16.216: ECE Application Programming

Fall 2015

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 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 = {2, 31};
   myStruct *p = &s1;
   myStruct list[10];
```

Which of the following code snippets will copy the values inside `s1` to every structure in the array `list`?

A.
```
list = s1;
```

B.
```
list = p;
```

C.
```
for (i = 0; i < 10; i++)
   list[i] = s1;
```

D.
```
for (i = 0; i < 10; i++) {
   list[i].a = s1.a;
   list[i].b = p->b;
}
```

E.
```
for (i = 0; i < 10; i++) {
   list[i]->a = s1.a;
   list[i]->b = p->b;
}
```

i. Only A

ii. Only B

iii. C and D

iv. B, C, and D

v. A 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 shows the appropriate way to shrink this array to half of its original size while keeping the first `n/2` values intact?

A.
```
p = (double *)malloc(n/2 * sizeof(double));
```

B.
```
p = (double *)realloc(p, n/2 * sizeof(double));
```

C.
```
p = (double *)malloc(sizeof(n/2));
```

D.
```
p = (double *)calloc(n/2, sizeof(double));
```

i. Only A

ii. Only B

iii. C and D

iv. A and B

v. A, B, and D

c. Which of the following choices dynamically allocates an array of ten strings, each of which can contain up to 50 characters?

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

ii.
```
p = (char *)calloc(10 * sizeof(char *));
```

iii.
```
p = (char **)malloc(10 * sizeof(char *));
for (i = 0; i < 10; i++)
     p[i] = (char *)malloc(50);
```

iv.
```
p = (char **)malloc(50 * sizeof(char *));
for (i = 0; i < 50; i++)
     p[i] = (char *)malloc(10);
```

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 also have two other pointers: `newNode`, which points to a newly allocated node, and `p`, which is uninitialized. Which of the following code snippets adds the newly allocated node to the very end of the list (i.e., as the last element)?

i.
```
list->next = newNode;
```

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

iii.
```
p = list;
while (p->next != NULL)
   p = p->next;
p->next = newNode;
newNode->next = NULL;
```

iv.
```
p = list;
while (p == NULL)
   p = p->next;
p->next = newNode;
newNode->next = NULL;
```

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 tonight."

iv. None of the above.

All of the above are "correct."

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 = 0x218A;

   v1 = x & 0x7E93;
   v2 = (~x) | 0x13F5;
   v3 = x ^ 0xFF00FF;
   v4 = x << 20;

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

   return 0;
}
```

OUTPUT:

0x2082
0xffffdff5
0xff2175
0x18a00000
0x218a

2 (continued)
b. (14 points)

```
int main() {
   unsigned int v1, v2, v3, v4;
   unsigned int x = 0xA11A;

   v1 = x << 4;
   v2 = x >> 5;
   v3 = (x << 20) >> 8;
   v4 = (v1 >> 10) << 12;

   printf("%x\n", x);
   printf("%.6x\n", v1);
   printf("%#x\n", v2);
   printf("%#.8x\n", v3);
   printf("%#.7x\n", v4);

   return 0;
}
```

OUTPUT:

```
a11a
0a11a0
0x508
0x0011a000
0x0284000
```

2 (continued)

c. (14 points)

```
int main() {
   unsigned int x = 0xABACAB81;
   unsigned int v1, v2, v3, v4;

   v1 = x & 0xF0F;
   v2 = (x & 0xFF00000) >> 20;
   v3 = (x & 0x7FC) >> 2;
   v4 = (x >> 20) & 0x1FE;

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

   return 0;
}
```

OUTPUT:

0x00000b01
0x000000ba
0x000000e0
0x000000ba

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 simply fill in the blank lines for 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. `void readArr(int arr[], FILE *posfp, FILE *arrfp);`

This function reads two files: an open binary file pointed to by `posfp`, and an open text file pointed to by `arrfp`. The binary file holds ten integers, sorted from low to high, representing positions within the text file. Values in those positions should be stored in the array `arr[]`. The first position is position 0.

For example, if the binary file holds `{0,2,4,6,8,10,12,14,16,18}`, the function should store every other value from the text file, starting with the first one—if the first values in the text file are `{5,10,15,20,25,30 …}`, 5, 15, and 25 are the first three values stored in `arr[]`.

```
void readArr(int arr[], FILE *posfp, FILE *arrfp) {
   int pos[10];      // List of positions
   int i, j;         // Array/loop indexes
   int inp;          // Input value

   // Read pos array from binary file pointed to by posfp
   fread(pos, sizeof(int), 10, posfp);

   // Repeatedly read value from file pointed to by arrfp
   // Fill arr with values in positions stored in pos array
   i = 0;
   for (j = 0; j < pos[9]; j++) {
      fscanf(arrfp, "%d", &inp);

      // If index matches current value from pos array, store
      //  input value in arr[] and increment array index
      if (j == pos[i]) {
         arr[i] = inp;
         i++;
      }
   }
}
```

3 (continued)
b. `int numStrings(FILE *fp);`

This function reads lines from an open text file pointed to by `fp` and returns the number of lines that could be read as strings using `scanf()`—in other words, the number of lines with no space (`' '`) or tab (`'\t'`) characters. For example, given a file containing `"This\n"`, `"is a\n"`, and `"string.\n"`, the function returns 2. Assume the maximum line length is 99 characters.

The function should read lines until it reaches the end of the file.

```
int numStrings(FILE *fp) {
   char buf[100];          // Input buffer
   int i;                  // Loop index
   int ns;                 // Number of strings

   // Initialize variables as needed
   ns = 0;

   // Read input lines until end of file is reached
   do {
      fgets(buf, 100, fp);

      // Check all input characters except last one; exit loop if
      //   tab or space is found. Update counter appropriately
      for (i = 0; i < strlen[buf] - 1; i++) {
         if (buf[i] == ' ' || buf[i] == '\t') {
            ns--;
            break;
         }
      }
      ns++;
   } while (!feof(fp));


   return ns - 1;
}
```

3 (continued)

c. `void writeOnlyLetters(FILE *infp, FILE *outfp);`

This function takes two file pointers as arguments. `infp` points to a text file that has been opened using mode `"r"`; `outfp` points to a text file that has been opened in mode `"w"`. The input file contains a single integer followed by a series of characters starting with a letter. The integer determines the maximum number of characters to be read from the input file.

After reading the integer from the input file, the function should read at most that number of characters from the same file, printing only the letters to the output file. For example, if the input file contains the characters `20ab234cde567fghi098jklmn908`, the output file would read `abcdefghijklmn`

Note that it is possible for the file to contain fewer than the maximum number of characters after the integer, in which case the function will reach the end of the file while reading characters.

```
void writeOnlyLetters(FILE *infp, FILE *outfp) {
    char ch;   // Input character
    int n;      // Total number of characters to read

    // Read value of n from input file
    fscanf(infp, "%d", &n);

    // Read at most n characters from input file
    while (n > 0) {
        ch = fgetc(infp);

        // Reached end of file--end loop
        if (ch == EOF) {      // if(feof(infp)) would also work
            break;
        }

        // Input character is letter--print to output file
        else if (isalpha(ch)) {
            fputc(ch, outfp);
        }

        n--;
    }
}
```