Section 1: Multiple choice *(select all that apply)* - *30* points

01. Suppose '*int c = 0xdeadbeef;*', which of the following C statements clear the rightmost two bits and leaves the rest of the bits unchanged?

(a) c = 0xdeadbeec
(b) c &= 0xfffffffc
(c) c = (c >> 2) << 2
(d) c |= 0xfffffffc
(e) c |= 0x00000003
(f) None of the above

02. Consider the following code snippet

```
1  float f = 1.0;
2  int b = *((int*) &f);
3  int c = 1;
```

Which of the following logical statements are true?

(a) b != c
(b) b > c
(c) b < c
(d) b == c
(e) None of the above

03. What is the smallest (most negative) normalized value represented using 6-bit IEEE-754 like encoding with one sign bit, 3 *exp* bits and 2 *frac* bits?

(a) 111011
(b) 111111
(c) 100100
(d) 000000
(e) 000100
(f) 011011

04. Assume that register **%rax** stores value *0x100* and register **%rdx** stores value *0x3*. What is the address calculated by*10(%rax, %rdx)*?

(a) 0x10D
(b) 0x113
(c) 0x927
(d) 0x903
(e) 10(%rax,%rdx) is not an address specification
(f) None of the above

05. What is the bit-vector corresponding to -10 when written using two's complement notation and using 5 bits?

(a) 00011
(b) 10110
(c) 10100
(d) 01010
(e) -10 cannot be represented using only 5 bits
(f) None of the above

06. What is the content of array A after executing the following code snippet?

```
1  long A[3] = {1, 2, 3};
2  long* p;
3  long** q;
4  p = A;
5  p++;
6  q = &p;
7  p++;
8  (*p) = (**q) * 2;
```

(a) 1,4,3
(b) 1,2,6
(c) 1,2,4
(d) 1,2,3
(e) 2,4,6
(f) 1,6,3
(g) None of the above

07. Suppose '*int x = 0x0023;*' and '*int y = 0x2300*', which of the following expressions evaluates to true if used in an if-statement?

(a) (x && y)
(b) (x & y)
(c) Both
(d) Neither

08. What is the printed by the printf statement in this code snippet?

```
1  int i = INT_MAX;
2  char* c = (char*) &i;
3  printf("%d \n", *c);
```

(a) -128
(b) **-1**
(c) 0
(d) 127
(e) None of the above

09. Which of the following types has a largest size (in terms of bytes), assuming 64-bit machine?

(a) double*
(b) float*
(c) char*
(d) short*
(e) All are same size

10. What is the value of x after the following code lines are executed?

```
1  int x = -15;
2  unsigned ux = 5;
3  x = x + ux;
```

(a) -10
(b) 10
(c) 22
(d) -22
(e) C does not allow addition of signed and unsigned values
(f) None of the above

Section 2: Bit Shifting - 15 points

Suppose that '*char x = 0x2A*' and '*char y = 0x55*'.

Determine the decimal values of the following C expressions. Assume arithmetic shift if necessary.

(I did not specify to interpret as an integer result or as a char result, which would affect the answer. Therefore answers have both and both can be considered correct.)

01. x | y

(x | y) as char = 127
(x | y) as int = 127

02. x && y

(x && y) as char = 1
(x && y) as int = 1

03. x << 3

(x << 3) as char = 80
(x << 3) as int = 336

04. (x & 0x0f) << 4

(x & 0x0f) << 4 as char = -96
(x & 0x0f) << 4 as int = 160

05. (x >> 7) & (~x)

