EECE.2160: ECE Application Programming

Spring 2018

Exam 2
March 30, 2018

Name: ____

Lecture time (circle 1): *8-8:50 (Sec. 201)* *12-12:50 (Sec. 202)*

For this exam, you may use only one 8.5" x 11" double-sided page of notes. All electronic devices (e.g., calculators, cell phones) are prohibited. Please turn off your cell phone ringer prior to the start of the exam to avoid distracting other students.

The exam contains 3 questions for a total of 100 points, plus a 10 point extra credit question. 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:

- Questions 1b and 2c require you to complete short functions. We have provided comments to describe what each function should do and written some of the code.
  - Note that each function contains both lines that are partially written (for example, a `printf()` call missing the format string and expressions) and blank spaces in which you must write additional code. **<u>You must write all code required to make each function work as described—do not simply fill in the blank lines.</u>**
  - Each test case is an example of how the program should behave in one specific case—**<u>it does not cover all possible results of running that program.</u>**
  - You can solve each of these questions using only the variables that have been declared, but you may declare and use other variables if you want.
- Carefully read the multiple choice problems. Questions 3a and 3b have exactly <u>one</u> correct answer, while Question 3c <u>may have more than one correct answer</u>.
- **<u>You cannot earn any extra credit without partial solutions to all parts of Questions 1, 2, and 3.</u>** In other words, don't try the extra credit until you've tried to solve every other question on the exam.

You will have 50 minutes to complete this exam.

Q1: Functions	/ 33
Q2: Arrays	/ 47
Q3: Multiple choice	/ 20
TOTAL SCORE	/ 100
Q4: EXTRA CREDIT	/ 10

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;
   x = *y;
   *y = temp;
}

void badSwap_2(int *x, int y) {
   int temp = *x;
   *x = y;
   y = temp;
}

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

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

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

   return 0;
}
```

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.

```
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







   // 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 (_____________________________________________________) {







   }

   // Finalize month and day values--day of month is what's left
   //   in the partial month after you've added all prior months







}
```

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};

   for (i = 1; i < 8; i += 2) {
      list[i] = list[i] + i;
      printf("%d %d\n", list[i], list[i - 1]);
   }

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

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");
   }

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

2 (continued)

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

a. `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

  // Outer loop to handle rows

  for (_______________________________________________________) {

    // Initialize values that reset on a per-row basis



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

    for (___________________________________________________) {






    }
    // Compute row average without dropped value


  }
}
```

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?"

4. (10 points) ***<u>EXTRA CREDIT</u>***

<u>REMEMBER, YOU CANNOT EARN EXTRA CREDIT WITHOUT WRITING AT LEAST PARTIAL SOLUTIONS FOR ALL OTHER PROBLEMS ON THE EXAM.</u>

<u>However, you can earn partial credit for a partial solution to this problem.</u>

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}:

<u>Step 1:</u> `temp = 2` `{5,2,3,7}` → `{5,`**<u>5</u>**`,3,7}` → `{`**<u>2</u>**`,5,3,7}`

<u>Step 2:</u> `temp = 3` `{2,5,3,7}` → `{2,5,`**<u>5</u>**`,7}` → `{2,`**<u>3</u>**`,5,7}`

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

Use the space on the next page to write your solution.

4 (continued) ***<u>SPACE TO SOLVE EXTRA CREDIT PROBLEM</u>***

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










}
```