16.216: ECE Application Programming

Summer 2012

Exam 3 Solution

1. (20 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:

```
abc 0 3 4
```

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

i.
```
FILE *fp;
int x = 0, y = 3, z = 4;
char str[10] = "abc";
fp = fopen("output.txt", "r");
scanf("%s %d %d %d", str, &x, &y, &z);
fclose(fp);
```

ii.
```
FILE *fp;
int x = 0, y = 3, z = 4;
char str[10] = "abc";
fp = fopen("output.txt", "w");
fprintf(fp, "%s ", str);
fprintf(fp, "%d %d %d\n", x, y, z);
fclose(fp);
```

iii.
```
FILE *fp;
int x = 0, y = 3, z = 4;
char str[10] = "abc";
fp = fopen("output.txt", "w");
fwrite(str, x, y, z);
fclose(fp);
```

iv.
```
int x = 0, y = 3, z = 4;
char str[10] = "abc";
printf("%s %d %d %d\n", str, x, y, z);
```

1 (cont.)

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

```
double arr[20];
```

Which of the following code snippets would correctly read the contents of this array from a file?

i.
```
FILE *fp = fopen("input.txt", "r");
fscanf(fp, "%lf", arr);
```

ii.
```
FILE *fp = fopen("input.txt", "r");
fread(fp, sizeof(double), 20, arr);
```

iii.
```
FILE *fp = fopen("input.txt", "r");
fread(arr, sizeof(double), 20, fp);
```

iv.
```
FILE *fp = fopen("input.txt", "r");
fwrite(arr, sizeof(double), 20, fp);
```

1 (cont.)

c. You are writing a program that should repeatedly read input characters until a space is entered, then read the space and the rest of the line that follows (up to 49 total characters) into an array:

Which of the following code sequences correctly read this input?

i.
```
char c;
char inp[50];
while ((c = getchar()) != ' ')
     ;    // Do nothing—empty loop
fgets(inp, 50, stdin);
```

ii.
```
char c;
char inp[50];
while ((c = getchar()) != ' ')
     ;    // Do nothing—empty loop
ungetc(c, stdin);
fgets(inp, 50, stdin);
```

iii.
```
char c;
char inp[50];
putchar(c);
fputs(inp, stdout);
```

iv.
```
char c;
char inp[50];
printf("%c %s\n", c, inp);
```

1 (cont.)

d. The following question uses the structure defined below:

```
typedef struct {
   int number;
   char name[40];
   char rating[7];
   int length;
   char time[4][7];
} Movie;
```

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

i.
```
Movie m;
scanf("%s", m->name);
```

ii.
```
typedef struct {
   Movie mList[10];
} TheaterData;

TheaterData td;
td.mList[0].length = 120;
```

iii.
```
Movie m;
m.length = 123;
```

iv.
```
Movie TDKR;
Movie *P = &TDKR;
strcpy(P->name, "The Dark Knight Rises");
```

1 (cont.)

e. Choose your favorite statement(s) from the list below. Circle all that apply (but don't waste too much time!):

i. "I'm so glad I spent the last six weeks taking this course—that's how I've always wanted to spend July and August!"

(You may be more likely to agree with this statement if you imagine saying it with just a hint of sarcasm.)

ii. "The exam ends here, right? We don't REALLY need to do the last two questions … "

iii. "You're going to drop our lowest program score or two … or five … right?"

iv. "I'd actually like to take this opportunity to leave some constructive feedback in the space below" *(Note: don't do this unless you have time left at the end of the exam.)*

All of the above are "correct."

2. (40 points) ***Strings; 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.

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.

a. (12 points)

```
void main() {
     int i, n;
     char *p;
     char str[] = "thisgridhidesone!";
     p = &str[5];
     n = 1;
     for (i = 0; i < 6; i++) {
          printf("%c", *p);
          p = p + n;
          n = n * -2;
     }
}
```

Solution: *This problem uses pointer arithmetic to move a pointer to various spots in this string, then prints the character being pointed to. Initially, n == 1 and p points to str[5], which is the character* `'r'`*. The table below shows the result of each statement in each loop iteration:*

Value of i	`printf("%c", *p);`	`p = p + n;`	`n = n * -2;`
0	Prints str[5] → `'r'`	p = p + 1 → pointer moves to str[6]	n = 1 * -2 = -2
1	Prints str[6] → `'i'`	p = p + (-2) → pointer moves to str[4]	n = -2 * -2 = 4
2	Prints str[4] → `'g'`	p = p + 4 → pointer moves to str[8]	n = 4 * -2 = -8
3	Prints str[8] → `'h'`	p = p + (-8) → pointer moves to str[0]	n = -8 * -2 = 16
4	Prints str[0] → `'t'`	p = p + 16 → pointer moves to str[16]	n = 16 * -2 = -32
5	Prints str[16] → `'!'`	p = p + (-32) → pointer moves to str[-16] *(Note: Pointer is now outside array, but this is the last loop iteration, so this invalid location isn't accessed)*	n = -32 * -2 = 64

Therefore, the final output is: ***`right!`***

2 (cont.)

