EECE.2160: ECE Application Programming

Spring 2018

Exam 3 Solution

1. (13 points) ***Strings***

Show the output of the short program below 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 I can easily recognize your final answer.

```
int main() {
   char str1[20];
   char str2[30];
   int n;

   strcpy(str1, "EECE.2160");
   strncpy(str2, "Spring 2018 Section 201", 11);
   str2[11] = '\0';

   printf("%s %s\n", str1, str2);

   n = strlen(str1);
   printf("str2[%d] = %c\n", n, str2[n]);

   strncat(str2, str1, 4);
   strncat(str1, str2, 4);
   printf("%s\n%s\n", str1, str2);

   return 0;
}
```

Solution:

```
EECE.2160 Spring 2018
str2[9] = 1
EECE.2160Spri
Spring 2018EECE
```

2. (33 points) ***Structures***

a. (13 points) Show the output of the short program below 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 I can easily recognize your final answer.

The `Res` and `ResPair` structure definitions, as well as the `setRes()` function definition, are in the extra document provided with the exam.

```
int main() {
   Res res1 = {10, 2, 5};
   ResPair rp;

   setRes(&rp.r1, res1.r, 2, res1.r / res1.i);
   setRes(&rp.r2, res1.v * rp.r1.r, res1.v, rp.r1.r);

   printf("res1: %.2lf %.2lf %.2lf\n", res1.v, res1.i, res1.r);
   printf("rp.r1: %.2lf %.2lf %.2lf\n",
          rp.r1.v, rp.r1.i, rp.r1.r);
   printf("rp.r2: %.2lf %.2lf %.2lf\n",
          rp.r2.v, rp.r2.i, rp.r2.r);

   return 0;
}
```

Solution:

```
res1: 10.00 2.00 5.00
rp.r1: 5.00 2.00 2.50
rp.r2: 25.00 10.00 2.50
```

2 (continued)

b. (20 points) Complete the function below (the `Box` structure is defined on the extra sheet):

```
int maxVol(Box list[], int n);
```

This function takes as arguments an array of `Box` structures, `list`, as well as the number of structures in the list, `n`. The function returns the index of the structure representing the box with the greatest volume. For example, given:

```
Box arr[4] = { {1, 3, 5}, {9, 9, 9}, {4, 3, 2}, {2, 2, 2} };
```

the function call `maxVol(arr, 4)` would return 1, as `arr[1]` has the greatest volume (729, which is greater than the volumes of `arr[0]` (volume 15), `arr[2]` (24), and `arr[3]` (8)).

```
int maxVol(Box list[], int n) {
   int i;            // Loop index
   int maxI;         // Index of largest prism
   double maxVol;    // Volume of largest prism

   // Initialize variables as needed
   maxI = 0;
   maxVol = list[0].W * list[0].L * list[0].H;


   // Go through list; update max variables if larger box found
   for (i = 1; i < n; i++) {
      if (list[i].W * list[i].L * list[i].H > maxVol) {
           maxVol = list[i].W * list[i].L * list[i].H;
           maxI = i;
      }
   }

   // Return index of box with greatest volume
   return maxI;
}
```

3. (20 points) ***File I/O***

Complete the function described below:

```
int *readBin(char *fname);
```

This function dynamically allocates and returns the address of an array that stores the contents of a binary input file with name `fname`.

The file is formatted so that the first integer value in the file is the number of remaining values; in other words, a file intended to store the six integers 0, 1, 2, 3, 4, and 5 would actually contain the values 6, 0, 1, 2, 3, 4, and 5.

If the file cannot be opened or the space for the array cannot be allocated, the function should return NULL.

```
int *readBin(char *fname) {
   FILE *fp;         // File pointer
   int *arr;         // Pointer to array
   int n;            // Array size

   // Open binary input file and check that it opens correctly
   fp = fopen(fname, "rb");

   if (fp == NULL) {    // fopen unsuccessful
      printf("Could not open file\n");
      return NULL;
   }

   // Read first value from file, which gives size of array, and
   // dynamically allocate array. Return NULL if allocation fails
   fread(&n, sizeof(int), 1, fp);
   arr = (int *)malloc(n * sizeof(int));

   if (arr == NULL) { // Allocate unsuccessful
      printf("Could not allocate array\n");
      return NULL;
   }

   // Read remainder of file into array and return array address
   fread(arr, sizeof(int), n, fp);
   return arr;
}
```

