16.216: ECE Application Programming

Fall 2011

Exam 1 Solution

1. (24 points, 6 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. Which of the following statements correctly defines `ClassNum` as a constant value?

i. `#constant ClassNum 16.216`

ii. `#define ClassNum 16.216`

iii. `#define ClassNum = 16.216`

iv. `double ClassNum = 16.216;`

b. Which of the statements below has the same result as `y = x * 2`?

i. `y = x >> 1;`

ii. `y = x << 1;`

iii. `y = x >> 2;`

iv. `y = x << 2;`

v. None of the above

1 (cont.)

c. Given an integer `val`, which of the following `if` statements prints `val` only if `val` is a positive, non-zero value that is less than 100?

i.
```
if ((val != 0) || (val < 100))
   printf("Value = %d\n", val);
```

ii.
```
if ((val != 0) && (val < 100))
   printf("Value = %d\n", val);
```

iii.
```
if ((val > 0) || (val < 100))
   printf("Value = %d\n", val);
```

iv.
```
if ((val > 0) && (val < 100))
   printf("Value = %d\n", val);
```

v. None of the above

d. Say you wish to print a double-precision variable, `d`, using precision `prec` and field width `width`. Which of the following statements correctly implements this operation?

i.
```
printf("%*lf", prec, d);
```

ii.
```
printf("%*lf", width, d);
```

iii.
```
printf("%10.5lf", d);
```

iv.
```
printf("%*.*lf", prec, width, d);
```

v.
```
printf("%*.*lf", width, prec, d);
```

2. (37 points) ***Reading code***

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() {
     int x, y;
     double d, e, f;

     x = 15;
     y = x + 5 / 4 - x;        // y = x + 5 / 4 – x =
                               //   = x + 1 – x = 1
     -x;                       // Has no effect

     d = 2;                    // Even though 2 is an int, d
                               //   still has type double
     e = 10 / d;               // e = 5.0 (double)
     f = (x - 10.0) / 10;      // f = (15 – 10.0) / 10
                               //   = (5.0) / 10 = 0.5

     printf("x = %d, y = %d\n", x, y);
     printf("d = %lf,\ne = %lf,\nf = %lf\n", d, e, f);

     return 0;
}
```

Solution: *See above for how each value is calculated*

OUTPUT:

```
x = 15, y = 1
d = 2.000000,
e = 5.000000,
f = 0.500000
```

2 (cont.)

b. (12 points)

```
int main() {
     float q, r, s;
     int p = 3, w = 9;

     q = 16.216;
     r = (q - 13.216) / 4 + 1.0;  // r = (16.216–13.216)/4 + 1.0
                                  //   = 3.0 / 4 + 1.0
                                  //   = 0.75 + 1.0 = 1.75
     s = w / p * 10.0 + 0.6789;   // s = 9 / 3 * 10.0 + 0.6789
                                  //   = 3 * 10.0 + 0.6789
                                  //   = 30.0 + 0.6789 = 30.6789

     printf("q:%-10f", q);
     printf("r:%+*.0f", p, r);
     printf("s:%*.*f", w, p, s);

     return 0;
}
```

Solution: *See above for how each value is calculated. As for the output formatting, note that:*

- *All output is on the same line, as no* `'\n'` *characters are listed in any of the* `printf()` *format strings.*
- `q` *is printed using the default precision of 6, since no precision is specified, and will therefore be printed as* `16.216000`*. This value will be left justified in a field of 10 characters, which means that one space will follow the nine characters used in the value.*
- `r` *is printed using a field width of 3 (since* `p == 3`*) and a precision of 0, which will force the value 1.75 to be rounded to 2. The* `+` *flag will force the* `'+'` *character to be shown before the value. This value will be right justified in the field, leaving a single space to its left.*
    - *There is no space to the right of this value, which means that the next* `printf()` *statement will begin printing its output immediately after the digit* `2`*.*
- `s` *is printed using a field width of 9 (*`w == 9`*) and a precision of 3 (*`p == 3`*). The last digit will be rounded up, giving us an output of* `30.679`*. Since this value contains six characters, three spaces are to the left of the right justified value (in between the* `':'` *and the digit* `3`*).*

OUTPUT:

```
q:16.216000 r: +2s:   30.679
```

c. (12 points)

```
int main() {
     int x;
     printf("Enter value: ");
     scanf("%d", &x);
     switch (x - 2) {
     case 8:
     case 9:
          if ((x % 2) == 1)
               printf("Isn't that odd?\n");
          else
               printf("Let's make this even.\n");
          break;
     case -1:
          printf("That's less than I thought.\n");
     default:
          printf("How did we get here?\n");
     }
     return 0;
}
```

Complete the output for each of the following three possible inputs (input values underlined):

i. `Enter value: 11`

Solution: *If `x == 11`, then `x-2 == 9`, and we go to the "`case 9:`" label. We then evaluate the condition: `((11 % 2) == 1)`, so the condition is true, and the output is:*

