EECE.2160: ECE Application Programming

Fall 2017

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 f(int a1, int *a2) {
   int temp = *a2;        Copy value to which a2 points
   *a2 = a1 * 2;          Double value of first argument and
                               store wherever a2 points

   a1 = temp;             Overwrite a1 with copy of original
   return a1 / 2;              value to which a2 pointed and
}                              return half of that value (does
                               not change original variable
                               passed as a1, since that argument
                               is passed by value)

int main() {
   int v1, v2, v3, v4;

   v1 = 20;
   v2 = 30;
   v3 = f(v1, &v2);       Sets v2 = v1 * 2 = 20 * 2 = 40 and
                               v3 = original v2 / 2 = 30 / 2 = 15

   printf("%d %d %d\n", v1, v2, v3);

   v4 = f(v2, &v3);       Sets v3 = v2 * 2 = 40 * 2 = 80 and
                               v4 = original v3 / 2 = 15 / 2 = 7

   printf("%d %d %d %d\n", v1, v2, v3, v4);

   return 0;
}
```

OUTPUT:

```
20 40 15
20 40 80 7
```

1 (continued)

b. (20 points) Complete the function described below: `void squareGrid(int nBox);`

This function prints a square grid, where the number of boxes in each row and column of the grid is determined by the input argument `nBox`. For each iteration of the "row" loop, the function actually prints two rows of output: a set of `'+'` and `'-'` characters for the top or bottom part of a set of boxes, and then a set of `'|'` and `' '` characters for the boxes' middle part.

For example, calling `squareGrid(3)` would generate the following output:

```
+-+-+-+
| | | |
+-+-+-+
| | | |
+-+-+-+
| | | |
+-+-+-+
```

```
void squareGrid(int nBox) {
  int   i, j;  // Loop indexes

  // Outer loop to control number of rows
  for (i = 0; i <= nBox; i++) {

    // Loop to print top/bottom of boxes
    for (j = 0; j <= nBox; j++) {
      if (j < nBox)
        printf("+-");
      else
        printf("+\n");
    }

    // For all rows except last one, print middle of boxes
    if (i < nBox) {
      for (j = 0; j <= nBox; j++) {
        if (j < nBox)
          printf("| ");
        else
          printf("|\n");
      }
    }
  }
}
```

2. (45 points) ***Arrays***

a. (12 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 arr[6] = {9, 3, 4};          arr[] = {9, 3, 4, 0, 0, 0}
  int i;

  for (i = 5; i >= 0; i--) {       Print array contents from
    printf("%d ", arr[i]);         last element to first

    if (arr[i] == 0)               Changes last three elements
      arr[i] = arr[5-i];           arr[5] = arr[0] = 9
                                   arr[4] = arr[1] = 3
                                   arr[3] = arr[2] = 4

    else                           Changes first three elements
      arr[i] = arr[i+1] - 1;       arr[2] = arr[3] - 1 = 3
                                   arr[1] = arr[2] - 1 = 2
                                   arr[0] = arr[1] - 1 = 1
  }
  printf("\n");
  for (i = 0; i < 3; i++)                    Prints elements in
    printf("%d %d\n", arr[i], arr[i+3]);     pairs, with indexes
                                             separated by 3
                                             (arr[0] & arr[3],
                                             arr[1] & arr[4],
                                             arr[2] & arr[5])

  return 0;
}
```

OUTPUT:

```
0 0 0 4 3 9
1 4
2 3
3 9
```

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. ***NOTE: Both printf() calls that print values from list[][] use a precision of 1.***

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;
   double list[2][4] = { {1.2, 3.4, 5.6, 7.8},
                         {9.8, 7.6, 5.4, 3.2} };

   for (i = 1; i >= 0; i--) {                  Prints array contents
      for (j = 3; j >= 0; j--) {               with last row first
         printf("%.1lf ", list[i][j]);         and last column first in
      }                                        each row
      printf("\n");
   }

   for (i = 0; i < 8; i++)                     Prints array elements
      printf("%.1lf\n", list[i%2][i%4]);       based on modulus result:
   return 0;                                   i = 0 → list[0][0]
}                                              i = 1 → list[1][1]
                                               i = 2 → list[0][2]
                                               i = 3 → list[1][3]
                                               i = 4 → list[0][0]
                                               i = 5 → list[1][1]
                                               i = 6 → list[0][2]
                                               i = 7 → list[1][3]
```

OUTPUT:

```
3.2 5.4 7.6 9.8
7.8 5.6 3.4 1.2
1.2
7.6
5.6
3.2
1.2
7.6
5.6
3.2
```

2 (continued)

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

`int mostWins(int p1[], int p2[], int n);`

This function takes three arguments: arrays `p1` and `p2`, which represent the scores of games between two players, and an integer `n`, which represents the number of games. The function should determine if player 1 or player 2 won more games and return the winning player's number. If both players won the same number of games, the function returns 0. For example:

- `mostWins({1,1,1}, {3,-1,2}, 3)` returns 2 (P2 won 2 of 3 games)
- `mostWins({0,1,2,3}, {-1,-2,-3,-4}, 4)` returns 1 (P1 won 4 of 4 games)
- `mostWins({1,2,3}, {3,2,1}, 3)` returns 0 (Each player won 1 game and tied 1 game, so neither player won more games than the other)

```
int mostWins(int p1[], int p2[], int n) {
   int i;              // Loop index
   int p1W, p2W;       // Wins for each player

   // Initialize variables
   p1W = p2W = 0;

   // Go through arrays and determine winner of each match,
   //  keeping track of the total number of wins for each player
   for (i = 0; i < n; i++) {

