16.216: ECE Application Programming

Fall 2011

Exam 3 Solution

1. (24 points, 4 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. You have a file, `output.txt`, that contains the following data:

```
1 2 3
abc
```

Which of the following code sequences could have produced this file?

i.
```
FILE *fp;
fp = fopen("output.txt", "r");
fscanf(fp, "%d %d %d", x, y, z);
fscanf(fp, "%s", str);
fclose(fp);
```

ii.
```
FILE *fp;
fp = fopen("output.txt", "w");
printf("%d %d %d\n", x, y, z);
printf("%s\n", str);
fclose(fp);
```

iii.
`FILE *fp;`
`fp = fopen("output.txt", "w");`
`fprintf(fp, "%d %d %d\n", x, y, z);`
`fprintf(fp, "%s\n", str);`
`fclose(fp);`

iv.
```
fprintf(stdout, "%d %d %d\n", x, y, z);
fprintf(stdout, "%s\n", str);
```

1 (cont.)

b. You have a program that contains an array declared as:

```
double arr[40];
```

Which of the following code snippets would correctly store the contents of this array in a binary file?

i.
```
FILE *fp = fopen("output.txt", "w");
fread(arr, sizeof(double), 40, fp);
```

ii.
```
FILE *fp = fopen("output.txt", "w");
fprintf(fp, "%lf", arr);
```

iii. ***<u>`FILE *fp = fopen("output.txt", "w");`</u>***
<u>`fwrite(arr, sizeof(double), 40, fp);`</u>

iv.
```
FILE *fp = fopen("output.txt", "w");
printf("The contents of this array in a binary file\n");
```

1 (cont.)
The following question uses the structure defined below:

```
typedef struct {
  char name[50];
  double progs[8];
  double exams[3];
} CourseData;
```

c. Which of the following choices is not a valid access to a field within a variable of type `CourseData`?

i.
```
CourseData cd;
cd.progs[7] = 95;
```

ii.
```
CourseData list[85];
list[0]->exams[0] = 83;
```

iii.
```
CourseData s1;
scanf("%s", s1.name);
```

iv.
```
typedef struct {
  CourseData students[49];
} SectionData;

SectionData sd;
sd.students[0].exams[1] = 75;
```

1 (cont.)
The following question uses the structure defined below:

```
typedef struct {
  char name[50];
  double progs[8];
  double exams[3];
} CourseData;
```

d. Which of the following choices show valid uses of a pointer to access a field within a variable of type `CourseData`?

A.
```
CourseData cd;
CourseData *p = &cd;
p->exams[2] = 85;
```

B.
```
double exAvg(CourseData *c) {
  return (c->exams[0] + c->exams[1] + c->exams[2]) / 3;
}
```

C.
```
CourseData a[10];
CourseData *p = a;
printf(p->name);
```
<u>Note:</u> This line contained a typo—it originally read
`printf(a->name);`
Given this error, everyone received full credit for this problem.

D.
```
CourseData b[15];
CourseData *p1 = &b[0];
CourseData *p2 = &b[14];
while (p1 != p2) {
  scanf("%lf", &p1->progs[0]);
  p1++;
}
```

i. A and C
ii. A and B
iii. B and C
iv. B and D
v. ***<u>A, B, C, and D</u>***

1 (cont.)

e. Why are structures typically passed by address, not by value, when they are used as function arguments?

i. Structures have to be passed by address—the C language doesn't allow you to copy the entire contents of a structure at once.

ii. Functions that deal with structures always need to be able to change the structure inside the function, which can't be done if the structure is passed by value.

iii. ***<u>Passing a structure by address uses less memory than passing it by value, since structures typically contain multiple fields and therefore use more space than a scalar variable.</u>***

iv. Passing a structure by value would cause a rift in the space-time continuum.

f. Circle one (or more) of the choices below that you feel best "answers" this "question."

i. 12

ii. "Thanks for the free points."

iii. "I don't REALLY have to answer the last two questions, do I?"

iv. "Of all the tests I've taken that started at 6:30 PM today, this one is my favorite."

v. "I don't think it's fair that Dr. Geiger can sit there and play Angry Birds while we're suffering through this test."
(I'm not actually playing Angry Birds … as far as you know.)

vi. ***<u>All, or any, of the above</u>***

2. (40 points) ***Arrays and pointers***

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.

I encourage you to 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. (13 points)

```
int main() {
     int x[10] = {0};
     int *ptr = x;
     int i;

     ptr[2] = 16;
     ptr[1] = 216;
     ptr[6] = 2011;

     ptr = ptr + 5;

     ptr[0] = 12;
     ptr[1] = 20;
     ptr[-2] = 3;

     ptr = ptr - 2;
     ptr[4] = ptr[5] = ptr[6] = 201;

     for (i = 0; i < 10; i++)
          printf("%d ", x[i]);
     printf("\n%d\n", *ptr);
     return 0;
}
```

Solution:

```
0 216 16 3 0 12 20 201 201 201

3
```

2 (cont.)

b. (13 points)

```
int main() {
     char str[] = "Applications";
     char *ptr1, *ptr2;
     char temp;

     ptr1 = str;
     ptr2 = &str[11];

     while (*ptr1 != '\0') {
          temp = *ptr1;
          *ptr1 = *ptr2;
          *ptr2 = temp;
          ptr1 += 4;
          ptr2 -= 3;
     }

     printf("%s\n", str);
     return 0;
}
```

