16.216: ECE Application Programming

Fall 2014

Exam 3
December 17, 2014

Name: _____ **ID #:** _____

For this exam, you may use only one 8.5" x 11" double-sided page of notes. All electronic devices (e.g., calculators, cellular phones, PDAs) are prohibited. If you have a cellular phone, please turn it off prior to the start of the exam to avoid distracting other students.

The exam contains 3 questions for a total of 100 points. 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:

- Question 3 has three parts, but you are only required to complete two of the three parts.
  - You may complete all three parts for up to 10 points of extra credit. If you do so, **please clearly indicate which part is the extra one—I will assume it is part (c) if you mark none of them.**
- For each part of Question 3, you must complete a short function. I have provided comments to describe what your function should do and written some of the code for you.
  - Note that each function contains both lines that are partially written (for example, a `printf()` call in which you are responsible for filling in the format string and expressions) and blank spaces in which you must write additional code. **You must write all code required to make each function work as described—do not assume you can simply fill in the blank lines and get full credit.**
  - Each function is accompanied by one or more test cases. Each test case is an example of how the function should behave in one specific case—**it does not cover all possible results of using that function.**
- You can solve each part of Question 3 using only the variables that have been declared, but you may declare and use other variables if you want.

You will have 3 hours to complete this exam.

Q1: Multiple choice	/ 20
Q2: Bitwise operators	/ 40
Q3: File, character, and line I/O	/ 40
TOTAL SCORE	/ 100
EXTRA CREDIT	/ 10

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 one choice you think best answers the question.

a. Consider a program containing the following structure definition and code snippet:

```
typedef struct {
   int a;
   int b;
} myStruct;

void main() {
   myStruct s1 = {3, 4};
   myStruct s2 = {5, 6};
   myStruct *p = &s1;
```

Which of the following statements will set the value of b inside s1 to 6? Note that <u>any</u> statement that makes that change is a correct answer, even if that statement changes other values as well.

A. `s1->b = 6;`

B. `s1.b = 6;`

C. `p->b = 6;`

D. `p.b = 6;`

E. `s1 = s2;`

i. A and D

ii. B and C

iii. A, D, and E

iv. B, C, and E

v. None of the above

1 (continued)

b. Assume that the integer pointer `p` points to a dynamically allocated array of 10 integers. Which of the following statements show the appropriate way to triple the size of this array without losing the original 10 values?

i. `p = (int *)malloc(30 * sizeof(int));`

ii. `p = (int *)calloc(30, sizeof(int));`

iii. `p = (int *)realloc(p, 30 * sizeof(int));`

iv. `p = (int *)realloc(p, 3 * sizeof(p));`

v. None of the above

c. Assume the character array `str[50]` contains a string of unknown length. Which of the following statements correctly allocates exactly enough space to hold this string (with no extra space after the end of the string) in a new array pointed to by the character pointer `sp`? (Note that the initial contents of the new array do not matter, just the amount of space.)

i. `sp = (char *)malloc(50 * sizeof(char));`

ii. `sp = (char *)calloc(strlen(s));`

iii. `sp = (char *)realloc(str, sizeof(str));`

iv. `sp = (char *)malloc(sizeof(char));`

v. `sp = (char *)calloc(strlen(s) + 1);`

1 (continued)

d. Say you have a simple linked list built using the following structure definition for each node:

```
typedef struct node {
     int value;          // Data
     struct node *next;  // Pointer to next node
} LLnode;
```

Assume you have a pointer, `list`, which points to the first element in the linked list. You want to change the list so that it is a circular list—in other words, rather than having the "`next`" pointer in the last node be `NULL`, that pointer will point to the first element.

Which of the following code snippets will correctly modify the existing list so that the "`next`" pointer in the last node points to the first node?

i.
```
LLnode *p = list;
p->next = list->next;
```

ii.
```
LLnode *p = list;
while (p != NULL)
   p = p->next;
p->next = list;
```

iii.
```
list->lastnode = list;
```

iv.
```
LLnode *p = list;
while (p->next != NULL)
   p = p->next;
p->next = list;
```

v. None of the above

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

i. "Thanks for the free points."

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

iii. "This is the best final exam I've taken so far today."

iv. None of the above.

