EECE.2160: ECE Application Programming

Spring 2018

Exam 2 Solution

1. (33 points) ***Functions***

a. (13 points) Show the output of the short program below 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 I can easily recognize your final answer.

```
void badSwap(int x, int *y) {
   int temp = x;                      Copies the value of x to
   x = *y;                            the variable y points to
   *y = temp;
}

void badSwap_2(int *x, int y) {
   int temp = *x;                     Copies the value of y to
   *x = y;                            the variable x points to
   y = temp;
}

int main() {
   int a = 2;
   int b = 1;
   int c = 6;
   int d = 0;

   badSwap(a, &b);                    Sets b = a = 2
   badSwap_2(&b, c);                  Sets b = c = 6
   printf("%d %d %d %d\n", a, b, c, d);

   badSwap_2(&a, c);                  Sets a = c = 6
   badSwap(b, &d);                    Sets d = b = 6
   printf("%d %d %d %d\n", a, b, c, d);

   return 0;
}
```

OUTPUT:

```
2 6 6 0
6 6 6 6
```

1 (continued)
b. (20 points) Complete the function described below:

```
void splitDay(int doy, int *m, int *dom);
```

Given an integer, `doy`, representing the day within a year of 365 days, determine the appropriate month and day and store them in the variables pointed to by `m` and `dom`, respectively. For example, calling `splitDay(10, &month, &day)` would set `month = 1` and `day = 10`, since the 10th day of the year is January 10; `splitDay(360, &month, &day)` would set `month = 12` and `day = 26`, as the 360th day of the year is December 26.

Students were responsible for writing bold, underlined, italicized code.

```
void splitDay(int doy, int *m, int *dom) {
   int daysPerMo[] = { 31, 28, 31, 30, 31, 30,     // Days in
                       31, 31, 30, 31, 30, 31 };   //  each month

   int sum;     // Sum of days after a certain number of months

   // Initialize variables
   *m = 0;
   sum = 0;

   // Find the month by adding the number of days in all previous
   //   months--when you've added too many days, the previous
   //   month is the correct one

   while (sum + daysPerMo[*m] < doy) {
      sum += daysPerMo[*m];
      (*m)++;
   }

   // Finalize month and day values--day of month is what's left
   //   in the partial month after you've added all prior months
   (*m)++;
   *dom = doy - sum;
}
```

2. (47 points) ***Arrays***

a. (14 points) Show the output of the short program below 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 I can easily recognize your final answer.

```
int main() {
   int i;
   int list[8] = {4, 8, -1, -3};    list = {4, 8, -1, -3,
                                             0, 0, 0, 0}

   for (i = 1; i < 8; i += 2) {      Adds index to odd numbered
      list[i] = list[i] + i;         array elements, so:
                                     list[1] = list[1] + 1 = 9
                                     list[3] = list[3] + 3 = 0
                                     list[5] = list[5] + 5 = 5
                                     list[7] = list[7] + 7 = 7

      Line below prints consecutive pairs of array elements, with
         higher-indexed value first
      printf("%d %d\n", list[i], list[i - 1]);
   }

   for (i = 7; i > 0; i /= 2) {
      list[i] += list[i/2];          list[7] += list[3]  = 7
      printf("%d ", list[i]);        list[3] += list[1]  = 9
   }                                 list[1] += list[0]  = 13

   printf("\n");
   return 0;
}
```

OUTPUT:

```
9 4
0 -1
5 0
7 0
7 9 13
```

2 (continued)

b. (13 points) Show the output of the short program below 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.

```
int main() {
   int i, j;
   int arr[5][2];
   int q = -5;

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

Loop goes thru array row by row & assigns/prints values

Since q is pre-incremented, arr[0][0] = -4, arr[0][1] = -3, etc.

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

Loop prints array column by column, starting with last column and last row

OUTPUT:

```
-4 -3
-2 -1
0 1
2 3
4 5
5 3 1 -1 -3
4 2 0 -2 -4
```

2 (continued)

c. (20 points) Complete the function described below:

`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]`.

Students were responsible for writing bold, underlined, italicized code.

