16.216: ECE Application Programming

Spring 2014

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

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

void main() {
   myStruct m;
   myStruct *p = &m;
```

Which of the following are valid statements (i.e., statements that will compile and execute correctly) that change the value of `a` inside the structure `m`?

A. `myStruct.a = 13;`

B. `m.a = 17;`

C. `m->a = 12;`

D. `p.a = 11;`

E. `p->a = 13;`

i. A and B

ii. B and D

iii. B and E

iv. C and D

v. A, C, and D

1 (continued)

b. Which of the following statements are valid statements that dynamically allocate an array of 35 double-precision values?

A. `double *p = (double *)malloc(35);`

B. `double *p = (double *)calloc(35);`

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

D. `double *p = (double *)malloc(35 * sizeof(double));`

E. `double *p = (double *)calloc(35, sizeof(double));`

i. A and B

ii. A, B, and C

iii. Only D

iv. Only E

v. ***D and E***

1 (continued)

c. The code below dynamically allocates space for a two-dimensional array:

```
double **arr;
arr = (double **) malloc(5 * sizeof(double *));
for (i = 0; i < 5; i++)
   arr[i] = (double *) malloc(10 * sizeof(double));
```

How many values are stored in this array, and how are they organized?

i. 25 values, organized as an array with 5 rows and 5 columns.

ii. 50 values, organized as an array with 10 rows and 5 columns.

iii. <u>50 values, organized as an array with 5 rows and 10 columns.</u>

iv. 100 values, organized as an array with 10 rows and 10 columns.

v. 216 values, and they aren't organized at all.

1 (continued)

d. Say you have a 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. Which of the following code snippets will allow you to determine the total number of elements stored in the linked list and store that number in a variable, `n`?

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

ii.
```
n = sizeof(list) / sizeof(LLnode);
```

iii.
```
LLnode *p = list;
n = 0;
while (p != NULL) {
   n++;
}
```

iv.
```
LLnode *p;
n = 0;
for (p = list; p != NULL; p = p->next) {
   n++;
}
```

v. None of the above

1 (continued)

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 today."

iv. None of the above.

Any 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. (14 points)

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

v1 = 0x0000019F & 0x00000FF0
= 0x00000190

v2 = 0x0000019F | 0x00000660
= 0x000007FF

v3 = 0x0000019F ^ 0x11111111
= 0x1111108E

v4 = 0x0000019F & ~0x000000F0
= 0x0000019F & 0xFFFFFF0F
= 0x0000010F

OUTPUT: Note: all values are printed with capital letters and a leading 0x, but no leading 0s.

```
0X19F
0X190
0X7FF
0X1111108E
0X10F
```

2 (continued)
b. (14 points)

```
void main() {
   unsigned int x = 0x3C;
   unsigned int v1, v2, v3, v4;

   v1 = (x << 8);                    v1 = 0x3C << 8 = 0x3C00
   v2 = (x >> 3);                    v2 = 0x3C >> 3 = 0x7
   v3 = (v2 << 12);                  v3 = 0x7 << 12 = 0x7000
   v4 = (v1 << 2) >> 4;              v4 = (0x3C00 << 2) >> 4
                                        = 0xF000 >> 4 = 0xF00

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

OUTPUT: Note that:

- The first output value is printed using lowercase letters (no other formatting)
- The second output value is printed using uppercase letters (no other formatting)
- The third output value is printed using lowercase letters and a leading 0x
- The fourth output value is printed using 6 digits
- The fifth output value is printed using 8 digits and a leading 0x.

```
3c
3C00
0x7
007000
0x00000f00
```

2 (continued)
c. (12 points)

```
void main() {
   unsigned int x = 0x12345678;
   unsigned int v1, v2, v3, v4;

   v1 = x & 0xFFFF;                  v1 = 0x5678
   v2 = (x & 0xFFFF0000) >> 16;      v2 = 0x12340000 >> 16
                                        = 0x1234
   v3 = (x & 0xF0F0) >> 4;           v3 = 0x5070 >> 4 = 0x507
   v4 = (x & 0x7E000) >> 12;         v4 = 0x44000 >> 12 = 0x44

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

OUTPUT: All output values are printed with a leading 0x, but no leading zeros.

```
0x5678 0x1234 0x507 0x44
```

3. (40 points, 20 per part) ***Bitwise operators***

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.**

a. `char *myfgets(char *s, int n, FILE *stream);`

This function behaves exactly like the C library function `fgets()`: It reads a line of input containing up to `(n-1)` characters from the file pointed to by `stream` and stores those characters in the array `s`. The function then returns the address of that array. Please note that:

- If the function encounters a newline (`'\n'`) within the first `(n-1)` characters, it stores that newline in the array and stops reading at that point.
- The function always adds a null terminator (`'\0'`) to the end of the array.
- Your solution must read a single character at a time.

Students were responsible for writing bold, italicized, underlined code:

```
char *myfgets(char *s, int n, FILE *stream) {
  int i;       // Index into s

  // Initialize variables (Unnecessary if using for loop)

  // Loop to read at most (n - 1) characters, exiting early if
  // you read '\n' (Loop type left blank to allow for or while)

  for (i = 0; i < (n - 1); i++) {
    s[i] = fgetc(stream);
    if (s[i] == '\n') {
      i++;
      break;
    }
  }

  // Add null terminator to end of array and return its address
  s[i] = 0;
  return s;
}
```

3 (continued)

b. `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.

Students were responsible for writing bold, italicized, underlined code:

```
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 its address
   fread(arr, sizeof(int), n, fp);
   return arr;
}
```

3 (continued)

c. `void printUint(unsigned int n, FILE *stream);`

This function essentially converts an unsigned integer, `n`, to the characters representing all of its digits and prints each character to a file pointed to by `stream`, using `fputc()`. Note that:

- The general algorithm: find the number of digits in `n`, then isolate each digit (starting with the leftmost digit), convert it to a character, and output that character.
- Hints:
  - Integer division can help you isolate each digit—for example, 12 / 10 = 1, 352 / 100 = 3, and 9999 / 1000 = 9.
  - The ASCII value of '0' is 48, which may help you convert a digit to a character.

```
void printUint(unsigned int n, FILE *stream) {
   unsigned int div;     // Value used to find # of digits

   // Initialize variables as needed
   div = 1;

   // Special case: if n is 0, just print 0 and end function
   if (n == 0) {
      fputc('0', stream);
      return;
   }

   // Find # of digits by trying to divide n by increasing powers
   // of 10 (1, 10, 100, etc.) until result is 0. Don't change n!

   while (n / div) { // Implicitly tests that result is nonzero
      div = div * 10;
   }

   // Isolate each digit (see examples above) and print each
   //   character using fputc(). Loop should run until you have
   //   printed all digits. Note: you can change n in this loop.
   do {
      div = div / 10;

      fputc(n / div + '0', stream);

      n = n % div;

   } while (div > 1);
}
```