16.216: ECE Application Programming

Fall 2013

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. Which of the following statements dynamically allocates a 100-element array of integers in which all values are initialized to 0?

i. `int *p = int [100];`

ii. `int *p = (int *)malloc(100);`

iii. `int *p = (int *)calloc(100);`

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

v. ***`int *p = (int *)calloc(100, sizeof(int));`***

b. 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];`

1 (continued)

c. 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. <u>Only B</u>

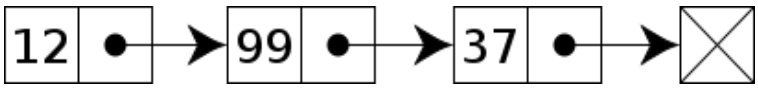
iii. A and C

iv. B and D

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

1 (continued)

d. Say you have a linked list in the state shown below:



Recall that each node uses the following structure definition:

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

Assume you have two pointers: `list`, which points to the first node in the list, and `newNode`, which points to a newly allocated node. Say you want to insert the newly allocated node between the first and second nodes in the list (in the picture above, between the nodes holding 12 and 99). Which of the following choices shows <u>all</u> pointer modifications in the correct order required to make this change?

i.
```
list = newNode;
```

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

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

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

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

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.

<u>All of the above are "correct."</u>

2. (40 points) ***Structures***

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)

```
typedef struct {
   double v;
   double i;
   double r;
} Res;

void main() {
   Res r1 = {10, 2, 5};
   Res r2, r3;

   r2.i = 2;
   r2.v = r1.r;                 r2.v = r1.r = 5
   r2.r = r2.v / r2.i;          r2.r = r2.v / r2.i = 5 / 2 = 2.5
      → After these 3 statements, r2 = {5, 2, 2.5}

   r3 = r2;                     r3 = r2 = {5, 2, 2.5}
   r3.i = r1.v;                 r3.i = r1.v = 10
   r3.v = r3.i * r3.r;          r3.v = r3.i * r3.r = 10 * 2.5 = 25
      → After these 3 statements, r3 = {25, 10, 2.5}

   printf("R1: %.2lf %.2lf %.2lf\n", r1.v, r1.i, r1.r);
   printf("R2: %.2lf %.2lf %.2lf\n", r2.v, r2.i, r2.r);
   printf("R3: %.2lf %.2lf %.2lf\n", r3.v, r3.i, r3.r);
}
```

OUTPUT:

```
R1: 10.00 2.00 5.00
R2: 5.00 2.00 2.50
R3: 25.00 10.00 2.50
```

2 (continued)
b. (12 points)

```
typedef struct {
     char c;
     int n;
} Elem;

void main() {
     Elem arr[9] = { {'S',  4}, {'i',  4}, {'l',  5}, {'n',  5},
                     {'o', -2}, {'o', -2}, {'t', -5}, {'u', -1},
                     {'\n', 1} };
     int i = 0;

     while (i < 9) {
          printf("%c", arr[i].c);
          i = i + arr[i].n;
     }
}
```

i is the index into the array of structures; this loop does the following:
--Print the character within element i of the array
--Use the integer inside element i of the array to modify i to move to the next thing in the array

You therefore don't necessarily move through the array in order. Here's what the loop does in each iteration:

- ***i = 0: Print 'S', then set i = 0 + 4 = 4***
- ***i = 4: Print 'o', then set i = 4 + -2 = 2***
- ***i = 2: Print 'l', then set i = 2 + 5 = 7***
- ***i = 7: Print 'u', then set i = 7 + -1 = 6***
- ***i = 6: Print 't', then set i = 6 + -5 = 1***
- ***i = 1: Print 'i', then set i = 1 + 4 = 5***
- ***i = 5: Print 'o', then set i = 5 + -2 = 3***
- ***i = 3: Print 'n', then set i = 3 + 5 = 8***
- ***i = 8: Print '\n' (newline), then set i = 8 + 1 = 9***
  → Loop ends since i is no longer < 9

OUTPUT:

```
Solution
```

2 (continued)
c. (14 points)

```
typedef struct {
     unsigned int f1;
     unsigned int f2;
     unsigned int f3;
     unsigned int f4;
} FList;

void split(unsigned int x, FList *i) {
     i->f1 = (x & 0xF0000000) >> 28;
     i->f2 = (x & 0x0FF00000) >> 20;
     i->f3 = (x & 0x000FFFF0) >> 4;
     i->f4 = (x & 0x0000000F);
}

void main() {
     FList A, B, C;

     split(0xA160216A, &A);
     split(0x10200034, &B);
     split(0xDEADBEEF, &C);

     printf("%#x %#.2x %#.4x %#x\n", A.f1, A.f2, A.f3, A.f4);
     printf("%#x %#.2x %#.4x %#x\n", B.f1, B.f2, B.f3, B.f4);
     printf("%#x %#.2x %#.4x %#x\n", C.f1, C.f2, C.f3, C.f4);
}
```

This function isolates different bit fields within x and places them in the structure pointed to by i. The fields are the 4 highest bits (bits 28-31), the next 8 bits (bits 20-27), the next 16 bits (bits 4-19), and the lowest 4 bits (bits 0-3)

