16.216: ECE Application Programming

Fall 2012

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 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 = "This"`
B. `s1 = "This", s2 = "That"`
C. `s1 = "Test", s2 = "tEsT"`
D. `s1 = "Exam 1", s2 = "Exam 2"`

i. Only A

ii. Only B

iii. B and C

iv. A and D

v. A, C, and D

b. Given the short code sequence below:

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

What will this program print?

i. `Length of string = 1`

ii. `Length of string = 3`

iii. ***<u>`Length of string = 6`</u>***

iv. `Length of string = 7`

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';
s1[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. ***<u>`Spring 1  Strin`</u>***

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. "Is Thanksgiving break here yet?"

  ii. "Is winter break here yet?"

  iii. "I think the difficulty for Programs 4, 5, and 6 was reasonable."

  iv. "I think Programs 4, 5, and 6 were too hard."

  v. "I won't know if the difficulty of those programs was reasonable until I get my grades for Programs 5 and 6."

<u>All of the above are "correct."</u>

2. (40 points) ***Functions***

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 det(int a, int b, int c, int d) {
     return (a * d) - (b * c);
}

void main() {
     int d1, d2, d3;

     d1 = det(5, 4, 4, 5);     d1   = 5 * 5 – 4 * 4
                                    = 25 – 16 = 9

     d2 = det(2, 8, 2, 10);    d2   = 2 * 10 – 8 * 2
                                    = 20 – 16 = 4

     d3 = det(d1, d1 / 3, d2 - 2, d2);
                               d3   = d1 * d2 – (d1/3) * (d2-2)
                                    = 9 * 4 – (9/3) * (4-2)
                                    = 9 * 4 – 3 * 2 = 36 – 6 = 30

     printf("%d %d %d\n", d1, d2, d3);
}
```

Output:

```
9 4 30
```

2 (cont.)

b. (14 points)

```
int f1(int *arg1) {            arg1 is passed by address
     (*arg1)--;                Decrement value arg1 points to
     return (*arg1) * 3;       Return that value multiplied by 3
}

int f2(int *arg2) {            arg2 is passed by address
     return f1(arg2) + 5;      Call f1, which will decrement
}                                value that arg2 points to,
                                 then return that value
                                 multiplied by 3.
                               Therefore, f2 returns:
                                 ((*arg2 - 1) * 3) + 5, and
                                 *arg2 is 1 less than orig. value

int f3(int arg3) {             arg3 is passed by value
     return f1(&arg3) + 3;     Call f1, which decrements arg3
}                                (but affects nothing outside
                                 this function), then returns
                                 that value multiplied by 3.
                               Therefore, f3 returns:
                                 ((arg3 - 1) * 3) + 3

void main() {
     int a, b, c;
     int x, y, z;
     a = b = c = 10;      // Set all three values to 10
     x = f1(&a);          x    = (a – 1) * 3
                               = (10 – 1) * 3 = 9 * 3 = 27
                          a    = a – 1 = 9

     y = f2(&b);          y    = ((b – 1) * 3) + 5
                               = ((10 – 1) * 3) + 5
                               = (9 * 3) + 5 = 27 + 5 = 32
                          b    = b – 1 = 9

     z = f3(c);           z    = ((c – 1) * 3) + 3
                               = ((10 – 1) * 3) + 3
                               = (9 * 3) + 3 = 27 + 3 = 30

     printf("%d %d %d\n", a, b, c);
     printf("%d %d %d\n", x, y, z);
}
```

Output:

```
9 9 10
27 32 30
```

2 (cont.)

c. (14 points)

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

     for (i = 0; i < (n / 2); i++) {    Loop goes through n/2
          arr[i] = arr[n - i - 1];         values, copying values
     }                                     from higher array
                                           positions to lower
}                                          array positions
                                        For example, if n = 4,
                                           first loop iteration
                                           copies arr[4-0-1] =
                                           arr[3] into arr[0]


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

     for (i = 0; i < 6; i++)
          printf("%d ", x[i]);
     printf("\n");

     f(x, 4);                Before function, x = {1, 2, 3, 4, 5, 6}
                             After function, x = {4, 3, 3, 4, 5, 6}

     for (i = 0; i < 6; i++)
          printf("%d ", x[i]);
     printf("\n");

     f(x, 6);                Before function, x = {4, 3, 3, 4, 5, 6}
                             After function, x = {6, 5, 4, 4, 5, 6}

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

Output:

```
1 2 3 4 5 6
4 3 3 4 5 6
6 5 4 4 5 6
```

3. (40 points, 20 per part) ***Arrays***
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. `void countDigits(int inVals[], int n, int count[10]);`

This function takes the following inputs:

- `int inVals[]`: Each value in this array is an integer between 0 and 9.
- `int n`: The number of elements in `inVals[]`.
- `int count[10]`: An array that will hold the number of occurrences of each value in the array `inVals[]` when the function is complete—`count[i]` will hold the number of times the value `i` is stored in `inVals[]`. For example:
  - `inVals[] = {3,5,7,3,0}` → `count[] = {1,0,0,2,0,1,0,1,0,0}`
  - `inVals[] = {1,1,1,1}` → `count[] = {0,4,0,0,0,0,0,0,0,0}`
  - `inVals[] = {0,1,2,3,4,5,6,7,8,9}`
    → `count[] = {1,1,1,1,1,1,1,1,1,1}`

Complete this function so that it checks all values within `inVals[]` and sets the values within `count[]` appropriately.

