EECE.2160: ECE Application Programming

Fall 2018

Exam 1 Solution

1. (46 points) ***C input/output; operators***

a. (13 points) Show the output of the short program below exactly as it will appear on the screen. Be sure to clearly leave 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.

```
int main() {
    int in1;
    int in2 = 10;
    double d1 = 20.18;
    double d2;

    in1 = d1 + in2 % 4;                in1  = 20.18 + 10 % 4
                                            = 20.18 + 2 = 22.18
                                            = 22 (truncated to int)

    d2 = d1 / 10;                      d2   = 20.18 / 10 = 2.018

    in2 = (in1 + 10) / (in1 - 10);     in2  = (22 + 10) / (22 - 10)
                                            = 32 / 12 = 2.67 = 2
                                              (integer division)

    d1 = in2 + d1;                     d1   = 2 + 20.18 = 22.18

    printf("%d\n%d ", in1, in2);
    printf("%lf\n%.2lf", d1, d2);

    return 0;
}
```

OUTPUT:

```
22
2 22.180000
2.02
```

1 (continued)

b. (13 points) For this program, assume the user inputs the line below. The digit `'1'` 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, exactly as they would be shown on the screen. The program may not read all characters on the input line, but `scanf()` will read something into all seven variables declared in the program. There are no formatting errors in the input!

```
     16.2 2 19 -5
```

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

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

    printf("%d %d\n", int1, int2);
    printf("%.2lf %.2lf\n", d1, d2);
    printf("%c%c%c%c\n", ch1, ch2, ch3, ch4);

    return 0;
}
```

SOLUTION:
This program reads its input as follows:

• *`int1` is the first integer value*	→ *`int1 = 16`*
• *`ch1` is the first character after 16*	→ *`ch1 = '.'`*
• *`d1` holds the next whole number*	→ *`d1 = 2`*
• *`ch2` is the first character after the 2 in "16.2"*	→ *`ch2 = ' '`*
• *`int2` holds the next integer*	→ *`int2 = 2`*
• *`ch3` is the first non-space character after 2*	→ *`ch3 = '1'`*
• *`d2` holds the next whole number*	→ *`d2 = 9`*
• *`ch4` is the first non-space character after `19`*	→ *`ch4 = '-'`*

OUTPUT: *Note that both double-precision values are printed with a precision of 2.*

```
16 2
2.00 9.00
. 1-
```

1 (continued)

c. (20 points) Complete this program so that it prompts the user to enter four input values representing a baseball player's statistics—hits (H), walks (BB), total bases (TB), and at bats (AB), then calculates and prints the following values, each on its own line:

- Batting average (AVG): the fraction of at bats in which the player gets a hit
- On-base percentage (OBP): the fraction of plate appearances (PA = AB + BB) in which the player reaches base by getting either a walk or a hit.
- Slugging percentage (SLG): the ratio of total bases to at bats.
- OPS: the sum of OBP and SLG.

Print each value using 3 decimal places. One example is below, with user input underlined.
(Italic comments next to each line of output won't be printed—they just explain the calculation)

```
Enter H / BB / TB / AB: 100 50 180 450
AVG: 0.222          (AVG = 100 / 450)
OBP: 0.300          (OBP = (100 + 50) / (450 + 50) = 150 / 500)
SLG: 0.400          (SLG = 180 / 450)
OPS: 0.700          (OPS = 0.300 + 0.400)
```

```
int main() {
   double H, BB, TB, AB, PA;   // Stats described above

   // Prompt for and read hits, walks, total bases, at bats
   printf("Enter H / BB / TB / AB: ");

   scanf("%lf %lf %lf %lf", &H, &BB, &TB, &AB);

   // Calculate any necessary intermediate values
   PA = AB + BB;

   // Print AVG, OBP, SLG, and OPS
   printf("AVG: %.3lf\n", H / AB);
   printf("OBP: %.3lf\n", (H + BB) / PA);
   printf("SLG: %.3lf\n", TB / AB);
   printf("OPS: %.3lf\n", (H + BB) / PA + TB / AB);

   return 0;
}
```

2. (34 points) ***Conditional statements***

a. (14 points) For the short program shown below, the first line of output (the prompt `"Enter v1 and v2: "`) and the user input (`4 1`) is listed at the bottom of the page.

Complete the rest of the output for this program, given those input values.

```
int main() {
   int v1, v2;
   printf("Enter v1 and v2: ");
   scanf("%d %d", &v1, &v2);
```

v1 = 4, v2 = 1

```
   if (v1 > v2)
      printf("%d > %d\n", v1, v2);
```

Condition true (4 > 1), so program prints: 4 > 1

```
   if (v2 > 0)
      printf("%d positive\n", v2);
   else if (v1 > 0)
      printf("%d positive\n", v1);
   else
      printf("Both negative\n");
```

First condition true (1 > 0), so program prints: 1 positive
Else if and else cases ignored

```
   switch (v1) {
      case 2: case 4:
         printf("%d %d\n", v1, v2);
      case 1: case 3:
         printf("%d %d\n", v2, v1);
         break;
      default:
         printf("None of the above\n");
   }

   return 0;
}
```

Since v1 == 4, program prints: 4 1
Since there is no break after the 1st printf(), program also prints: 1 4

OUTPUT (the first line is given; write the remaining line(s)):

```
Enter v1 and v2: 4 1
4 > 1
1 positive
4 1
1 4
```

2 (continued)

b. (20 points) Complete this program, which prompts for and reads one non-space character, then tests that character to see if it is a letter, number, or neither. In each case, the program reprints the character with an appropriate description, as shown in the tests below (user input is underlined).