Solution:

```
sppliiatconA
```

2 (cont.)

c. (14 points)

```
int main() {
	int orig[] = {75, 82, 13, 90, 51, 58, -1};
	int scaled[6];
	int *po = orig;
	int *ps = scaled;
	int i;

	while (*po != -1) {
		*ps = *po + 6;
		po++;
		ps++;
	}

	for (i = 0; i < 6; i++) {
		po--;
		ps--;
		printf("%d %d\n", *po, *ps);
	}
	return 0;
}
```

Solution:

```
58 64
51 57
90 96
13 19
82 88
75 81
```

3. (36 points, 18 per part) ***Arrays (1-D, 2-D, and strings) and functions***
For each part of this problem, you are given a function to complete. Note that some of the code is provided for you. **CHOOSE ANY TWO OF THE THREE PARTS** and fill in the spaces with appropriate code. **If you complete all three, I will grade only the first two.**

a. `void countOccurrences(int occ[], int n);`

- Given: an array, `occ[]`, and the array length, `n`.
  - Note: You must correctly initialize `occ[]` before using it.
- Repeatedly prompt for and read integers from standard input and count how many times each value is entered, using `occ[]`—`occ[0]` is the number of times 0 is entered, `occ[1]` is the number of times 1 is entered, and so on.
- Exit the function when the input value is not a valid array index for `occ[]`.
- Example: assume n = 4, and user enters: `0 0 0 1 1 2 2 2 2 2 3 -1`
  - `occ[0] = 3, occ[1] = 2, occ[2] = 5, occ[3] = 1`
  - Prompts (`"Enter integer value: "`) not shown in example to save space

Boldfaced, underlined code was to be filled in by student

```
void countOccurrences(int occ[], int n) {
     int inval;      // Input value
     int i;          // Loop index

     // Initialize array
     for (i = 0; i < n; i++)
          occ[i] = 0;


     while (1) {                // Infinite loop--will exit when
                                //   you reach return statement

          // Prompt for and read integer
          printf("Enter integer value: ");
          scanf("%d", &inval);

          // Invalid array index: any negative value or value
          //   greater than or equal to n
          if ((inval < 0) || (inval >= n)
               return;    // Input value is invalid--exit function

          // If input value is valid, use it as an array index
          //   and increment the appropriate element of occ[]
          else
               occ[inval]++;
     }
}
```

3 (cont.)

b. `void matrixMul(int nR, double d[][3], double m1[][2],`
`double m2[][3]);`

- Given: three 2-D arrays, `d`, `m1`, and `m2`, and one array dimension, `nR`.
    - o Size of `d`: `nR x 3`
    - o Size of `m1`: `nR x 2`
    - o Size of `m2`: `2 x 3`
- Perform the matrix multiplication `m1 * m2` and store the results in `d`.
- Example: In the example below, `nR == 2`:

m1

1	2
3	4

*

m2

5	6	7
8	9	10

=

d

1*5 + 2*8 = 21	1*6 + 2*9 = 24	1*7 + 2*10 = 27
3*5 + 4*8 = 47	3*6 + 4*9 = 54	3*7 + 4*10 = 61

- Hint: The problem may be slightly easier to solve if you take the example above and replace each value with an array name and index—for example, the value 1 is actually `m1[0][0]`—to get a better idea of how to calculate each element of `d`.

Boldfaced, underlined code was to be filled in by student

```
void matrixMul(int nR, double d[][3], double m1[][2], double m2[][3]){
     int i;          // Loop variable for rows of m1 and d
     int j;          // Loop variable for columns of m2 and d

     for (i = 0; i < nR; i++) {
          for (j = 0; j < 3; j++) {
               d[i][j] = m1[i][0] * m2[0][j] + m1[i][1] * m2[1][j];
          }
     }
}
```

3 (cont.)

c. `int removeSub(char str[], int start, int end);`

- Given: a string `str`, and positions within that string, `start` and `end`.
  - You can assume `start` and `end` do not need to be checked for errors.
  - Neither position references the null character at the end of `str`.
- Remove characters from `str` between positions `start` and `end`, including those two positions, and shift the remaining characters so you now have a shorter string.
- Return the number of characters in the new string.
- Examples: Assume you have an array `char s[] = "Test string";`
  - `removeSub(s, 0, 0)` → `returns 10; s = "est string"`
  - `removeSub(s, 0, 4)` → `returns 6; s = "string"`
  - `removeSub(s, 4, 7)` → `returns 7; s = "Testing"`

Boldfaced, underlined code was to be filled in by student

```
int removeSub(char str[], int start, int end) {
   int pos1, pos2;          // Positions within string str[]

   pos1 = start;            // Position to which you’ll start copying
                            //   new characters--you're overwriting
                            //   the part of the string that starts
                            //   here, and keeping everything before
                            //   this point
   pos2 = end + 1;          // Position from which you start copying
                            //   new characters--the characters at the
                            //   end of the string that you’ll keep

   do {
      str[pos1] = str[pos2];
      pos1++;
      pos2++;
   } while (str[pos2-1] != ‘\0’);

   return strlen(str);
}
```