num1 + num2;
      num1 = num2 - num1;
      count++;
   }

   return count;
}
```

3 (continued)

c. `int findMode(int arr[], int n);`

This function finds and returns the mode—the value that occurs most often—in an integer array `arr[]` of length `n`. For example, if `arr = {1, 2, 3, 3, 4, 4, 4}`, the mode is 4; if `arr = {1, 6, 2, 1, 6}`, the mode is 6.

To find the mode, the function counts the number of occurrences of each value in `arr[]`, and then determines the maximum number of occurrences, which in turn determines the mode. You may assume `arr[]` contains only values between 1 and 10, so the `numCount[]` array declared in the function can be used to count the occurrences of all possible values.

Students were responsible for writing bold, underlined, italicized code.

```
int findMode(int arr[], int n) {
   int numCount[10]; // Counts number occurrences in arr
   int i;            // Loop index
   int mode;              // Mode

   // Initialize variables as needed
   for (i = 0; i < 10; i++)
      numCount[i] = 0;

   // Count # of occurrences of each value in arr
   for (i = 0; i < n; i++) {
      numCount[arr[i] - 1]++;
   }

   // Determine mode--find which number occurred most often
   mode = 1;
   for (i = 1; i < 10; i++) {
      if (numCount[i] > numCount[mode - 1])
         mode = i + 1;
   }

   return mode;
}
```