b. (14 points)

```
void main() {
     char s1[20] = "";
     char s2[20] = "";
     strcat(s1, "ab");              s1 = "ab"
     strcat(s2, "ac");              s2 = "ac"
     printf("%s %s\n", s1, s2);
     strcat(s1, s2);                s1 = "abac"
     strcat(s2, s1);                s2 = "acabac"
     printf("%s %s\n", s1, s2);
     strncat(s1, s2, 3);            s1 = "abacaca"
     strncat(s2, s1, 3);            s2 = "acabacaba"
     printf("%s %s\n", s1, s2);
}
```

Solution: *Changes to s1 and s2 are shown above (with the characters added via concatenation underlined in each step); the final output is*

```
ab ac
abac acabac
abacaca acabacaba
```

2 (cont.)

c. (14 points)

```
void main() {
     char s1[10] = "baaaad";
     char s2[10] = "b";
     int i = 1;

     do {
          printf("%s %s\n", s1, s2);
          strncat(s2, "a", 1);
          i++;
     } while (strncmp(s1, s2, i) == 0);
}
```

Solution: *Each time through the loop, the following actions occur:*

- *The current states of s1 and s2 are printed*
- *An "a" is added to s2*
- *The length of the strings being compared (represented by i) is incremented*
- *The first i characters in s1 and s2 are compared; when they do not match, the loop ends*

Since s1 contains 4 "a"s, s1 and s2 will match until a 5th "a" is added to s2. This happens in the fifth loop iteration, meaning that the program output will be:

```
baaaad b
baaaad ba
baaaad baa
baaaad baaa
baaaad baaaa
```

3. (40 points, 20 per part) ***Arrays and functions***

For each part of this problem, you are given a function to complete. **CHOOSE ANY TWO OF THE THREE PARTS** and fill in the spaces provided with appropriate code. **You may complete all three parts for up to 10 points of extra credit, but must clearly indicate which part is the extra one—I will assume it is part (c) if you mark none of them.**

a. `int checkIfSorted(int a[], int n);`

Given an array of integer values, `a[]`, and the size of the array, `n`, check if the array is sorted from smallest to largest value. If so, return 1; if not, return 0. For example, if:

- `a = {1, 2, 3, 4, 5}` → `checkIfSorted(a, 5)` returns 1
- `a = {1, 2, 4, 3, 5, 7, 8}` → `checkIfSorted(a, 7)` returns 0

Solution:

```
int checkIfSorted(int a[], int n) {
     int i;              // Loop variable

     // GO THROUGH ARRAY—AS SOON AS YOU FIND A PAIR OF
     //   INCORRECTLY SORTED VALUES, RETURN 0
     for (i = 0; i < n-1; i++) {

          // TEST CONDITION THAT SHOWS ARRAY IS NOT SORTED
          //   AND RETURN 0 IF TRUE
          if (a[i] > a[i+1])
               return 0;
     }

     return 1;           // IF FUNCTION REACHES THIS POINT,
                         //   ARRAY MUST BE SORTED
}
```

3 (cont.)

b. `int longestMatch(char *s1, char *s2);`

Given two strings, s1 and s2, return the length of the longest matching character sequence between the two, starting with the first character of each. For example:

- `longestMatch("string", "other")` returns 0
- `longestMatch("string", "stuff")` returns 2
- `longestMatch("string", "strings")` returns 6

```
int longestMatch(char *s1, char *s2) {
     int nC;   // Current string length being tested

     // INITIALIZE VARIABLES IF NEEDED
     nC = 0;

     // KEEP TESTING STRINGS UNTIL MATCH IS NOT FOUND
     //   OR YOU'VE REACHED END OF STRING S1
     while (nC < strlen(s1)) {

          // STRINGS DON'T MATCH—RETURN LENGTH OF LONGEST
          //   MATCHING STRING
          if (strncmp(s1, s2, nC + 1) != 0)
               return nC;

          // UPDATE nC
          nC++;
     }

     // IF YOU REACH THIS POINT, ALL CHARACTERS UP TO LENGTH
     //   OF S1 MATCH—RETURN nC
     return nC;
}
```

3 (cont.)

c. `int countZeroColumns(int arr[][10], int n);`

Given a 2-D integer array, `arr[][10]`, and the number of rows in the array, `n`, return the number of columns in this array in which most of the values are 0. For example, if the array has three rows, you count the number of columns that contain two or more zeroes.

```
int countZeroColumns(int arr[][10], int n) {
     int totalCols = 0;       // Total # of columns with
                              // mostly zero values
     int zeroesPerCol;        // # of zeroes in a given column
     int i, j;

     // GO THROUGH ALL COLUMNS
     for (j = 0; j < 10; j++) {

          // FOR EACH COLUMN, GO THROUGH ALL ROWS AND COUNT
          //    NUMBER OF ZEROES
          zeroesPerCol = 0;
          for (i = 0; i < n; i++) {
               if (arr[i][j] == 0)
                    zeroesPerCol++;
          }

          // AFTER CHECKING ALL ROWS IN A GIVEN COLUMN, CHECK
          //    TO SEE IF THE MAJORITY OF VALUES IN THAT COLUMN
          //    ARE ZERO—IF SO, UPDATE COUNT OF COLUMNS WITH
          //    MOSTLY ZERO VALUES
          if (zeroesPerCol > (n / 2))   // Majority of
               totalCols++;             // column is zeroes
     }

     return totalCols;        // Return count of columns with
                              //   mostly zeroes
}
```