

16.216: ECE Application Programming

Spring 2013

Exam 2
March 27, 2013

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 that problem, you must complete a short program. I have provided comments to describe what your program should do, as well as written some of the program 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 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 choice you think best answers the question.

a. What will the following short program print?

```
void f(int arr[], int n) {
    int i;
    for (i = 0; i < n; i++)
        arr[i] += 5;
}

void main() {
    int x[6] = {1, 2, 3, 4, 5, 6};

    f(x, 3);
    printf("%d %d %d %d %d %d\n",
           x[0], x[1], x[2], x[3], x[4], x[5]);
}
```

- i. 1 2 3 4 5 6
- ii. 4 5 6 4 5 6
- iii. 4 5 6 7 8 9
- iv. 6 7 8 4 5 6
- v. 6 7 8 9 10 11

b. In a given program, assume there is a two-dimensional array: `int arr[5][10];`

Which of the following choices shows a valid prototype for a function that could use `arr` as an argument?

- i. `void f1(int x);`
- ii. `int f2(int x[]);`
- iii. `double f3(int x[10]);`
- iv. `void f4(int x[5][]);`
- v. `void f5(int x[][10]);`

1 (continued)

c. Given the following variables:

```
int z[5] = {0, 1, 2, 3, 4};  
int *p = z;
```

Which of the following statements will change the third value in the array from 2 to 12?

- A. `z[3] = 12;`
- B. `p[2] = 12;`
- C. `*p = *p + 10;`
- D. `z[2] = z[3] * z[4];`

- i. Only A
- ii. A and D
- iii. B and C
- iv. B and D
- v. C and D

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. "I really could have used another snow day today."
- v. "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)

```
void main() {
    int i;
    int list[8] = {3, 5, -2, -4};

    for (i = 0; i < 8; i++)
        printf("%d ", list[i]);
    printf("\n");

    for (i = 0; i < 8; i += 2) {
        list[i] = list[i] + i;
        printf("%d ", list[i]);
    }
    printf("\n");

    for (i = 7; i > 0; i /= 2) {
        list[i]++;
        printf("%d ", list[i]);
    }
    printf("\n");
}
```

2 (continued)

b. (12 points)

```
void main() {
    int i, j;
    int tab[2][5];
    int v = 0;

    for (i = 0; i < 2; i++) {
        for (j = 0; j < 5; j++) {
            tab[i][j] = v++;
            printf("%d ", tab[i][j]);
        }
        printf("\n");
    }

    for (j = 4; j >= 0; j--) {
        for (i = 1; i >= 0; i--) {
            printf("%d ", tab[i][j]);
        }
        printf("%\n");
    }
}
```

2 (continued)

c. (14 points)

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

    int arr[4][4] = { {0, 2, 4, 6},
                      {3, 5, 7, 11},
                      {-1, -2, -4, -8},
                      {13, 21, 34, 55} };

    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 function 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. `double approx(double x, int n);`

This function should calculate the following series approximation for the value $1 / (1-x)$, which is valid if $|x| < 1$:

$$\frac{1}{1-x} \approx \sum_{k=0}^n x^k = 1 + x + x^2 + x^3 + \dots + x^n$$

The function takes two arguments—the values of x and n , as shown above—and should return the approximate value calculated. For example, if $x = 0.5$ and $n = 3$, the function should return:

$$1 + 0.5 + 0.5^2 + 0.5^3 = 1 + 0.5 + 0.25 + 0.125 = 1.875$$

```
double approx(double x, int n) {
    double total;           // Running total for approximation
    double x_i;             // x to the power of i
    int i;                  // Loop index

    // Initialize variables as needed

    // Calculate series approximation as described above

    // Return result
}
```

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

    return 0;

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

    // 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 minMaxAvg(double x[], int n, double *min,  
                  double *max, double *avg)
```

This function takes an array, `x`, that contains `n` values, and finds the minimum and maximum values in the array, while also computing the average.

Note that:

- `min`, `max`, and `avg` hold the addresses of variables outside this function that will hold the minimum, maximum, and average values when the function is done.
- You must use those pointers to initialize the variables that `min`, `max`, and `avg` point to before going through the array.

```
void minMaxAvg(double x[], int n, double *min,  
               double *max, double *avg) {  
  
    int i; // Loop index  
  
    // Initialize variables as needed  
  
  
  
    // Go through entire array, and, for each element:  
    // 1. Ensure the value is included in the average  
    // 2. If the value is less than the current minimum,  
    //     overwrite the current minimum  
    // 3. If the value is more than the current maximum,  
    //     overwrite the current maximum  
  
  
  
  
  
  
  
  
  
    // Perform any additional calculations needed outside the loop  
    // that goes through the entire array  
  
}
```