2. (40 points) ***Bitwise operators***

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() {
   unsigned int v1, v2, v3, v4;
   unsigned int x = 0x216;

   v1 = x | 0x123;
   v2 = x ^ 0xFFFFFFFF;
   v3 = (~x) << 4;
   v4 = x & 0xFFFFF012;

   printf("%#x\n", v1);
   printf("%#x\n", v2);
   printf("%#x\n", v3);
   printf("%#x\n", v4);
   printf("%#x\n", x);
}
```

2 (continued)

b. (14 points)

```
void main() {
   unsigned int x = 0xABC;
   unsigned int v1, v2, v3, v4;

   v1 = x << 20;
   v2 = x << 2;
   v3 = (v1 >> 4) << 8;
   v4 = (x >> 3) << 7;

   printf("%X\n", x);
   printf("%#x\n", v1);
   printf("%#.5X\n", v2);
   printf("%.8x\n", v3);
   printf("%#.8X\n", v4);
}
```

2 (continued)

c. (14 points)

```
void main() {
   unsigned int val = 0xDEADBEEF;
   unsigned int v1, v2, v3, v4;

   v1 = val & 0x000000FF;
   v2 = (val & 0x0FFF0000) >> 16;
   v3 = (val & 0x78000000) >> 24;
   v4 = (val & 0x00010100) >> 4;

   printf("%#.8X\n", v1);
   printf("%#.8X\n", v2);
   printf("%#.8X\n", v3);
   printf("%#.8X\n", v4);
}
```

3. (40 points, 20 per part) ***File, character, and line I/O***

For each part of this problem, you are given a short program 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.**

Remember, you must write all code required to make each function work as described—**do not assume you can simply fill in the blank lines and get full credit.** Also, remember that each example provided is only applicable in one specific case—**it does not cover all possible results of using that function.**

a. `int longestLine(FILE *fp, char list[][100], int nS);`

This function will read `nS` lines from an open text file pointed to by `fp` and store those lines in the 2D array `list`. Note that each row of a two-dimensional character array can hold a single string—`list[0]` will hold the first line and `list[nS-1]` will hold the last one.

As it reads the lines, your function will keep track of which line is the longest and will return the row index of that line. For example, if the input file contains the lines `"Test\n"`, `"No\n"`, and `"Longest line\n"`, your function will return 2, since the third line (with index 2) is longest.

```
int longestLine(FILE *fp, char list[][100], int nS) {
   int i;                 // Loop index
   int maxlen = 0;        // Length of longest string
   int maxind = 0;        // Array index of longest string

   // Loop to read nS different lines of input
   for (__________________;__________________;__________________) {




      // After reading line, check to see if it is longer than
      // current max and update variables accordingly









   }

   // Return index of longest line
   return maxind;
}
```

3 (continued)

b. `void readTwoFiles(FILE *intfp, FILE *chfp, FILE *outfp);`

This function takes three file pointers as arguments. The first two (`intfp`, `chfp`) point to open input files that are set up as follows:

- `intfp` points to a text file containing a set of integers: for example, `3 5 2`
- `chfp` points to a text file containing a set of characters: for example, `abc12345xy`

Your function should repeatedly read an integer from the file `intfp` points to until reaching the end of the file. For each integer, read that number of characters from the file `chfp` points to and print them to the file `outfp` points to, which is also already open when the function is called. Each series of characters read should be printed on a separate line.

For example, given the input file contents above, the output file would contain the following:

```
abc
12345
xy
```

```
void readTwoFiles(FILE *intfp, FILE *chfp, FILE *outFP) {
   int inval;             // Integer input
   char ch;               // Character input
   int i;                 // Loop index

   // Loop to repeatedly read an integer from first input file

   while (_____________________________________________________) {



      // After reading integer, read that number of characters
      //  from second input file and print a single line
      //  containing those characters to the output file









   }
}
```

3 (continued)

c. `int readLastTen(double arr[]);`

This function prompts for and opens a binary file, then repeatedly fills the array `arr[]` with ten double-precision values from that file. The goal is to store only the last ten values from the file in the array, so your program should repeatedly read from the file until you know you've read the last values. You may assume the number of values in the file is a multiple of 10.

The function will return 1 if it successfully opens and reads the file, and 0 otherwise.

```
int readLastTen(double arr[]) {
  FILE *fp;             // File pointer
  char name[50];        // File name

  // Prompt for and read file name, then use name to open binary
  //   input file
  printf("Enter file name: ");

  scanf(_____________________________,_____________________________);









  // Only read file if it opens successfully
  if (_____________________________________________________________) {

    // Repeatedly fill array with 10 values from input file
    //   until you've read last 10 values, then close file









    return 1;               // Return 1 to indicate success
  }

  // Return 0 if file opening failed
  else return 0;
}
```