```
Isn't that odd?
```

ii. `Enter value: 8`

Solution: *If `x == 8`, then `x-2 == 6`, which matches none of the cases. We therefore go to the `default` case and print:*

```
How did we get here?
```

iii. `Enter value: 1`

Solution: *If `x == 1`, then `x-2 == -1`, and we go to the "`case -1:`" label. Note that there is no `break` statement at the end of the case, so the program will print the following two lines:*

```
That's less than I thought.
How did we get here?
```

3. (39 points, 13 per part) ***Writing code***

For each part of this problem, you are given a short program to complete. **CHOOSE ANY THREE OF THE FOUR PARTS** and fill in the space provided with appropriate code. **If you complete all four, I will grade only the first three.**

a. Complete the program below so that it prompts for and reads three hexadecimal values, then prints each value in original form and with one additional change:

- First value: highest (bit 31) and lowest (bit 0) bits cleared (equal to 0)
- Second value: bit 16 flipped.
- Third value: middle eight bits (bits 12-19) set to 1

Outputs should contain 8 digits and a leading 0x. Leading zeroes should be added if necessary.

```
int main() {
     unsigned int hex1, hex2, hex3;     // Hexadecimal inputs

     // Prompt for and read inputs
     printf("Hex values: ");
     scanf("%x %x %x", &hex1, &hex2, &hex3);

     // Print first value with bits 0 and 31 cleared
     printf("%#010x %#010x\n", hex1, hex1 & 0x7FFFFFFE);

     // Print second value with bit 16 flipped
     printf("%#010x %#010x\n", hex2, hex2 ^ 0x00010000);

     // Print third value with bits 12-19 set to 1
     printf("%#010x %#010x\n", hex3, hex3 | 0x000FF000);

     return 0;
}
```

3 (cont.)

b. Complete the program below so that it prompts the user to enter the inputs shown, reads those values, and computes a weighted average, using the following variables:
   - `input1`, `input2`: The values to be averaged.
   - `percent1`, `percent2`: The weight for each value, written as a percentage.
     - `percent1 + percent2` should always equal 100.

The average should always be printed using three places after the decimal point.

```
int main() {
     double input1, input2;        // Values to be averaged
     int percent1, percent2;       // Weight for each value

     // Prompt for and read inputs
     printf("Inputs: ");
     scanf("%lf %lf", &input1, &input2);
     printf("Weights: ");
     scanf("%d %d", &percent1, &percent2);

     // Calculate and print average
     printf("Average: %.3lf\n",
           ((input1 * percent1) + (input2 * percent2)) / 100.0);

     return 0;
}
```

3 (cont.)

c. Complete the program below so that it first prompts for and reads:
   - The upper and lower bounds of a range, written as `lower -> upper`
   - Two additional input values separated by a comma: `input1, input2`

The program should then print the following output:

- If `input1` is outside of the range `lower-upper`, print "`Input 1 out of range`".
- If the previous condition is false, but `input2` is inside the range `lower-upper` (including those end points), print "`Both inputs in range`".
- If neither of the previous conditions is true, print "`Conditions false`".

```
int main() {
     double lower, upper;          // Lower and upper bounds
     double input1, input2;        // Values to be checked

     // Prompt for and read inputs
     // Be sure to use proper formatting in scanf() strings,
     //    as values are separated by specific characters
     printf("Range: ");
     scanf("%lf -> %lf", &lower, &upper);
     printf("Inputs: ");
     scanf("%lf, %lf", &input1, &input2);

     // First, check if input1 is out of range
     if ((input1 < lower) || (input1 > upper))
          printf("Input 1 out of range\n");

     // If not, check if input2 is in range
     else if ((input2 >= lower) && (input2 <= upper))
          printf("Both inputs in range\n");

     // If both conditions false, print appropriate message
     else
          printf("Conditions false\n");

     return 0;
}
```

3 (cont.)

d. Complete the program below so that it prompts for and reads a single, case-insensitive character (i.e., treat `'A'` and `'a'` the same), followed by two integers. Depending on the character entered, your program should print the following information about these numbers:

- `'A'`, `'a'` → print the average of the integers in the form: `Average = <result>`
  - Always show one digit after the decimal point for the average.
- `'B'`, `'b'`, `'L'`, `'l'` → print the largest (most positive) of the integers in the form: `Largest = <result>`
- Any other character → print "`Invalid command`"

```
int main() {
     char cmd;           // Case-insensitive command input
     int num1, num2;     // Two numeric inputs

     // Prompt for and read inputs
     printf("Command: ");
     scanf("%c", &cmd);
     printf("Numbers: ");
     scanf("%d %d", &num1, &num2);

     // Check command and perform appropriate action
     switch (cmd) {
     case 'A':      // Average
     case 'a':
          printf("Average: %.1lf\n", (num1 + num2) / 2.0);
          break;

     case 'B':      // Largest value
     case 'b':
     case 'L':
     case 'l':
          if (num1 > num2)
               printf("Largest: %d\n", num1);
          else
               printf("Largest: %d\n", num2);
          break;

     default:
          printf("Invalid command\n");

     return 0;
}
```