16.216: ECE Application Programming

Fall 2013

Exam 1 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 choice you think best answers the question.

a. What is the output of the short code sequence below?

```
int i;
for (i = -1; i > -6; i *= -2) {
   printf("%d ", i);
}
```

i. Nothing

ii. `-1`

iii. `-1 2`

iv. `-1 2 -4`

v. -1 2 -4 8

b. What is the output of the short code sequence below?

```
int x = 18;
while (x > 0) {
   printf("%d ", x);
   x = x - 6;
}
```

i. `12 18`

ii. `18 12`

iii. `6 12 18`

iv. 18 12 6

v. `18 12 6 0`

1 (continued)

c. Given the code sequence below:

```
int x, y;
do {
   scanf("%d %d", &x, &y);
} while ((x < 0) || (y > 0));
```

Which of the following possible input pairs will cause the do-while loop to end? In other words, which value(s) will cause the loop condition to be false?

A. 0 0
B. 1 1
C. -1 -1
D. -3 5
E. 5 -3

i. A and D

ii. A and E

iii. B and C

iv. B, C, and D

v. A, B, C, and E

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. "This course is moving too quickly."

ii. "This course is moving too slowly."

iii. "I've attended very few lectures, so I don't really know what the pace of the course is."

iv. "I appreciate the opportunity to take a 50 minute nap three times a week."

v. "I hope the rest of the exam is this easy."

All of the above are "correct."

2. (40 points) ***C input/output; operators***

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 x, y, z;
     double q = 16.216;

     x = 15 / (2 + 2) * 3;          x = 15 / 4 * 3
                                      = 3 * 3 = 9

     y = (x + 5) % (x - 5);         y = (9 + 5) % (9 – 5)
                                      = 14 % 4 = 2

     z = q + x;                     z = 16.216 + 9 = 25.216 = 25
                                        (truncated to int)

     q = x / y;                     q = 9 / 2 = 4
                                        (q is type double, but x
                                        and y are both ints, so
                                        you use integer division
                                        to compute this result.)


     printf("%d %d", x, y);
     printf("%d", z);
     printf("\n%lf", q);
}
```

OUTPUT: *Note that the only newline in this program is in the third call to* `printf()`*, so* `x`*,* `y`*, and* `z` *are all printed on the same line, with no space between the values of* `y (2)` *and* `z (25)`*.*

9 225

4.000000

2 (continued)

b. (14 points)

```
void main() {
     int i;

     double d1 = 5.0;
     double d2, d3;

     i = d1 / 2 + 3.5;          i = 5.0 / 2 + 3.5 = 2.5 + 3.5 = 6
                                     (truncated to int, although
                                     the value doesn't change)

     d2 = d1 / 100.0;           d2 = 5.0 / 100.0 = 0.05

     d3 = i + 0.12786;          d3 = 6 + 0.12786 = 6.12786

     d1 = d1 / 10;              d1 = 5.0 / 10 = 0.5

     printf("%.4d\n", i);
     printf("%+.0lf %-6.2lf %07.4lf\n", d1, d2, d3);
}
```

OUTPUT: *Note that:*

- *For this problem, spaces are shown in red below*
- *Printing an integer with a precision of 4 is equivalent to specicfying a field width of 4 with leading zeroes, which explains the three zeroes printed before the digit 6.*
- *There is a single space between +1 and 0.05, since that second value is left-justified within its field..*
- *There are three spaces between 0.05 and 06.1279. The first two are part of the field used to print 0.05—that value is printed with a field width of 6. The third space is simply the space between format specifiers.*

```
0006
+1 0.05   06.1279
```

2 (continued)

c. (14 points)

For this program, assume the user inputs the line below. The digit `'5'` is the first character the user types. Each space between numbers is a single space character (`' '`). You must determine how `scanf()` handles this input and then print the appropriate results.

```
  5 7 -17.3 200
```

```
void main() {
     int int1, int2;
     double d1, d2;
     char ch1, ch2, ch3, ch4;

     scanf("%d %lf %c %d %c %lf%c%c",
          &int1, &d1, &ch1,
          &int2, &ch2, &d2,
          &ch3, &ch4);

     printf("%d %d\n", int1, int2);
     printf("%.3lf %.3lf\n", d1, d2);
     printf("%c%c%c%c\n", ch1, ch2, ch3, ch4);
}
```

This program reads its input as follows:

- *`int1` is the first integer value* → *`int1 = 5`*
- *`d1` holds the next whole number* → *`d1 = 7`*
- *`ch1` holds the first non-space character after `7`* → *`ch1 = '-'`*
- *`int2` holds the next integer value* → *`int2 = 17`*
- *`ch2` is the first non-space character after `17`* → *`ch2 = '.'`*
- *`d2` holds the next whole number* → *`d2 = 3`*
- *`ch3` is the first character after `3`* → *`ch3 = ' '`*
- *`ch4` is the first non-space character after `.216`* → *`ch4 = '2'`*

<u>OUTPUT:</u> *Note that both double-precision values are printed with a precision of 3.*