```
void countDigits(int inVals[], int n, int count[10]) {
     int i;                  // Loop index

     /**** RESET ALL ELEMENTS OF count[] TO 0 ****/
     for (i = 0; i < 10; i++)
          count[i] = 0;

     /**** VISIT EACH ELEMENT OF inVals[] AND UPDATE
           count[]APPROPRIATELY. FOR EXAMPLE, IF
           inVals[i] == 0, ADD 1 TO count[0]     *****/
     for (i = 0; i < n; i++)
          count[inVals[i]]++;

}
```

3 (cont.)

b. `void bubbleSort(int list[], int n)`

Complete this function to implement a simple yet inefficient sorting algorithm called a bubble sort that will sort the array `list[]`, which contains `n` elements. The algorithm uses two loops:

- The inner loop visits all elements of `list[]` and compares consecutive pairs of values. If the elements are out of order, they are swapped. For example, if `list[]` initially holds `{1,4,3,2}`:
    - First inner loop iteration: 1 < 4 → no swap
    - Second inner loop iteration: 4 > 3 → swap; list now holds `{1,3,4,2}`
    - Third inner loop iteration: 4 > 2 → swap; list now holds `{1,3,2,4}`
- The outer loop executes as long as at least one swap occurs in the inner loop, a condition indicated by the variable `swapped`. In the example above, the outer loop would execute at least one more time, since two swaps occurred in the inner loop.

```
void bubbleSort(int list[], int n) {
     int i;         // Loop index
     int temp;      // Temporary variable
     int swapped;   // Variable that indicates a swap occurred

     do {               // Start of outer loop
          swapped = 0;  // Clear swapped → if swapped == 0 at
                        // end of inner loop, no swaps this time

          /* INNER LOOP: CHECK ALL CONSECUTIVE PAIRS OF
             VALUES; SWAP ANY PAIRS THAT ARE OUT OF ORDER.
             IF SWAP OCCURS, SET swapped TO APPROPRIATE VALUE */
          for (i = 0; i < n - 1; i++) {
               if (list[i] > list[i+1]) {
                    temp = list[i];
                    list[i] = list[i+1];
                    list[i+1] = temp;
                    swapped = 1;
               }
          }

     /***** FILL IN APPROPRIATE CONDITION IN BLANK SPACE BELOW
            SO OUTER LOOP EXECUTES ANOTHER ITERATION IF ANY
            VALUES WERE SWAPPED INSIDE INNER LOOP          ****/

     } while (swapped == 1);
}
```

3 (cont.)

c. `void progAvgDrop1(double s[][10], double avg[], int n)`

This function takes the following arguments:

- `double s[][10]`: A 2D array in which each row represents the programming assignment scores for a single student; each student has 10 assignment scores.
- `double avg[]`: A 1D array that will hold the average score for each student after the lowest score is dropped.
- `int n`: The number of rows in `s[][10]` and the number of elements in `avg[]`.

Complete this function so that it calculates an average for each student in which the lowest score is dropped and stores that average in the appropriate entry of `avg[]`. In other words, `avg[x]` is the average of the 9 highest values in row `x` of `s[][10]`.

```
void avgDrop1(double s[][10], double avg[], int n) {
     double min;         // Minimum exam score for given student
     int i, j;           // Row & column numbers
     double sum;         // Running sum used to compute average

     /* SET UP OUTER LOOP TO HANDLE EACH ROW; SHOULD INITIALIZE
        VARIABLES COMPUTED ON ROW-BY-ROW BASIS HERE */

     for (i = 0; i < n; i++) {
          sum = s[i][0];
          min = s[i][0];

          /* SET UP INNER LOOP TO GO THROUGH ALL COLUMNS AND
             CALCULATE RUNNING TOTAL WHILE ALSO ACCOUNTING FOR
             LOWEST VALUE */

          for (j = 1; j < 10; j++) {
               sum += s[i][j];
               if (s[i][j] < min)
                    min = s[i][j];
          }

          /* COMPUTE AVG FOR EACH ROW WITHOUT LOWEST VALUE */
          avg[i] = (sum – min) / 9;
     }
}
```