```
Enter character: E       Enter character: 6       Enter character: !
E is a letter            6 is a number            ! is neither
```

Hints: You can compare a character variable to a character constant. For example, the condition `(ch < '?')` is true for all characters with ASCII values less than `'?'`.

Your program should check for the valid range(s) for each type of character (uppercase and lowercase letters are separate). ASCII values are *usually* higher later in the alphabet—`'D'` is greater than `'C'`, but it's not greater than `'c'`. ASCII values increase as expected for numeric characters—`'3'` is greater than `'2'`.

You do not need to know any actual ASCII values to solve this problem!

```
int main() {
    char ch;     // Input character

    // Prompt for/read input character
    printf("Enter character: ");
    scanf("%c", &ch);

    // Test for letter and print appropriate statement
    if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))
        printf("%c is a letter\n", ch);

    // Test for number and print appropriate statement
    else if (ch >= '0' && ch <= '9')
        printf("%c is a number\n", ch);

    // Handle all other cases
    else
        printf("%c is neither\n", ch);
}
```

3. (20 points, 5 points each) <u>***While and do-while loops***</u>

a. What is the output of the short code sequence below? **<u>Choose only one answer.</u>**

```
x = 1;
while ((x % 3) != 0) {
   printf("%d ", x);
   x = x + 4;
}
```

i. No output—the loop condition is initially false

ii. `1`

iii. <u>***1 5***</u>

iv. `1 5 9`

v. `1 5 9 13`

b. 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 <u>end</u>? In other words, which value(s) will cause the loop condition to be false? **<u>This question has at least one correct answer, but may have more than one correct answer! Circle ALL choices that correctly answer the question.</u>**

i. <u>***0 0***</u>

ii. 1 1

iii. -1 -1

iv. -3 5

v. <u>***5 -3***</u>

3 (continued)

c. Which loops below produce the following output?

```
* * *
```

This question has at least one correct answer, but may have more than one correct answer! Circle ALL choices that correctly answer the question.

i.

```
int a = 3;
while (a) {
   printf("* ");
   a = a - 1;
}
```

ii.

```
int b = 0;
do {
   b = b + 1;
   printf("* ");
} while (b <= 3);
```

iii.

```
int c = 10;
while (c > 0) {
   c = c / 3;
   printf("* ");
}
```

iv.

```
int d = 1;
do {
   printf("* ");
   d = d * 2;
} while (d – 6 < 0);
```

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 hope the next exam is as easy as this question.”

All choices are “correct.”

4. (10 points) ***EXTRA CREDIT***

REMEMBER, YOU CANNOT GET EXTRA CREDIT WITHOUT WRITING AT LEAST PARTIAL SOLUTIONS FOR ALL OTHER PROBLEMS ON THE EXAM.

Complete the program below, which calculates and prints all Fibonacci numbers that are strictly less than the input value `limit`. The first two Fibonacci numbers are 0 and 1; each subsequent value in the sequence is the sum of the two numbers preceding it: ***0, 1, 1, 2, 3, 5, 8, 13, 21, 34 …***

If `limit` is so low that it prevents you from printing at least the first three values in the sequence, print an error message. Examples are shown below:

- If `limit == 5`, print: `0 1 1 2 3`
- If `limit == 40`, print: `0 1 1 2 3 5 8 13 21 34`
- If `limit == -1`, print: `Error: Limit is too low`

Solution:

```
int main() {
     int limit;           // Upper limit
     int num1, num2;      // Values used to calculate sequence

     // Prompt for and read limit
     printf("Enter limit: ");
     scanf("%d", &limit);

     // COMPLETE PROGRAM AS DESCRIBED ABOVE
     // Exit if limit is too low to print at least 3 values
     if (limit < 2) {
          printf("Error: Limit is too low\n");
          return 0;
     }

     // Initialize num1 and num2 and print first 2 values
     num1 = 0;
     num2 = 1;
     printf("0 1 ");

     // Loop that keeps calculating Fibonacci numbers as long as
     //   next value is less than limit
     while (num1 + num2 < limit) {
          printf("%d ", num1 + num2);       // Print next value

          // Update num1 and num2—SEE NEXT PAGE FOR EXPLANATION
          num2 = num1 + num2;      // num2 = most recent value
          num1 = num2 - num1;      // num1 = old value of num2
     }

     return 0;
}
```

4 (continued)

To see why the equation for `num1` works, let `num1′` and `num2′` be the new values of `num1` and `num2` calculated in each loop iteration. Once we overwrite `num2`, to get its old value, we have to work with the equation for the new value.

Since `num2′ = num1 + num2`, it follows that `num2 = num2′ - num1 = num1′`