16.216: ECE Application Programming

Fall 2015

Exam 2
November 4, 2015

Name: _____ **ID #:** _____

For this exam, you may use only one 8.5" x 11" double-sided page of notes. All electronic devices (e.g., calculators, cellular phones, PDAs) are prohibited. If you have a cellular phone, please turn it off prior to the start of the exam to avoid distracting other students.

The exam contains 3 questions for a total of 100 points. Please answer the questions in the spaces provided. If you need additional space, use the back of the page on which the question is written and clearly indicate that you have done so.

Please read each question carefully before you answer. In particular, note that:

- Question 3 has three parts, but you are only required to complete two of the three parts.
  - You may complete all three parts for up to 10 points of extra credit. If you do so, **please clearly indicate which part is the extra one—I will assume it is part (c) if you mark none of them.**
- For each part of Question 3, you must complete a short function. I have provided comments to describe what your function should do and written some of the code for you.
  - Note that each function contains both lines that are partially written (for example, a `printf()` call in which you are responsible for filling in the format string and expressions) and blank spaces in which you must write additional code. **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.**
  - Each function is accompanied by one or more test cases. Each test case is an example of how the function should behave in one specific case—**it does not cover all possible results of using that function.**
- You can solve each part of Question 3 using only the variables that have been declared, but you may declare and use other variables if you want.

You will have 50 minutes to complete this exam.

Q1: Multiple choice	/ 20
Q2: Arrays	/ 40
Q3: Functions	/ 40
TOTAL SCORE	/ 100
EXTRA CREDIT	/ 10

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;
char s1[20];
char s2[20];
strcpy(s1, "xo");
strcpy(s2, "xx");
for (i = 0; i < 4; i++) {
   strcat(s1, s2);
   if (i > 1)
      s2[0] = 'o';
}
printf("%s\n", s1);
```

i. `xo`

ii. `xoxx`

iii. `xoox`

iv. `xoxxxxxxxx`

v. `xoxxxxxxox`

1 (continued)

b. Which of the following code snippets will attempt to open a file for output and redirect output to the screen if the file cannot be opened?

i.
```
FILE *fp = fopen("outfile.txt", "r");
if (fp == NULL)
   fp = stdin;
```

ii.
```
FILE *fp = fopen("outfile.txt", "w");
if (fp == NULL)
   fp = stdin;
```

iii.
```
FILE *fp = fopen("outfile.txt", "r");
if (fp == NULL)
   fp = stdout;
```

iv.
```
FILE *fp = fopen("outfile.txt", "w");
if (fp == NULL)
   fp = stdout;
```

v.
```
FILE *fp = fopen("outfile.txt", "w");
if (fp != NULL)
   fp = stdout;
```

1 (continued)

c. Assuming the file pointer `fp` accesses a valid input file, which of the following code snippets will read characters from that file, one at a time, until a letter is found? (In other words, the last character read into the variable `ch` will be a letter.)

i.
```
ch = fgetc(fp);
```

ii.
```
do {
   ch = fgetc(fp);
} while (isalpha(ch));
```

iii.
```
do {
   ch = fgetc(fp);
} while (!isalpha(ch));
```

iv.
```
do {
   ch = fgetc(fp);
} while (ch != EOF);
```

v.
```
fgets(&ch, 1, fp);
```

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?"

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. (14 points)

```
int main() {
   int i;
   double arr[] = {1.23, 2.34, 3.45, 4.56, 5.67, 6.78};
   int ind[7] = {5, 1, 3, 2, 0, 4};

   for (i = 5; i > 0; i -= 2)
      printf("%.2lf %.2lf\n", arr[i], arr[i-1]);

   for (i = 0; i < 7; i++)
      printf("%d %.2lf\n", ind[i], arr[ind[i]]);

   return 0;
}
```

2 (continued)

b. (12 points)

```
int main() {
   int mat[4][2] = { {9, -5}, {0, 3}, {4, -2}, {-1, -8} };
   int i, j;

   j = 0;
   for (i = 0; i < 4; i++) {
      j = j + mat[i][1];
      if (j < 0)
         j = -j;
      if (j > 7)
         j = j % 8;
      printf("%d\n", mat[j%4][j%2]);
   }

   return 0;
}
```

2 (continued)

c. (14 points)

