

Do not open until I give a verbal instruction to do so

Name:

NetId:

Exam Algorithm:

- Collect this exam
- Proceed to seat
- Sign the exam and your cheat sheet.
- Read instructions below carefully
- When I give the verbal instruction, open and begin the exam
- Do not leave seat (*unless medical emergency*)
- Finish exam
- If you finish before 3:15 {
 - Raise your hand
 - Hand me your exam your cheat sheet
 - Wait until 3:15 (You may use your phone)
- } else {
 - Return your exam and your cheat sheet to me
- }
- Exit

Instructions:

- You have 70 minutes.
- If you make a mess, clearly indicate your final answer.
- Read the instructions for each section carefully!
- If you have any questions during the exam, raise your hand and I will try to get to you.
- At the end of this exam there are blank pages. Use these as your scrap paper.
- There will be zero tolerance for cheating.
- If you **must** use the bathroom bring me your phone and exam, then you may go.
- At no point panic, if you get stuck, move on and return to the question later.

Do not open until I give a verbal instruction to do so

Grading Sheet

	Points
Section 1	/ 30
Section 2	/ 15
Section 3	/ 15
Section 4	/ 20
Section 5	/ 20
Total	/ 100

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 &= 0xffffffffc`
- (c) `c = (c >> 2) << 2`
- (d) `c |= 0xffffffffc`
- (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.

01. `x | y`

02. `x && y`

03. `x << 3`

04. `(x & 0x0f) << 4`

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

Section 3: X86-64 - 15 points

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

1	<code>long foo (long x, long y)</code>	1	<code>???:</code>
2	<code>{</code>	2	<code>addq %rdi, %rsi</code>
3	<code> long t1 = x + 3 * y;</code>	3	<code>leaq 0(,%rsi,8), %rax</code>
4	<code> long t2 = t1 * 5;</code>	4	<code>subq %rsi, %rax</code>
5	<code> long t3 = t2 - y;</code>	5	<code>leaq 0(,%rdi,4), %rdx</code>
6	<code> return t3;</code>	6	<code>negq %rdx</code>
7	<code>}</code>	7	<code>subq %rdi, %rdx</code>
8		8	<code>addq %rdx, %rax</code>
9	<code>long bar (long x, long y)</code>	9	<code>ret</code>
10	<code>{</code>	10	
11	<code> long t1 = 4 * x + y;</code>	11	<code>???:</code>
12	<code> long t2 = t1 - 9;</code>	12	<code>leaq (%rsi,%rsi,2), %rax</code>
13	<code> long t3 = t2 * 3 * y;</code>	13	<code>addq %rax, %rdi</code>
14	<code> return t3;</code>	14	<code>leaq (%rdi,%rdi,4), %rax</code>
15	<code>}</code>	15	<code>subq %rsi, %rax</code>
16		16	<code>ret</code>
17	<code>long baz (long x, long y)</code>	17	
18	<code>{</code>	18	<code>???:</code>
19	<code> long t1 = x + y;</code>	19	<code>leaq -9(%rsi,%rdi,4), %rax</code>
20	<code> long t2 = t1 * 7;</code>	20	<code>leaq (%rax,%rax,2), %rax</code>
21	<code> long t3 = t2 - 5 * x;</code>	21	<code>imulq %rsi, %rax</code>
22	<code> return t3;</code>	22	<code>ret</code>
23	<code>}</code>		

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

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

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

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.

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.

** Be concise! Overly verbose or circuitous answers lose points.

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:

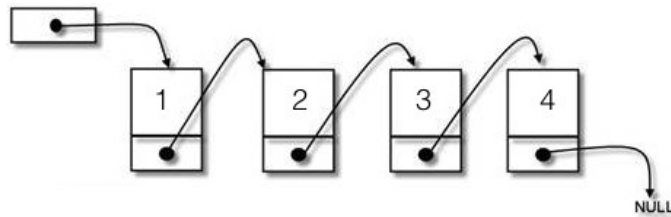
- **Type:** normalized, denormalized or special
- **M_2 :** the value of M using binary format
- **M_{10} :** the value of M using decimal format
- **E_{10} :** the value of E in decimal format
- **V_{10} :** the numeric value represented by the binary sequence using decimal format.

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

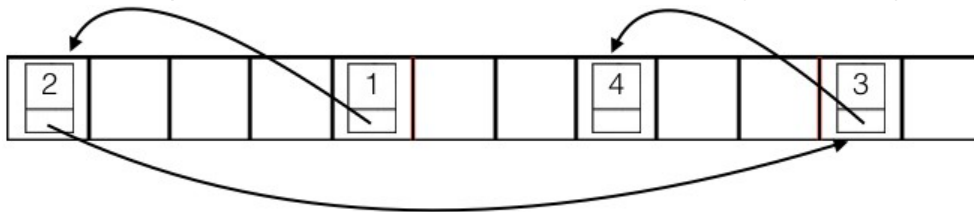
Bit Vector	Type	M_2	M_{10}	E_{10}	V_{10}
011111					
101101					
000011					

Section 5: C Programming - 20 points

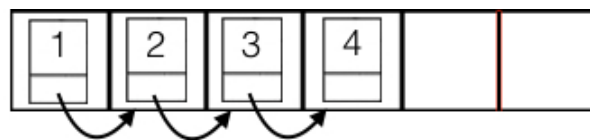
In lab1, we added and removed nodes on a linked list. Conceptually we think of a linked list as looking something like this...



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.

CSCI-UA.0201

Spring 2016

Midterm Exam A

CSCI-UA.0201

Instructor: Randy Shepherd

Spring 2016

Midterm Exam A

CSCI-UA.0201

Instructor: Randy Shepherd

Spring 2016
