16.216: ECE Application Programming

Spring 2014

Exam 2

April 2, 2014

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) ***<u>Multiple choice</u>***

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. Which values listed below for the strings `s1` and `s2` will cause the function `strncmp(s1, s2, 4)` to return the value 0?

   A. `s1 = "This", s2 = "That"`

   B. `s1 = "This", s2 = "This"`

   C. `s1 = "Exam 1", s2 = "Exam 2"`

   D. `s1 = "test", s2 = "TEST"`

i. Only A

ii. Only B

iii. B and C

iv. A and D

v. B, C, and D

b. Given the short code sequence below:

```
int i;
char str[20] = "baa";
for (i = 0; i < 3; i++)
  strcat(str, "ab");
printf("Length of string = %d\n", strlen(str));
```

What will this program print?

i. `Length of string = 2`

ii. `Length of string = 3`

iii. `Length of string = 5`

iv. `Length of string = 9`

v. `Length of string = 20`

1 (cont.)

c. Given the code sequence below:

```
char s1[20];
char s2[20];
strcpy(s1, "String 1");
strncpy(s2, s1, 5);
s2[5] = '\0';
s2[1] = 'p';
printf("%s  %s\n", s1, s2);
```

What will this program print?

i. `String 1  String 1`

ii. `String 1  Strin`

iii. `Spring 1  String 1`

iv. `Spring 1  Strin`

v. `String 1  Sprin`

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. (12 points)

```
void main() {
  double arr[6] = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6};
  int i;

  for (i = 5; i >= 0; i--)
    printf("%.1lf ", arr[i]);     // Prints each value using
                                  //  precision of one
  printf("\n");

  for (i = 0; i < 6; i += 2) {
    arr[i] = arr[i] + arr[i+1];
    printf("%.1lf ", arr[i]);
  }
  printf("\n");
}
```

2 (continued)

b. (14 points)

```
void main() {
  int tab[3][2] = { {0,  5},
                    {10, 15},
                    {20, 25} };
  int i, j;

  for (j = 0; j < 2; j++) {
    for (i = 0; i < 3; i++)
      printf("%d ", tab[i][j]);
    printf("\n");
  }

  for (i = 0; i < 6; i++)
    printf("%d ", tab[i/2][i/3]);
  printf("\n");
}
```

2 (continued)

c. (14 points)

```
void main() {
   int i, r, c;
   int rn[7] = {2, 0, 1, 3, 2, 1, 2};
   int cn[7] = {3, 1, 2, 3, 0, 1, 1};

   int arr[4][4] = { {1, 3, 5, 7},
                     {10, 8, 6, 4},
                     {-1, -2, -4, -8},
                     {30, 31, 20, 14} };

   for (i = 0; i < 7; i++) {
      r = rn[i];
      c = cn[i];
      printf("%d\n", arr[r][c]);
   }
}
```

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 minMax(int arr[], int n, int *min, int *max);`

This function reads in an array, `arr[]`, that contains `n` elements, as well as two pointers, `min` and `max`, to variables outside the function. When the function is complete, the variables pointed to by `min` and `max` should contain the smallest and largest values in the array.

For example, if your main function contains an integer array `x = {1, -3, 5, 7, 12}`, and integer variables `a` and `b`, calling `minMax(x, 5, &a, &b)` will set `a` to `-3` and `b` to `12`.

```
void minMax(int arr[], int n, int *min, int *max) {
  int i;       // Loop index

  // Initialize min/max values







  // Complete this loop to visit all array elements and
  //   update minimum/maximum values as needed

  for (_________________; _________________; ________________) {








  }
}
```

3 (continued)

b. `int isPrime(unsigned int num);`

This function reads in a single unsigned (i.e., non-negative) integer. The function returns 1 if the number is prime and 0 otherwise. A prime number is an integer greater than 2 that is divisible only by 1 and itself. To test for a prime number in this function, do the following:

- Return 0 if the number is even or less than 3, since those values cannot be prime.
- For all other values, test to see if the number is divisible by all odd values from 3 up to the square root of `num`. Note that:
  - You may use the `sqrt()` function from `<math.h>` to find the square root of any number (for example, `sqrt(81)` returns 9). Assume `<math.h>` is included.
  - Recall that, if x is divisible by y, the remainder of x / y is 0.

```
int isPrime(unsigned int num) {
   int div;          // Divisor to be tested

   // Simple test for non-prime values: all even numbers and
   //   values less than 3 are not prime--return 0 in these cases

   if (_________________________________________________________)
      return 0;


   // Otherwise, go through all possible odd divisors between 3
   //   and the square root of num.

   for (_________________; __________________; ________________) {


      // If num is divisible by divisor, it's not prime--return 0



           return 0;


   }



   // If you check all possible divisors and don't find one that
   //   works, number must be prime--return 1
   return 1;
}
```

3 (continued)

c. `void countOccurrences(int occ[], int n);`

- Given: an array, `occ[]`, and the array length, `n`.
  - Note: You must correctly initialize `occ[]` before using it.
- Repeatedly prompt for and read integers from standard input and count how many times each value is entered, using `occ[]`—`occ[0]` is the number of times 0 is entered, `occ[1]` is the number of times 1 is entered, and so on.
- Exit the function when the input value is not a valid array index for `occ[]`.
- Example: assume n = 4, and user enters: `0 0 0 1 1 2 2 2 2 2 3 -1`
  - `occ[0] = 3, occ[1] = 2, occ[2] = 5, occ[3] = 1`
  - Prompts (`"Enter integer value: "`) not shown in example to save space

```
void countOccurrences(int occ[], int n) {
     int inval;     // Input value
     int i;         // Loop index

     // Initialize array




     // Loop infinitely (return statement exits function)



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

          // If input value is invalid, exit function
          if (_________________________________________________)

               return;



          // If input value is valid, use it as an array index
          //   and increment the appropriate element of occ[]




}
```