```
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

  // Outer loop to handle rows

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

    // Initialize values that reset on a per-row basis
    sum = s[i][0];
    min = s[i][0];

    // Inner loop to handle columns—body of this loop must
    //   add entire row and find value to drop

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

    // Compute row average without dropped value
    avg[i] = (sum - min) / 9;
  }
}
```

3. (20 points, 5 points each) ***Multiple choice***

For questions 3a and 3b, circle or underline the one choice you think best answers the question.

a. Given the following code snippet:

```
int x = 100;
for (i = 1; i < 5; i++) {
   x = x - 10;
}
```

Which of the following choices can replace the `for` loop and produce the exact same final value for `x`? Assume `x` is always initialized to 100.

i.
```
i = 5;
while (i >= 1) {
   x = x - 10;
   i--;
}
```

ii.
```
i = 0;
while (i < 5) {
   x = x - 10;
   i++;
}
```

[✓ ***iii.***]
```
i = 0;
while (i < 8) {
   x += (-10);
   i += 2;
}
```

iv.
```
i = x;
while (i > 60) {
   i = i - 10;
}
```

3 (continued)

b. What is the output of the short code sequence below?

```
int i;
for (i = 0; i < 15; i += 3) {
   printf("%d ", i++);
   printf("%d ", ++i);
}
```

i. `0 3 6 9 12`

ii. ***`0 2 5 7 10 12`***

iii. `1 2 6 7 11 12`

iv. `1 1 6 6 11 11`

v. `3 30 2018`

c. Assume each of the loops below accesses the contents of an array declared as: `int x[10]`. Which of the loops only accesses values inside the array (in other words, all loop indexes are valid)? **This problem may have more than one correct answer.**

i.
```
for (i = 0; i < 10; i++)
   printf("%d %d", x[i], x[i+1]);
```

ii.
```
for (i = 500; i > 0; i -= 20)
   arr[i%10] = arr[i%5] + arr[i%7];
```

iii.
```
for (i = 0; i <= 10; i++)
   arr[i] -= 5;
```

iv.
```
i = 0;
do {
   printf("%d ", i);
   i = arr[i];
} while (i >= 0 && i < 10);
```

v.
```
for (i = 5; i < 50; i += 5)
   printf("%d ", arr[i/5]);
```

3 (continued)

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?"

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

4. (10 points) ***EXTRA CREDIT***

Write the function with the function prototype and description below:

`void insertionSort(int arr[], int n);`

This function sorts `n` values in an integer array, `arr`, from lowest to highest, as follows:

1. Start with the second array element; copy it into a temporary variable (`temp`).
2. Compare `temp` to all values to the left (i.e., with a lower array index) until you find the appropriate spot for that value. At each step:
    a. If the value you're testing is greater than `temp`, move that value one spot to the right.
    b. Otherwise, place `temp` one spot to the right and go to step 3.
    c. If you test all array elements to the left of the current one and don't stop at step (b), `temp` should be the new first value in the array.
3. Copy the 3rd array element to `temp`; repeat steps 2a-2c. Continue with the 4th, 5th, etc.

The example below shows how the function would sort the array {5, 2, 3, 7}:

Step 1: `temp = 2` `{5,2,3,7}` → `{5,`**5**`,3,7}` → `{`**2**`,5,3,7}`

Step 2: `temp = 3` `{2,5,3,7}` → `{2,5,`**5**`,7}` → `{2,`**3**`,5,7}`

Step 3: `temp = 7` No sorting done, since 7 is greater than all values to its left

```
void insertionSort(int arr[], int n) {
   int i, j;          // Array index variables
   int temp;          // Temporary value

   // Start with 2nd value and visit all remaining array elements
   for (i = 1; i < n; i++) {

      // Copy current array value to temp
      temp = arr[i];

      // Test values to the left of the current position until
      //   you find right spot for temp or have tested all values
      // Values greater than temp should be shifted right 1 spot
      j = i – 1;
      while((temp < arr[j]) && (j >= 0)) {
         arr[j+1] = arr[j];
         j = j-1;
      }

      // At this point, you've found right spot--copy temp there
      arr[j+1] = temp;
   }
}
```