(x >> 7) & (~x) as char = 0
(x >> 7) & (~x) as int = 0

Section 3: X86-64 - 15 points

What follows is some C functions and their derived x86-64 assembly proceedures. ***However, the assembly proceedures are not in the same order as the C functions from which they were compiled***. Moreover, the assembly proceedure starting at line 1 on the right does not necessarily derive from the C function beginning on line 1 on the left.

```
1  long foo (long x, long y)
2  {
3      long t1 = x + 3 * y;
4      long t2 = t1 * 5;
5      long t3 = t2 - y;
6      return t3;
7  }
8
9  long bar (long x, long y)
10 {
11     long t1 = 4 * x + y;
12     long t2 = t1 - 9;
13     long t3 = t2 * 3 * y;
14     return t3;
15 }
16
17 long baz (long x, long y)
18 {
19     long t1 = x + y;
20     long t2 = t1 * 7;
21     long t3 = t2 - 5 * x;
22     return t3;
23 }
```

```
1  ???:
2    addq    %rdi, %rsi
3    leaq    0(,%rsi,8), %rax
4    subq    %rsi, %rax
5    leaq    0(,%rdi,4), %rdx
6    negq    %rdx
7    subq    %rdi, %rdx
8    addq    %rdx, %rax
9    ret
10
11 ???:
12   leaq    (%rsi,%rsi,2), %rax
13   addq    %rax, %rdi
14   leaq    (%rdi,%rdi,4), %rax
15   subq    %rsi, %rax
16   ret
17
18 ???:
19   leaq    -9(%rsi,%rdi,4), %rax
20   leaq    (%rax,%rax,2), %rax
21   imulq   %rsi, %rax
22   ret
```

01. On what line number in the assembly does the code for the C function *foo(long, long)* begin? **(2 points)**

11

02. On what line number in the assembly does the code for the C function *bar(long, long)* begin? **(2 points)**

18

03. On what line number in the assembly does the code for the C function *baz(long, long)* begin? **(2 points)**

1

04. Note line 12 and 13 of the assembly. What line(s) of C code do they derive from? Explain in your own words exactly how lines 12 and 13 accomplish the intent of the corresponding lines of C code.

These lines correspond to line 3 of the C code. (1 point)
Line 12 in the assembly 'leaq(%rsi, %rsi, 2), %rax' mutiplies 'y' by 3 and puts the result in register %rax. (2 point)
Line 13 of the assembly 'addq %rax, %rdi' sums the result of line 12 with %rdi (which contains x's value). (2 point)

05. Note line 19 of the assembly. What line(s) of C code does it derive from? Explain in your own words exactly how line 19 accomplishes the intent of the corresponding lines of C code. **(5 points)**

This caculation is equivalent to lines 11 and 12 in the C code. (1 point)
'leaq -9(%rsi, %rsi, 4), %rax' performs the calculation y + x * 4 - 9 and stores it in register %rax. (2 point)
This is possible since t1 is only ever used in the context of calculating t2. (1 point)

Section 4: IEEE 754 - 20 points

Consider the following 6-bit floating point representation based on the IEEE floating point format. There is a sign bit ***s*** in the most significant bit. The next three bits are the ***exp***, with an exponent bias of 3. The last two bits are the ***frac***.

The rules are like those in the IEEE standard (normalized, denormalized, representation of 0, infinity, and NaN).

We consider the floating point format to encode numbers in a form:

$$V = (-1)^s \times M \times 2^E$$

...where M is the significand/mantissa and E is the exponent.

Fill in missing entries in the table with the following instructions for each column:

- **Type**: normalized, denormalized or special
- **M_2**: the value of M using binary format
- **M_{10}**: the value of M using decimal format
- **E_{10}**: the value of E in decimal format
- **V_{10}**: the numeric value represented by the binary sequence using decimal format.

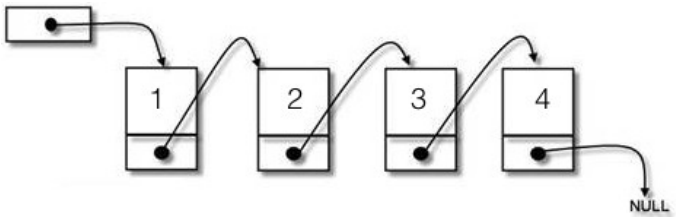
Use the below space for your work (as well as the provided scrap paper), but enter all the answers into the table.

