EECE.2160: ECE Application Programming

Spring 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. 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 <u>one-dimensional</u> 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. ***<u>B and C</u>***

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?"

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)

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

   The loop below prints elements from the array in groups
     of 3, starting with the last element (element 9). Those
     values are printed in reverse order, so, for example,
     the first loop iteration prints arr[9], arr[8], and arr[7]
   The loop runs 3 times, printing all values except arr[0].
   for (i = 9; i > 0; i -= 3)
      printf("%d %d %d\n", arr[i], arr[i-1], arr[i-2]);

   This loop starts with the first loop element and uses the
     contents of the array to determine which element it
     accesses next—i changes by adding the current element each
     time. The loop therefore does the following:

   • i = 0: print arr[0] = 3, i = i + arr[0] = 0 + 3 = 3
   • i = 3: print arr[3] = 5, i = i + arr[3] = 3 + 5 = 8
   • i = 8: print arr[8] = -4, i = i + arr[8] = 8 + (-4) = 4
   • i = 4: print arr[4] = -3, i = i + arr[4] = 4 + (-3) = 1
   • i = 1: print arr[1] = 9, i = i + arr[1] = 1 + 9 = 10
   • Loop ends at that point

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

   return 0;
}
```

OUTPUT:

```
2 -4 10
-1 7 -3
5 4 9
3
5
-4
-3
9
```

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;

   This loop prints each row of the array in reverse, since j
     (the column index) goes from 4 to 0, not 0 to 4.
   for (i = 0; i < 2; i++) {
      for (j = 4; j >= 0; j--) {
         printf("%.1lf ", mat[i][j]);    // Print with precision 1
      }
      printf("\n");
   }

   This loop prints 5 values from the array, where the indices
     are determined using integer division with i. That variable
     decreases by 2 each iteration, giving the following:

   • i = 9: print mat[9/5][9/2] = mat[1][4] = 0.0
   • i = 7: print mat[7/5][7/2] = mat[1][3] = 9.9
   • i = 5: print mat[5/5][5/2] = mat[1][2] = 8.8
   • i = 3: print mat[3/5][3/2] = mat[0][1] = 2.2
   • i = 1: print mat[1/5][1/2] = mat[0][0] = 1.1

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

   return 0;
}
```

OUTPUT:

```
5.5 4.4 3.3 2.2 1.1
0.0 9.9 8.8 7.7 6.6
0.0
9.9
8.8
2.2
1.1
```

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];
}
```

This function goes through the array, starting at position st, and adds together elements separated by g spaces. inc determines the increase in i each iteration.

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

Since size of array is 8, vals[8] = {12, 15, 13, 3, 1, 0, 0, 0}

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

Prints original array contents

```
   f(vals, 0, 3, 2);
```

Sets vals[3] = vals[0] + vals[3],
vals[5] = vals[2] + vals[5],
vals[7] = vals[4] + vals[7]

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

   f(vals, 4, -2, 3);
```

Sets vals[2] = vals[4] + vals[2],
vals[5] = vals[7] + vals[5]

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

   return 0;
}
```

OUTPUT:

```
12 15 13 3 1 0 0 0
12 15 13 15 1 13 0 1
12 15 14 15 1 14 0 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 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:

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

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

```
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
   for (i = 0; i < h; i++) {
      for (j = 0; j < w; j++){

         // Determine appropriate output character
         if ((i == 0) || (i == h - 1) || (j == 0) || (j == w-1))
            printf("*");

         else
            printf(" ");
      }
      printf("\n");
   }
}
```

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.)

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

```
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 (v >= 10) {

    // Reinitialize tot each time through this loop
    tot = 1;

    // Isolate lowest digit of v, multiply tot by that digit, &
    //  remove that digit from v. Repeat until no digits remain
    while (v > 0) {
      dig = v % 10;
      tot = tot * dig;
      v = v / 10;
    }

    // Update persistence (to count this loop iteration)
    //   and v (to be the result of multiplying repeatedly)
    p++;
    v = tot;
  }

  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`.

<u>Students were required to write bold, underlined, italicized code.</u>

```
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
   wins = 0;
   cur = 0;
   *streak = 0;

   // Loop to go through res array
   for (i = 0; i < ng; i++) {

      // Count each win and update streak variables
      if ((res[i] == 'W') || (res[i] == 'w')) {
         wins++;
         cur++;
         if (cur > *streak)
            *streak = cur;
      }

      // For each loss, reset current streak
      else
         cur = 0;
   }

   // Calculate and return win percentage
   return (wins / ng) * 100;
}
```