EECE.2160: ECE Application Programming

Fall 2018

Exam 2 Solution

1. (35 points) ***Functions***

a. (15 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 f1(int *arg1) {
   (*arg1)--;
   return (*arg1) * 3;
}

int f2(int arg2) {
   ++arg2;
   return arg2 + 10;
}


int main() {
   int a, b, c, d;
   a = 5;
   b = f1(&a);


   c = f2(a);

   d = f1(&b);


   a = f2( f1(&c) );


   printf("%d %d %d %d\n", a, b, c, d);
   return 0;
}
```

Since arg1 is passed by address, this function decrements whatever arg1 points to outside the function, & returns that value * 3

Since arg2 is passed by value, incrementing it in the function has no effect outside f2(), but the incremented value is used to calculate the return value

a = a – 1 = 4
b = a * 3 = 12

c = (a + 1) + 10 = 15

b = b – 1 = 11
d = d * 3 = 33

Call to f1 is evaluated first:
c = c – 1 = 14
Function returns 14 * 3 = 42
f1() return value passed to f2():
a = (42 + 1) + 10 = 53

OUTPUT:

53 11 14 33

1 (continued)

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

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

This function should calculate the following series approximation for the value 1 / (1-x), which is valid if the absolute value of x is less than 1:

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

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
   total = 0;
   x_i = 1;


   // Loop to calculate series approximation as described above
   for (i = 0; i <= n; i++) {
      total = total + x_i;
      x_i = x_i * x;            // After 1 iteration, x_i = x
                                // After 2 iterations, x_i = x^2
                                // After 3 iterations, x_i = x^3
                                //  … etc.
   }

   // Return result
   return total;
}
```

2. (41 points) ***Arrays***

a. (15 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, k;
   int vals[8] = {21, 60, 1, 6, -5, 7};
      vals actually holds: {21, 60, 1, 6, -5, 7, 0, 0}

   int inds[8] = {3, 1, -1, 6, 8, 0, 10, 5};

   for (i = 6; i >= 0; i -= 2) {
      vals[i] += inds[i];
      printf("%d %d\n", vals[i], vals[i+1]);
      printf("%d %d\n", inds[i], inds[i+1]);
   }

   for (i = 0; i < 8; i++) {
      k = inds[i];
      if (k >= 0 && k < 8)
         printf("%d\n", vals[k]);
      else
         printf("%d\n", k);
   }

   return 0;
}
```

Loop changes even-numbered positions in vals; prints consecutive values in each array

Loop uses inds[i] as index into vals if valid (between 0 and 7) and prints vals[inds[i]] Otherwise, loop prints inds[i]

OUTPUT

```
10 0
10 5
3 7
8 0
0 6
-1 6
24 60
3 1
6
60
-1
10
8
24
10
7
```

2 (continued)

b. (6 points) You are given a two dimensional array declared as: `double arr[5][10];`

You also have two `int` variables, `i` and `j`, and two variables of type `double`, `sum` and `avg`.

Which of the following short code sequences correctly calculate the average of all values in this array and stores it in the variable `avg`? **This question has at least one correct answer, but may have more than one correct answer! Circle ALL choices that correctly answer the question.**

i.
```
sum = 0;
for (i = 0; i < 5; i++) {
   for (j = 0; j < 10; j++) {
      sum += arr[i][j];
   }
   avg += sum / 10;
}
```

ii.
```
sum = 0;
for (i = 4; i >= 0; i--) {
   for (j = 9; j >= 0; j--) {
      sum += arr[i][j];
   }
}
avg = sum / 50;
```

iii.
```
sum = 0;
for (i = 0; i < 10; i++) {
   for (j = 0; j < 5; j++) {
      sum = sum + arr[j][i];
   }
}
avg = sum / 50;
```

iv.
```
sum = 0;
for (i = 0; i < 5; i++) {
   for (j = 0; j < 10; j++) {
      sum = arr[i][j];
   }
}
avg = sum / 50;
```

2 (continued)

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

