16.216: ECE Application Programming

Spring 2015

Exam 2 Solution

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. How many iterations does the `for` loop below execute?

```
for (i = 40; i >= 0; i -= 5)
   printf("%d ", i);
```

Values of i: 40, 35, 30, 25, 20, 15, 10, 5, 0, -5 (i changes to -5 at the end of the last loop iteration, making the loop condition false and preventing a tenth iteration)

i. 1

ii. 5

iii. 8

iv. 9

v. 40

b. Which of the following declarations creates an array of 5 integers?

a. `int arr[] = {1, 3, 5, 7, 9};` **Contains the 5 integers given**

b. `int arr[] = {5};` **Contains only one value, 5**

c. `int arr[5] = {1, 3, 5};` **Contains {1, 3, 5, 0, 0}**

d. `int arr[5] = {0};` **Contains {0, 0, 0, 0, 0}**

i. Only A

ii. Only B

iii. A and C

iv. A, C, and D

v. B, C, and D

1 (continued)

c. Which of the following choices accurately describes the contents of a two-dimensional array declared using the following statement:

```
int list[5][10];
```

i. The array `list` contains a total of 15 integers.

ii. The array `list` contains a total of 50 integers, organized as 10 rows and 5 columns.

iii. The array `list` contains a total of 50 integers, organized as 5 rows and 10 columns.

iv. The array `list` contains two integers—5 and 10.

v. None of the above.

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?"

All of the above are "correct"

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)

```
int main() {
   int i;
   int arr[8] = {1, 29, 65, 3, 15, 9};

   for (i = 0; i < 8; i += 2)
      printf("%d %d\n", arr[i], arr[i+1]);


   for (i = 7; i > 0; i /= 2)
      printf("%d\n", arr[i] + arr[i-1]);

   return 0;
}
```

Prints elements 0 & 1, then 2 & 3, 4 & 5, and 6 & 7

i starts at 7, then goes to:
7/2 = 3
3/2 = 1
1/2 = 0 (loop ends)
So, this loop prints the sum of elements 7 & 6, 3 & 2, and 1 & 0

OUTPUT:

1 29

65 3

15 9

0 0

0

68

30

2 (continued)

b. (14 points)

```
int main() {
   int i, j;
   int tab[2][5];

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

   for (i = 0; i < 10; i++)
      printf("%d ", tab[i%2][i/2]);

   return 0;
}
```

Assigns value to each element and prints array column by column, not row by row.

Array contents (by row):

0 2 4 6 8

1 3 5 7 9

Row/col numbers as follows:

i = 0 → tab[0][0]

i = 1 → tab[1][0]

i = 2 → tab[0][1]

i = 3 → tab[1][1]

i = 4 → tab[0][2]

i = 5 → tab[1][2]

i = 6 → tab[0][3]

i = 7 → tab[1][3]

i = 8 → tab[0][4]

i = 9 → tab[1][4]

OUTPUT:

```
0 1
2 3
4 5
6 7
8 9
0 1 2 3 4 5 6 7 8 9
```

2 (continued)

c. (14 points) *The original exam had an extremely unfortunate error: I didn't include the definition of the function `f()`, which made the problem impossible to complete.*

```
void f(double arr[], int n1, int n2) {
   int i;

   for (i = n1; i < n2; i++)
     arr[n2 - 1 - i] = arr[i];
}
```

Copies values from one array position to another—see below. Note that function may copy value it wrote in earlier iteration, as shown in examples

```
int main() {
   double list[] = {1.23, 4.56, 7.89, 10.11, 12.13, 13.14};
   int i;

   f(list, 0, 4);
   for (i = 0; i < 6; i++)
      printf("%.2lf ", list[i]);
   printf("\n");
```

Loop in f() does following:
arr[3] = arr[0] = 1.23
arr[2] = arr[1] = 4.56
arr[1] = arr[2] = 4.56
arr[0] = arr[3] = 1.23

```
   f(list, 1, 3);
   for (i = 0; i < 6; i++)
      printf("%.2lf ", list[i]);
   printf("\n");
```

Loop in f() does following:
arr[1] = arr[1] = 4.56
arr[0] = arr[2] = 4.56

```
   f(list, 4, 6);
   for (i = 0; i < 6; i++)
      printf("%.2lf ", list[i]);
   printf("\n");
}
```

