16.216: ECE Application Programming

Spring 2015

Exam 3

May 4, 2015

Name: ____

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: Strings and 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, 35};
   myStruct *p = &s1;
   myStruct list[5];
```

Which of the following choices will copy the values inside `s1` to the first structure in the array `list`?

A. `list[0] = s1;`

B.
```
list[0].a = s1.a;
list[0].b = s1.b;
```

C.
```
list[0].a = s1->a;
list[0].b = s1->b;
```

D.
```
list[0].a = p.a;
list[0].b = p.b;
```

E.
```
list[0].a = p->a;
list[0].b = p->b;
```

i. Only A

ii. C and D

iii. B and E

iv. A, C, and D

v. A, B, and E

1 (continued)

b. Assume that the double pointer `p` points to a dynamically allocated array containing `n` elements. Which of the following statements could have been used to create this array?

A. `p = (double *)malloc(n * sizeof(double));`

B. `p = (double *)realloc(p, n * sizeof(int));`

C. `p = (double *)malloc(sizeof(n));`

D. `p = (double *)calloc(n, sizeof(double));`

i. Only A

ii. A and B

iii. C and D

iv. B and C

v. A and D

c. Which of the following choices dynamically allocates a two-dimensional array with 20 rows that contains 100 integers in total?

i.
```
p = (int *)malloc(100 * sizeof(int));
```

ii.
```
p = (int *)calloc(20 * sizeof(int));
```

iii.
```
p = (int **)malloc(5 * sizeof(int *));
for (i = 0; i < 5; i++)
     p[i] = (int *)malloc(20 * sizeof(int));
```

iv.
```
p = (int **)malloc(20 * sizeof(int *));
for (i = 0; i < 20; i++)
     p[i] = (int *)malloc(5 * sizeof(int));
```

v. None of the above

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 node in the linked list. You are also given two other variables, which are declared as follows:

```
LLnode *p;          // General node pointer
int n = 0;          // Counter
```

Which of the following code snippets will set `n` equal to the total number of nodes in the list?

i.
```
p = list;
n = sizeof(p);
```

ii.
```
p = list;
while (p == NULL) {
   p = p->next;
   n++;
}
```

iii.
```
p = list;
while (p != NULL) {
   n += p->val;
   p = p->next;
}
```

iv.
```
p = list;
while (p != NULL) {
   p = p->next;
   n++;
}
```

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. "It's nice to finally have an exam that isn't at 8:00 AM."

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)

```
int main() {
    unsigned int v1, v2, v3, v4;
    unsigned int x = 0x1108;

    v1 = x | 0xACED;
    v2 = (~x) & 0xFF0096AA;
    v3 = x ^ 0xF0F0F0F0;
    v4 = x << 8;

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

    return 0;
}
```

2 (continued)
b. (14 points)

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

    v1 = x << 12;
    v2 = x >> 3;
    v3 = (x << 24) >> 4;
    v4 = (v1 >> 16) << 6;

    printf("%x\n", x);
    printf("%.8x\n", v1);
    printf("%#x\n", v2);
    printf("%#.8x\n", v3);
    printf("%#.4x\n", v4);

    return 0;
}
```

2 (continued)

c. (14 points)

```
int main() {
    unsigned int x = 0xBADCAB15;
    unsigned int v1, v2, v3, v4;

    v1 = x & 0x3CF;
    v2 = (x & 0xFFF00) >> 8;
    v3 = (x & 0x808) >> 3;
    v4 = (x >> 16) & 0xF6;

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

    return 0;
}
```

3. (40 points, 20 per part) <u>***Strings and I/O***</u>

For each part of this problem, you are given a short program to complete. **<u>CHOOSE ANY TWO OF THE THREE PARTS</u>** and fill in the spaces provided with appropriate code. **<u>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.</u>**

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

a. `void makeLine(char *s, FILE *binfp, FILE *txtfp);`

This function will read two files: an open binary file pointed to by `binfp` that contains a series of integers, and an open text file pointed to by `txtfp` that contains several lines.

The integers in the binary file represent the lengths of each line in the text file. This function should use that information to read all of the lines from the text file and concatenate them together into a single string stored in `s`.

Note that the line lengths include the `'\n'` character at the end of each line. Also, when concatenating lines together, include a space between each line.

For example, given a text file containing `"Line 1\n"`, `"is a\n"`, and `"string.\n"`, the corresponding binary file would hold the values 7, 5, and 8, and the string `s` would be set to `"Line 1 is a string."`

```
void makeLine(char *s, FILE *binfp, FILE *txtfp) {
   int n;           // Size of input line
   char ins[100];   // Input line

   // Read an integer from the binary file until there are no
   //   integers left to read

   while (___________________________________________________) {

      // Read a line from the text file and concatenate it to
      //   the string s, making sure there is a space between
      //   each pair of concatenated lines









   }
}
```

3 (continued)

b. `int countOcc(char *fname, char *s);`

This function takes two strings as arguments: the name of a text input file (`fname`) and a string to be matched (`s`). The function should open the text file, repeatedly read a string from that file, and return the number of times that the first characters of each input string match the string `s`. For example, given an input file `in.txt` containing `"string"`, `"straw"`, and `"sand"`:

- `countOcc("in.txt", "s")` would return 3
- `countOcc("in.txt", "str")` would return 2
- `countOcc("in.txt", "strings")` would return 0

The function should return 0 if the input file cannot be opened.

```
int countOcc(char *fname, char *s) {
   char inp[50];         // Input string
   int n;                // Number of occurrences of s
   FILE *fp;             // File pointer

   // Initialize variables as needed



   // Open file; return 0 if file cannot be opened



   if (_________________________________________________________)
      return 0;

   // Repeatedly read string from file and update n if
   //    beginning of string matches s

   while (____________________________________________________) {







   }
   // Return number of occurrences of s

}
```

3 (continued)
c. `unsigned int readHex(FILE *fp);`

This function reads a hexadecimal integer from the open text file referenced by `fp`, converts it to decimal, and returns that value. The algorithm is similar to the code below, which converts a series of digits to their integer equivalent:

```
   while (isdigit(c = getchar()))    n = n * 10 + (c - 48);
```

`n` is the running total of all digits read thus far, `c` is the input character, and 48 is the ASCII value of `'0'`. The major differences you'll need to account for in this function are as follows:

- The hexadecimal integer always contains a leading `0x` and 8 digits (i.e., `0x00123456`).
- Each digit can be a letter or a number, and letters can be uppercase or lowercase. If `ch` is a letter, `tolower(ch)` returns the letter in lowercase, while `toupper(ch)` returns it in uppercase. The ASCII values of `'A'` and `'a'` are 65 and 97, respectively.
- The significance of each digit is a multiple of 16, not 10.

```
unsigned int readHex(FILE *fp) {
   char c;                      // Input character
   unsigned int tot = 0;        // Running total
   int i;                       // Loop index

   // Read (and ignore) leading 0x



   // Loop to read 8 digits, convert each one to appropriate
   //   integer value, and update total accordingly

   ___________ (______________________________________________) {


      if (____________________________) {     // Digit is a number


      }
      else {                                  // Digit is a letter



      }
      tot =                                   // Update total
   }
   return tot;                  // Return total
}
```