      // Player 1 won
      if (p1[i] > p2[i])
         p1W++;

      // Player 2 won
      else if (p2[i] < p1[i])   // Can't just use else—array
         p2W++;                 //  entries could be equal!
   }

   // Return number of winning player or 0 if tied
   if (p1W > p2W)
      return 1;

   else if (p2W > p1W)
      return 2;

   else
      return 0;
}
```

3. (20 points, 5 points each) ***<u>For loops; strings</u>***

For each of the multiple choice questions below, clearly indicate your response by circling or underlining the <u>one</u> choice you think best answers the question.

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

```
int i;
int val = 1;
for (i = 0; i < 5; i += val) {
   val += i;
   printf("%d %d ", i, val);
}
```

i. `0 1`

ii. `1 1 2 3`

iii. `0 1 2 3 4`

iv. [✓ ***<u>`0 1 1 2 3 5`</u>***]

v. `0 1 1 2 2 4 3 7 4 11`

b. Given two strings, `s1 = "Exam 2"` and `s2 = "Exasperated"`, which of the following function calls will return 0?

i. `strlen(s1);`

ii. `strcmp(s1, s2);`

iii. [✓ ***<u>`strncmp(s1, s2, 3);`</u>***]

iv. `strncat(s1, s2, 3);`

v. All of the function calls above return non-zero values

3 (continued)

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

```
char s1[50] = "Q";
char s2[50] = "3";
int i;

for (i = 0; i < 3; i++) {
   strcat(s1, s2);
   strncpy(s2, s1, i + 1);
   s2[i+1] = '\0';         // Ensure s2 is null terminated
                           // Does not affect final output
}
printf("%s %s\n", s1, s2);
```

i. `Q 3`

ii. `Q3 Q`

iii. `Q3Q Q3`

iv. ***<u>Q3QQ3 Q3Q</u>***

v. `Q3QQ3Q3Q Q3QQ`

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 answers are "correct."</u>

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:

```
unsigned int substrMatch(char *s1, char *s2, unsigned int len,
                         unsigned int *pos1, unsigned int *pos2);
```

A substring is a short string within a larger string that can be defined by its starting position and length. For example, given the string `char s[] = "Example"`, a substring of length 3 starting at position 2 within `s` is `"amp"`.

The `substrMatch()` function searches its two string arguments, `s1` and `s2`, to see if both strings contain a matching substring of length `len`.

- If a match exists, the function returns 1, and the starting positions of the matching substring within `s1` and `s2` are stored in the variables pointed to by `pos1` and `pos2`, respectively.
- If no match exists, the function returns 0, and the variables pointed to by `pos1` and `pos2` are unchanged.

Notes and hints:

- Given a string `s[]`, you can access a substring at position `i` using `&s[i]`. For example, given `char s[] = "Example"`, `printf(&s[2]);` would print `ample`
- If there are multiple matching substrings of the desired length in `s1` and `s2`, your function should "find" the substring that appears earliest in `s1`.
- The only string functions you are allowed to use in your solution are the ones we discussed in class: `strcpy()/strncpy()`, `strcmp()/strncmp()`, `strlen()`, and `strcat()/strncat()`.

Test cases:

Given `char q[] = "strictest"`, `char r[] = "test"`, and unsigned ints `p1`, `p2`:

- `substrMatch(q, r, 2, &p1, &p2)` returns 1, with `p1 = 0` and `p2 = 2`
  (a substring of length 2 (`"st"`) is found at position 0 in `q` and position 2 in `r`)
- `substrMatch(q, r, 3, &p1, &p2)` returns 1, with `p1 = 5` and `p2 = 0`
  (a substring of length 3 (`"tes"`) is found at position 5 in `q` and position 0 in `r`)
- `substrMatch(q, r, 5, &p1, &p2)` returns 0
  (no matching substring of length 5 exists in the 2 strings)

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

4 (continued) ***SOLUTION*** A few notes on the solution:

- Basically, you need to go through both `s1` and `s2` until either (1) you find a matching substring in both strings or (2) you reach the end of both without finding any match.
    - "Going through" the strings requires you to track your position in each, so the solution contains a pair of for loops, one for `s1` and one for `s2`.
    - The upper limit on each loop is the length of the string minus `len`, which is the last possible position a substring of length `len` could start.
- The hint about accessing a substring through the address of its starting character tells you can use any string function with a substring by passing in its starting address.
    - That means you can use string comparison functions to compare substrings by passing the starting addresses of those substrings.
    - So, the `strncmp()` function is a perfect fit for this problem, since it will allow us to compare two substrings of our desired length.
- If you didn't think of using the `strncmp()` function, you could basically write your own version—a loop that goes through `len` characters in each string, starting at position `i` in `s1` and `j` in `s2`, and tracks whether all the characters match.

```
unsigned int substrMatch(char *s1, char *s2, unsigned int len,
                         unsigned int *pos1, unsigned int *pos2)
{
  int i, j;         // Loop indexes

  // Go through both strings, comparing substrings of length
  //   "len" until a match is found
  // strlen(<string>) - len is last position in which
  //   substring of length "len" can start in <string>
  for (i = 0; i <= strlen(s1) - len; i++) {
    for (j = 0; j <= strlen(s2) - len; j++) {

      // Match found at position i in s1, j in s2,
      //   so assign i to whatever pos1 points to,
      //   j to whatever pos2 points to, and return 1
      if (strncmp(&s1[i], &s2[j], len) == 0) {
        *pos1 = i;
        *pos2 = j;
        return 1;
      }
    }
  }

  // If you reach the end of both loops without returning,
  //   no match --> return 0 without changing positions
  return 0;
}
```