```
5 17
7.000 3.000
-. 2
```

3. (40 points, 20 per part) ***C input/output; conditional statements***

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.**

a. This program should prompt for and read a real number representing a wavelength, then test to see if that wavelength falls into one of three ranges of visible light: blue (450-495 nm), yellow (570-590 nm), or red (620-750 nm). If the value is in none of those ranges, the program should indicate the wavelength is unknown. All ranges include the given endpoints.

In all cases, reprint the wavelength using one decimal place, as shown in the test cases below (user input is underlined):

```
Enter WL (nm): 470       Enter WL (nm): 630.5     Enter WL (nm): 600.179
470.0 nm --> blue        630.5 nm --> red         600.2 nm --> unknown
```

Students were responsible for the code shown in bold, underlined font.

```
void main() {
   double wav;       // Wavelength input by user

   // Prompt for and read input wavelength
   printf("Enter WL (nm): ");

   scanf("%lf", wav);

   // Test for blue light and print appropriate output
   if ((wav >= 450) && (wav <= 495))
      printf("%.1lf nm --> blue\n", wav);

   // Test for yellow light and print appropriate output
   else if ((wav >= 570) && (wav <= 590))
      printf("%.1lf nm --> yellow\n", wav);

   // Test for red light and print appropriate output
   else if ((wav >= 620) && (wav <= 750))
      printf("%.1lf nm --> red\n", wav);

   // Handle all other cases
   else
      printf("%.1lf nm --> unknown\n", wav);
   }
```

3 (continued)

b. Approximate values can be expressed with a margin of error—for example, 50 ± 5%, which implies a range from 47.5 to 52.5, since 2.5 is 5% of 50. This program should read an approximate value and error margin (as a percentage), as well as a value to be tested, and determines if the tested value is within the range of valid approximations. Two sample program runs are shown below, with user input underlined:

```
Enter approx. value, error margin, and test value: 50 5 49
49.00 is within +/- 5.00% of 50.00 ←OUTPUT USES 2 DECIMAL PLACES

Enter approx. value, error margin, and test value: 100 10 89.9
89.90 is not in range              ←OUTPUT USES 2 DECIMAL PLACES
```

Students were responsible for the code shown in bold, underlined font.

```
void main() {
   double app;  // Approximate value (midpoint of valid range)
   double pct;  // Error margin expressed as a percentage
   double err;  // Error margin as an actual value
   double test; // Value to be tested

   // Prompt for/read approx. value, err. margin, and test value
   printf("Enter approx. value, error margin, and test value: ");

   scanf("%lf %lf %lf", &app, &pct, &test);


   // Calculate err (actual value of error margin)
   err = (pct / 100) * app;

   // Check if test is in range and print message if it is
   if ((app - err) <= test && test <= (app + err))
      printf("%.2lf is within +/- %.2lf%% of %.2lf \n",
        test, pct, app);

   // Handle case where value is not in valid range
   else
      printf("%.2lf is not in range\n", test);

}
```

3 (continued)

c. This program should first prompt for and read three integer input values. The program should then test how many valid inputs were entered and do the following steps:

- If all three inputs are valid, print their sum, the product of the first two values, and the first value alone.
- If two inputs are valid, print the product of the first two values and the first value alone.
- If only one input is valid, print that value alone.
- If no inputs are valid, print "`No valid inputs`".

Three sample program runs are shown below, with user input underlined:

```
Enter 3 ints: 2 3 4      Enter 3 ints: 2 4 A      Enter 3 ints: a b c
Sum is 9                 Product is 8             No valid inputs
Product is 6             v1 is 2
v1 is 2
```

Students were responsible for the code shown in bold, underlined font.

```
void main() {
   int v1, v2, v3;        // Integer input values
   int n;                 // Number of valid inputs

   // Prompt for and read three integers
   printf("Enter 3 ints: ");
   n = scanf("%d %d %d", &v1, &v2, &v3);

   // Begin test for number of valid inputs
   switch (n) {

   // 3 valid inputs--print sum (as well as product, v1)
   case 3:
      printf("Sum is %d", v1 + v2 + v3);

   // >= 2 valid inputs--print product (as well as v1)
   case 2:
      printf("Product is %d", v1 * v2);

   // >= 1 valid input--print v1 (and nothing after that)
   case 1:
      printf("v1 is %d", v1);
      break;

   // No valid inputs
   case 0:           // default also accepted, but it's not 100%
                     //   correct, as scanf() can return other
                     //   values (i.e. EOF, which is usually -1)
      printf("No valid inputs\n");           // No valid inputs
}
```