```
int findMode(int arr[], int n);
```

This function finds and returns the mode—the value that occurs most often—in an integer array `arr[]` of length `n`. For example, if `arr = {1, 2, 3, 3, 4, 4, 4}`, the mode is 4; if `arr = {1, 6, 2, 1, 6}`, the mode is 6.

To find the mode, the function counts the number of occurrences of each value in `arr[]`, and then determines the maximum number of occurrences, which in turn determines the mode. You may assume `arr[]` contains only values between 1 and 10, so the `numCount[]` array declared in the function can be used to count the occurrences of all possible values.

Solution notes:

- *Since we know all values in arr[] are between 1 and 10, numCount[0] will count the number of 1s, numCount[1] counts the number of 2s, and so on, up to numCount[9], which counts the number of 10s.*
- *Your mode variable will essentially be the index (+1) of the position in numCount that has the highest value after the occurrence-counting loop.*

```
int findMode(int arr[], int n) {
   int numCount[10]; // Counts number occurrences in arr
   int i;            // Loop index
   int mode;         // Mode

   // Initialize variables as needed (may need loop)
   for (i = 0; i < 10; i++)
      numCount[i] = 0;
   mode = 1;

   // Use loop to count # of occurrences of each value in arr
   for (i = 0; i < n; i++) {
      numCount[ arr[i] - 1 ]++;
   }

   // Loop to determine mode--which number occurred most often
   for (i = 1; i < 10; i++) {
      if (numCount[i] > numCount[mode-1])
         mode = i + 1;
   }

   return mode;
}
```

3. (24 points, 6 points each) ***Strings***

a. Given strings `s1 = "Exam 1"`, `s2 = "Exam 2"`, and `s3 = "Exactly"`, which of the following calls to string comparison functions will return the value 0? **This question has at least one correct answer, but may have more than one correct answer! Circle ALL choices that correctly answer the question.**

i. `strcmp(s1, s2);`

ii. strncmp(s1, s2, 5);

iii. `strcmp(s1, s3);`

iv. strncmp(s1, s3, 3);

b. Given the short code sequence below:

```
int i;
char str[20] = "x";
for (i = 0; i < 4; i++)
   strcat(str, "ox");
printf("Length of string = %d\n", strlen(str));
```

What will this program print? **Choose only one answer.**

i. `Length of string = 1`

ii. `Length of string = 3`

iii. `Length of string = 5`

iv. Length of string = 9

v. `Length of string = 20`

3 (continued)

c. Given the code sequence below:

```
char s1[20];
char s2[20];
strcpy(s1, "Summer");
s1[1] = 'i';
strncpy(s2, s1, 4);
s2[3] = '\0';
printf("%s  %s\n", s1, s2);
```

What will this program print? **Choose only one answer.**

i. `Summer  Simmer`

ii. `Simmer  Simmer`

iii. `Summer  Simm`

iv. `Simmer  Simm`

v. ***`Simmer  Sim`***

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 are "correct."

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

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

However, you can earn partial credit for a partial solution to this problem.

Write the function with the function prototype and description below:

```
void removeChar(char *str, char c);
```

This function should remove all occurrences of the character `c` from the string `str`. Remember that, since the string is passed by address, changes made to `str` inside the function are seen outside the function.

For example, if the string `s1 = "This is a message"` before each function call shown below, then:

- After calling `removeChar(s1, 'i')` → `s1 = "Ths s a message"`
- After calling `removeChar(s1, ' ')` → `s1 = "Thisisamessage"`
- After calling `removeChar(s1, 's')` → `s1 = "Thi i a meage"`

Hint: to access part of a string (a substring) starting after the first character, use the address of the desired starting character. For example, given the original `s1` above, `&s1[5]` can be used to access the substring `"is a message"`. Using substrings with built-in string functions can simplify your solution.

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

4 (continued) ***SPACE TO SOLVE EXTRA CREDIT PROBLEM***

```
void removeChar(char *str, char c)
{
   int i = 0;        // Index variable

   // Loop to ensure all characters are covered
   while (i < strlen(str)) {

      // If match is found, shift all characters that follow
      //    to the left by one position
      // NOTE: This solution uses the strcat() function—if
      //    the character to be removed is overwritten with
      //    a null character, str[] essentially holds two
      //    strings--one before the removed character, and
      //    one after. strcat() combines the two,
      //    overwriting the '\0' with the next character.
      if (str[i] == c) {
         str[i] = '\0';
         strcat(str, &str[i+1]);
      }

      // If there is no match, move onto the next character
      // NOTE: i should not be incremented first since,
      //   doing so would lead to a character being skipped
      else
         i++;
   }
}
```