A = {0xA, 0x16, 0x0216, 0xA}
B = {0x1, 0x02, 0x0003, 0x4}
C = {0xD, 0xEA, 0xDBEE, 0xF}

OUTPUT:

```
0xa 0x16 0x0216 0xa
0x1 0x02 0x0003 0x4
0xd 0xea 0xdbee 0xf
```

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. `unsigned int chooseBits(unsigned int val, int lo, int n);`

This function extracts and returns a bit field from an unsigned int, `val`. The other arguments are the starting bit position (`lo`) and number of bits to extract (`n`). (The least significant (rightmost) bit is position 0.) The key is generating the proper bit mask, which is done in one of two ways:

- Generate two "masks"—one with 1s starting at the lowest position and going up to position 31; the other with 1s starting at the highest position and going down to position 0—then combine them into one mask with 1s in the positions that overlap.
- Start with a mask with the maximum number of 1s. Shift it one direction to create a mask containing `n` bits set to 1, then the other direction to put those 1s in the proper position.

Function examples are below; the bit mask generated inside the function is shown in parentheses:

- `chooseBits(0x12345678, 8, 16) = 0x3456` *(mask = 0x00FFFF00)*
- `chooseBits(0xAABBCCDD, 24, 8) = 0xAA` *(mask = 0xFF000000)*
- `chooseBits(0xDEADBEEF, 0, 4) = 0xF` *(mask = 0x0000000F)*

Students were responsible for completing bold, underlined, italicized code.

```
unsigned int chooseBits(unsigned int val, int lo, int n) {
  unsigned int mask;    // Bitmask used to isolate correct bits

  // Set mask so that it contains 1 in n_bits consecutive
  //   positions, starting at position lo_bit
  mask = 0xFFFFFFFF >> (31 - (lo + n - 1));
  mask = mask & (0xFFFFFFFF << lo);

  // Use mask to clear unwanted bits, then shift value
  //   into lowest bit position and return
  return (val & mask) >> lo_bit;
}
```

3 (continued)

b. `void printBinary(FILE *fp, unsigned int val);`

This function should, given an unsigned integer (`val`), print its value in binary to the file pointed to by `fp`. To make the value more readable, the function should print a space after every group of 4 bits. For example:

- `printBinary(stdout, 0x01010101)` prints:
  `0000 0001 0000 0001 0000 0001 0000 0001`
- `printBinary(stdout, 0xDEADBEEF)` prints:
  `1101 1110 1010 1101 1011 1110 1110 1111`

Students were responsible for completing bold, underlined, italicized code.

```
void printBinary(FILE *fp, unsigned int val) {
   unsigned int mask;          // Bit mask used to test each bit
   int i;                 // Loop index

   // Initialize variables if needed
   mask = 0x80000000;

   // Write loop to go through all bits of val
   for (i = 0; i < 32; i++) {

      // Use mask to test value of bit
      // Explicitly test for and print one value (0 or 1) to file
      if ((val & mask) == 0)
         fprintf(fp, "0");


      // Print other value (1 or 0, depending on what's
      //   printed above)
      else
         fprintf(fp, "1");

      // Print a space after every 4th digit
      if ((i % 4) == 3)
         fprintf(fp, " ");

      // Update mask to be used to test next bit
      mask = mask >> 1;
   }
}
```

3 (continued)

c. `unsigned int mulWithoutMul(unsigned int v1, unsigned int v2);`

This function performs multiplication `v1*v2` without using the `*` operator. As briefly discussed in class, multiplication can be broken into shift and add operations. For example:

- `v1 * 7  = (v1 * 4) + (v1 * 3) = (v1 << 2) + (v1 + v1 + v1)`
- `v1 * 33 = (v1 * 32) + v1      = (v1 << 5) + v1`

The function performs this operation by first determining the shift amount, then finding the number of times to repeatedly add x. Consider the following hints:

- The shift amount is the exponent of the largest power of 2 that's less than or equal to v2.
- The number of times to add is the difference between v2 and that largest power. Looking at the examples above:
  - If v2 = 7, largest power of 2 is $4 = 2^{\underline{2}}$, so left shift by **2**, then add 7 – 4 = **3 times**
  - If v2 = 33, largest power of 2 is $32 = 2^{\underline{5}}$, so left shift by **5**, then add 33 – 32 = **1 time**

Students were responsible for completing bold, underlined, italicized code.

```
unsigned int mulWithoutMul(unsigned int v1, unsigned int v2) {
   unsigned int shamt;         // Amount to shift by
   unsigned int result;   // Final result
   int pow2;                   // Largest power of 2 <= v2
   int i;                      // Loop index

   // Find shift amount by finding largest power of 2 <= v2, then
   //  start computing result by shifting v1 left by shift amount
   shamt = 0;
   pow2 = 1;
   while (pow2 <= v2) {
      pow2 = pow2 * 2;
      shamt++;
   }
   result = v1 << (shamt - 1);

   // Finish computing result by repeatedly adding v1 to result
   // Number of times to add v1 is based on what's left over
   //  in v2 after taking out largest power of 2 found above
   for (i = 0; i < (v2 - (pow2/2)); i++)
      result = result + v1;

   return result;
}
```