4. (14 points) **Bitwise operators**

Show the output of the short program below 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 I can easily recognize your final answer.

```
void main() {
   unsigned int x = 0x0000019F;
   unsigned int v1, v2, v3, v4;

   v1 = x & 0x00000FF0;
   v2 = x | 0x00000660;
   v3 = x ^ 0x11111111;
   v4 = x & ~(0x0000000F << 4);
   printf("%x\n", x);
   printf("%X\n", v1);
   printf("%.5x\n", v2);
   printf("%#X\n", v3);
   printf("%#.8x\n", v4);
}
```

Solution:

```
19f
190
007ff
0X1111108E
0x0000010f
```

5. (20 points, 4 points each) ***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 the following variables:

```
double *d;      // Dynamically allocated double array
int n;          // Current size of array
```

If `d` points to a dynamically allocated array holding `n doubles`, which of the following statements will reallocate space for this array so that the size of the array is divided in half without changing the remaining elements? In other words, if the array currently holds 10 elements, it will be shrunk to hold only 5 elements; those 5 elements will remain unchanged from the original array values.

i. `d = d / 2;`

ii. `d = (double *)malloc(n / 2 * sizeof(int));`

iii. `d = (double *)calloc(n / 2, sizeof(int));`

iv. `d = (double *)realloc(d, n / 2 * sizeof(int));`

v. `d = int [n/2];`

Technically, the answer should be "none of the above," thanks to the fact that all allocation function calls use "sizeof(int)", not "sizeof(double)". I therefore gave full credit for choices (ii), (iii), and (iv).

Had the problem been written properly, the answer would have been iv—you're reallocating the array, which none of the other choices do.

5 (continued)

b. Which of the following code sequences show an example of a memory leak?

A.
```
int *x = (int *)malloc(10 * sizeof(int));
int *y = x;
free(y);
```

B.
```
int *p1 = (int *)malloc(10 * sizeof(int));
int *p2 = (int *)malloc(10 * sizeof(int));
p1 = p2;
```

C.
```
int *x = (int *)malloc(10 * sizeof(int));
free(x);
x = NULL;
```

D.
```
int *p1 = (int *)malloc(10 * sizeof(int));
int *p2 = (int *)malloc(10 * sizeof(int));
free(p1);
p1 = p2;
```

i. Only A

ii. Only B

iii. A and C

iv. B and D

v. All of the above (A, B, C, and D)

5 (continued)

c. You are writing a program that should repeatedly read input characters until a space is entered, then read 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);
```

5 (continued)

d. Which of the following code snippets would flip the 8 highest bits (bits 24-31) and 8 lowest bits (bits 0-7) of an unsigned integer, x, without changing the middle 16 bits?

i. `x = x | 0xFF0000FF;`

ii. `x = x & 0x00FFFF00;`

iii. ***`x = x ^ 0xFF0000FF;`***

iv. `x = x ^ 0x00FFFF00;`

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

iii. "At least we're not here at 8:00 in the morning."

iv. None of the above.

All of the above are "correct."

6. (10 points) ***EXTRA CREDIT***

Everyone may attempt this problem, **even if you have not at least partially solved the rest of the exam.** Remember, you can earn partial credit for a partial solution to this problem.

Show the output of the short program below exactly as it will appear on the screen. Be sure to clearly indicate spaces between characters when necessary. *(Hint: The final output is easily readable, but you must show at least some work to get credit.)*

A definition for the `XCstruct` structure, as well as more hints, are on the extra handout.

```
int main() {
   XCstruct list[] = {  {4, 0x7AFE,    {6} },
                        {1, 0xA,       {0} },
                        {5, 0x6FEAF,   {2, 4} },
                        {6, 0xFFFFEF,  {3, 5, -3, -3, 2} } };
   char buf[25];
   int i, j, k;
   unsigned m, n, p;

   k = 0;
   for (i = 0; i < 4; i++) {
      m = 0x0000000F << (4 * (list[i].nch - 1));
      n = 4 * (list[i].nch - 1);
      p = 0;

      for (j = 0; j < list[i].nch; j++) {
         buf[k] = (list[i].val & m) >> n;

         if (buf[k] > 9 && buf[k] < 15)
            buf[k] += 87;
         else if (buf[k] == 15)
            buf[k] += 97 + list[i].add[p++];
         else
            buf[k] += 97;