Loop in f() does following:
arr[1] = arr[4] = 12.13
arr[0] = arr[5] = 13.14

OUTPUT:

```
1.23 4.56 4.56 1.23 12.13 13.14
4.56 4.56 4.56 1.23 12.13 13.14
13.14 12.13 4.56 1.23 12.13 13.14
```

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. `double approxLog(double x, int n);`

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

$$\log(1-x) \approx -\sum_{k=1}^{n} \frac{x^k}{k} \approx -\left(x + \frac{x^2}{2} + \frac{x^3}{3} + \cdots + \frac{x^n}{n}\right)$$

The function takes two arguments—the values of x and n, as shown above—and should return the approximate value calculated. Assume n is at least 1. For example, if x = 5 and n = 3, the function should return: $-(5 + 5^2/2 + 5^3/3) = -(5 + 12.5 + 41.6667) = -59.1667$

Students were responsible for boldfaced, underlined, italicized code

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

   // Initialize variables as needed
   tot = x;
   pow = x;

   // Loop to calculate series approximation
   for (i = 2; i <= n; i++) {
      pow *= x;
      tot += pow/i;
   }

   // Return result
   return -tot;
}
```

3 (continued)
b. `void reduceFraction(int num, int den, int *rNum, int *rDen);`

This function takes a fraction represented by numerator `num` and denominator `den`, calculates the greatest common divisor (GCD) of those numbers to reduce the fraction, and stores the reduced numerator and denominator in integers pointed to by `rNum` and `rDen`. For example, calling `reduceFraction(15, 60, &r, &d)` would reduce the fraction 15/60 to 1/4, storing 1 in `r` and 4 in `d`. Assume the denominator is always non-zero.

The algorithm for finding the GCD of two numbers, *x* and *y*, is as follows:

1. If *y* is 0, *x* is the GCD.
2. Otherwise, calculate *r*, the remainder of *x* / *y*.
3. Let *x* = *y*, and *y* = *r*.
   - So, *x* holds the "old" value of *y*, and the new value of *y* is the remainder from Step 2.
4. Return to Step 1.

```
void reduceFraction(int num, int den, int *rNum, int *rDen) {
   int gcd;     // Greatest common divisor
   int rem;     // Remainder
   int temp;    // Temporary value

   // Initialize variables—let "gcd" start as denominator and
   //   "temp" start as numerator
   gcd = den;
   temp = num;

   // Loop to calculate greatest common divisor of gcd and temp,
   //   using steps 1-4 described above (gcd → "x", temp → "y")
   while (temp != 0) {
      rem = gcd % temp;
      gcd = temp;
      temp = rem;
   }

   // Reduce each term of fraction and store results in variables
   //   to hold reduced numerator & denominator
   *rNum = num / gcd;
   *rDen = den / gcd;
}
```

3 (continued)
c. `void selectionSort(int arr[], int n);`

Complete this function to implement a simple selection sorting algorithm that will sort the array `arr[]`, which holds `n` values, from lowest to highest. The algorithm works as follows:

- The outer loop starts at the first array element and goes up to the second-to-last element.
- The inner loop finds the lowest value left between position `i` and the end of the array and swaps that value with whatever's in position `i`.

For example, consider the array {9, 5, 1, 3}:

- When i = 0: 1 is minimum value in positions 0-3, and it's swapped with 9: {1, 5, 9, 3}
- When i = 1: 3 is minimum value in positions 1-3, and it's swapped with 5: {1, 3, 9, 5}
- When i = 2: 5 is minimum value in positions 2-3, and it's swapped with 9: {1, 3, 5, 9}

```
void selectionSort(int arr[], int n) {
   int i,j;          // Loop index variables
   int iMin;         // Index--not value--of current minimum
   int temp;         // Temp value to swap elements

   // Go through first n-1 elements of array

   for (i = 0; i < n-1; i++) {

      // Find minimum value between positions i and n-1
      // Start by assuming minimum is in position i, then test
      //   all positions after that
      iMin = i;

      for (j = i+1; j < n; j++) {

         // New minimum found--store its index

         if (arr[j] < arr[iMin]) {
            iMin = j;
         }
      }

      // If position i isn't already min, swap min value with
      //   value in position i

      if (iMin != i) {
         temp = arr[iMin];
         arr[iMin] = arr[i];
         arr[i] = temp;
      }
   }
}
```