Bit Vector	Type	M_2	M_{10}	E_{10}	V_{10}
011111	**Special (1 point)**	**Blank | N/A | NaN (1 point)**	**Blank | N/A | NaN (1 point)**	**Blank | N/A | NaN (1 point)**	**Nan (2 point)**
101101	**Normalized (1 point)**	**1.01 (1 point)**	**1.25 (1.5 point)**	**0 (1.5 point)**	**-1.25 (2 point)**
000011	**Denormalized (1 point)**	**0.11 (1 point)**	**.75 (1.5 point)**	**-2 (1.5 point)**	**.1875 (2 point)**

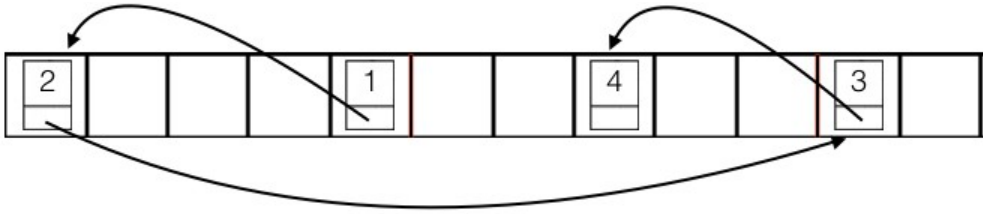
Note: Some base 10 answers might be in fraction equivalent values
Note: If they fill in the values of the bit patterns for special values but list type and V_{10} correctly, only deduct 1 pt.

Section 5: C Programming - 20 points

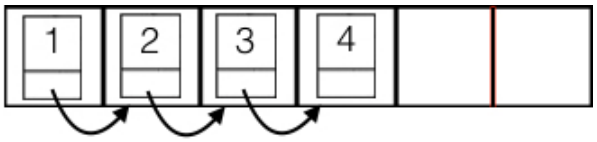
In lab1, we added and removed nodes on a linked list. Conceptually we think of a linked list as looking something like this…



As we allocate nodes for a given linked list, we typically do some one at a time. Furthermore, we usually have a call to malloc() for *each* node. A consequence of this is that the nodes are scattered throughout memory, like this…



To "pack" a linked list would be to move every node in the list such that they are adjacent (next to each other) in memory, like this…



Write a function **pack(...)** that takes two arguments:

- the head of a linked list (whose nodes are allocated at arbitrary points in heap memory)
- the number of nodes in the linked list

The function will modify the head to point to an identical linked list where all list nodes are adjacent in heap memory.

The function should return void.

Think carefully about your parameter types.

Make sure you do not leak memory.

You not have to write a whole program, no main function required.

In your program you can use the following linked list node struct, assume it is declared and defined.

```
struct node {
  int value;
  struct node* next;
};
```

Put your answer on the following, lined page.

```c
void pack(struct node** headp, int length)
{
    // Malloc a contiguous block of memory on the heap
    // that is big enough for the entire list.
    struct node* new_list = malloc(sizeof(struct node) * length);

    // Go through the list and make a copy of all the
    // nodes and place it into the contiguous block in order.
    struct node* n = *headp;
    int i;
    for(i = 0 ; i < length ; ++i) {
        struct list_node copy = { n->value, NULL };
        new_list[i] = copy;
        n = n->next;
    }

    // Go through the list again and "link" each node in the
    // contiguous block to the next node in memory.
    // Also, free all the nodes in the old list.
    n = *headp;
    for(i = 0 ; i < length - 1 ; ++i) {
        // Link each node in the new list
        new_list[i].next = &new_list[i+1];
        // Free the old nodes
        struct list_node* free_me = n;
        n = n->next;
        free(free_me);
    }

    // Don't forget to free the last node of the old list!
    free(n);

    // Point head at the new packed list
    *headp = new_list;
}
```

Grading Guide

Additive points – For each of the below, award the points.

- **1 pt - correctly choosing double pointer type for head parameter**
- **1 pt - rest of the method signature**
- **2 pt - using malloc properly**
  - **1 pt - single allocation of contiguous block**
  - **1 pt – using sizeof properly**
- **1 pt – creating struct and copying it into array**
- **1 pt – accessing struct members properly**
- **1 pt – looping over the old list properly**
- **1 pt – linking the new list properly**
- **2 pt – freeing old list**
- **10 pts – general flow of program.**
  - **At a 10k ft level, is the code attempting to do the right things. Does their answer indicate an understanding of the problem and what is required for a solution? Moreover, 5 pts for writing any reasonable C code, then 5 more pts if the code demonstrates conceptual undertanding.**