```
void f(int arr[], int n, int gap, int inc) {
   int i, tmp;

   for (i = 0; i < n - gap; i += inc) {
      tmp = arr[i];
      arr[i] = arr[i+gap];
      arr[i+gap] = tmp;
   }
}

int main() {
   int list[10] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};
   int i;

   f(list, 10, 5, 2);
   for (i = 0; i < 10; i++)
      printf("%d ", list[i]);
   printf("\n");

   f(list, 6, 1, 1);
   for (i = 0; i < 10; i++)
      printf("%d ", list[i]);
   printf("\n");

   return 0;
}
```

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. `int findReverse(int n);`

This function returns a value that contains the digits of its input argument, `n`, in reverse. For example, `findReverse(456) = 654` and `findReverse(2015) = 5102`. The general algorithm requires you to isolate each digit, add it to a running total, then remove that digit from the remaining total. For example, the steps required for reversing 456 would be:

- Current digit = 6 → Running total = 6, remaining total = 45
- Current digit = 5 → Running total = 65, remaining total = 4
- Current digit = 4 → Running total = 654, remaining total = 0

```
int findReverse(int n){
   int dig;          // Current digit
   int res;          // Running total and ultimately final result

   // Initialize variables as needed




   // Loop until there are no more digits to examine

   _________ (_____________________________________________________) {

      // Isolate the lowest remaining digit, then combine it with
      //   the digits already tested (running total)









   }
}
```

3 (continued)

b. `void rgb2cmyk(int R, int G, int B, double *C, double *M, double *Y, double *K);`

This function converts from one color encoding scheme (RGB) to another (CMYK). The RGB arguments are passed by value; each RGB value is between 0-255. The CMYK results are arguments passed by address; each CMYK value is between 0-1. The algorithm is as follows:

- Normalize each RGB value (convert it to the range 0 to 1) by dividing by the maximum possible RGB value.
- Find the maximum (`max`) of these normalized values.
- The black (K) value is simply 1 – max.
- The other values are based on black and one other color; each can be found using the formula: *(1 - <normalized color> - K) / (1 – K)*. Cyan (C) is based on red and black, magenta (M) is based on green and black, and yellow (Y) is based on blue and black.

For example, if red (R) = 51, green (G) = 102, and blue (B) = 153, their normalized values are 0.2, 0.4, and 0.6, respectively. Therefore, K = 1 – 0.6 = 0.4, C = (1 – 0.2 – 0.4) / (1 – 0.4) = 0.67, M = (1 – 0.4 – 0.4) / (1 – 0.4) = 0.33, and Y = (1 – 0.6 – 0.4) / (1 – 0.4) = 0.

```
void rgb2cmyk(int R, int G, int B, double *C, double *M,
              double *Y, double *K) {
   double rN, gN, bN;          // Normalized versions of r, g, b
   double max;                 // Max of normalized RGB values

   // Normalize RGB values




   // Find max of normalized RGB values




   // Calculate CMYK values




}
```

3 (continued)

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

This function tests the array `arr[]` to see if its first n elements are sorted and returns:

- A negative value if `arr[]` is sorted from lowest to highest value
- A positive value if `arr[]` is sorted from highest to lowest value
- 0 if `arr[]` is not sorted

The general algorithm is as follows:

- Check the difference between consecutive elements until you find two that don't match.
- Then, check if each pair of consecutive elements is in the same order (lo → hi or hi → lo) as the first different pair. If you find a pair that's out of order, return 0.
- If the whole array is sorted, return a positive or negative value as described above

```
int isSorted(int arr[], int n) {
   int ord;    // Tracks whether array is sorted
   int i;      // Loop index

   // Set ord to difference between first two array elements
   // Should be negative if first element has lower value



   // Go through remaining elements; return 0 if order is wrong

   for (_____________________________________________________) {

      // If all previous elements match, recalculate ord

      if (_____________________________________________________)




      // If array is found to be out of order, return 0
      // (condition may require two lines to write)

      else if (_____________________________________________

               _____________________________________________) {
         return 0;
   }

   // Array is sorted--return appropriate value

   return ___________________________________________;
}
```