         k++;
         n -= 4;
         m = m >> 4;
      }
      buf[k++] = ' ';
   }
   buf[0] -= 32;
   buf[k-1]++;
   buf[k] = '\0';
   printf("%s\n", buf);
   return 0;
}
```

6 (continued)
Solution: Each structure in the array will essentially be transformed into a word to be printed, as the contents of each structure represent the number of digits in the "word" (`nch`), the "word" itself (`val`), and a set of values used to transform some of the digits into characters (`add[]`).

The outer loop goes through all structures, while the inner loop goes through each digit in the "word." `m` is a bit mask to be used to isolate each digit from the "word," `n` is the amount by which the digit is shifted to move its least significant bit to bit position 0, and p is an index into each structure's `add[]` array. Each digit in the word is stored as a letter in the `buf[]` array:

- If the digit is between 0x0 and 0x9 (the else case in the inner loop), it could represent a letter between 'a' and 'j'—the digit is added to the ASCII value of 'a' (97).
    - This case only occurs twice in the program, as shown below
- If the digit is between 0xa and 0xe, it becomes that letter ('a', 'b', 'c', 'd', or 'e')
    - For example, 0xa + 87 = 97 = 'a'
- If the digit is 0xf, it could represent a letter between 'k' and 'z'. To transform it appropriately, the digit is added to the ASCII value of 'a' (97) and a value chosen from the add[] array stored in the structure.
    - For example, to get the letter 'v' (ASCII value 118), we add 15 + 97 + 6

After each structure is processed, a space is placed between the words in the `buf[]` array to create a readable sentence.

When both loops are done, the following changes are made to the `buf[]` array:

- `buf[0] -= 32` converts the first letter to uppercase ('h' to 'H')
- `buf[k-1]++` adds 1 to the value of the last character, changing that space (ASCII value 32) to an exclamation point (ASCII value 33)
- `buf[k] = 0` null-terminates the array so it can be printed as a string.

The table on the next page shows how each digit from the structures is converted to a letter stored in the `buf[]` array, as well as how the spaces are stored and character changes are made at the end. As you can see, the final output will be:

```
Have a great summer!
```

i	j	k	m	n	p	Digit	buf[k]	Notes
0								list[0].val = 0x7AFE
0	0	0	0xF000	12	0	0x7	7 + 97 = 104 = ‘h’	Changed to ‘H’ at end
0	1	1	0xF00	8	0	0xA	10 + 87 = 97 = ‘a’	
0	2	2	0xF0	4	0	0xF	15 + 97 + 6 = 118 = ‘v’	
0	3	3	0xF	0	1	0xE	14 + 87 = 101 = ‘e’	
		4					= ‘ ‘	buf[k++] = ‘ ‘
1								list[1].val = 0xA
1	0	5	0xF	0	0	0xA	10 + 87 = 97 = ‘a’	
		6					= ‘ ‘	buf[k++] = ‘ ‘
2								list[2].val = 0x6FEAF
2	0	7	0xF0000	16	0	0x6	6 + 97 = 103 = ‘g’	
2	1	8	0xF000	12	0	0xF	15 + 97 + 2 = 114 = ‘r’	
2	2	9	0xF00	8	1	0xE	14 + 87 = 101 = ‘e’	
2	3	10	0xF0	4	1	0xA	10 + 87 = 97 = ‘a’	
2	4	11	0xF	0	1	0xF	15 + 97 + 4 = 116 = ‘t’	
		12					= ‘ ‘	buf[k++] = ‘ ‘
3								list[3].val = 0xFFFFEF
3	0	13	0xF00000	20	0	0xF	15 + 97 + 3 = 115 = ‘s’	
3	1	14	0xF0000	16	1	0xF	15 + 97 + 5 = 117 = ‘u’	
3	2	15	0xF000	12	2	0xF	15 + 97 – 3 = 109 = ‘m’	
3	3	16	0xF00	8	3	0xF	15 + 97 – 3 = 109 = ‘m’	
3	4	17	0xF0	4	4	0xE	14 + 87 = 101 = ‘e’	
3	5	18	0xF	0	4	0xF	15 + 97 + 2 = 114 = ‘r’	
		19					= ‘ ‘	buf[k++] = ‘ ‘ Changed to ‘!’ at end
		20					= ‘\0’	buf[k] = ‘\0’