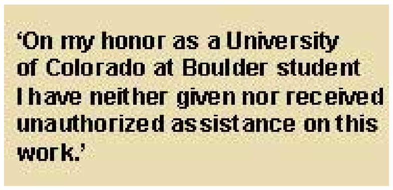
Full Name: ____


'On my honor as a University of Colorado at Boulder student I have neither given nor received unauthorized assistance on this work.'

CSCI 2400, Summer 2013

Sample Midterm Exam

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name on the front. Put your name or student ID on each page.
- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
- This exam is OPEN BOOK and you can use a *single page* of notes. Please attach your single sheet of notes to your exam when you're done. You can not use a computer or calculator. Good luck!

Problem	Page	Possible	Score
1	1	24	
2	2	13	
3	3	20	
4	5	18	
5	6	10	
6	7	15	
Total		100	

1. **[24 Points]** In the following questions assume the variable `x` is a signed integer and that the machine uses two's complement representation. Also assume that `TMax` is the maximum integer, `TMin` is the minimum integer, and `W` is one less than the word length (e.g., `W` = 31 for 32-bit integers).

   The >> operator behaves as an arithmetic shift.

   Match each of the descriptions on the left with a line of code on the right (write in the letter). You will be given 6 points for each correct match.

```
                         a)  (x << W) >> W

1)  x > 0                b)  1 << W

2)  x != 0               c)  !(!x | x >> W)

3)  TMin                 d)  ~x - (TMin + TMax)

4)  -x                   e)  ( (x >> W) & 0x1 ) | !x

                         f)  ((~x+1 | x) >> W) & 1
```

2. **[13 Points]** Assume we are running code on a 9-bit machine using two's complement arithmetic for signed integers. Also assume that `TMax` is the maximum integer, `TMin` is the minimum integer. Fill in the empty boxes in the table below. The following definitions are used in the table:

```
int y = -12;
int x = 7;
```

Note: You need not fill in entries marked with "–".

Each blank space is 1 point.

In the column labeled "Over/Under", you should indicate if an overflow (carry out of the highest bit) or underflow (borrow from the highest bit) occured.

Expression	Decimal Representation	Hex Representation	Over/Under?
–		0x1c4	–
–	74		–
y	-12		
x+y			
x + TMax			
TMin-x			

3. **[20 Points]** Jane Q. Programmer is working on a weather model program on a computer with 10 bit IEEE floating point numbers that use round-to-even mode. This includes a sign bit, 5-bit exponnent field with bias 16 and a 4 bit mantissa / fractional field).

   (a) **[5 Points]** How would you represent the number 240 in this floating point format? You should indicate what portion of the binary digits below are sign, exponent and mantissa components and use the IEEE format as described in the book.

   (b) **[5 Points]** Show what `0000001110` represents in this floating point format? Convert the mantissa portion to a decimal (base 10) number (*e.g.* 1.234) and show the exponent expressed as a power of two. It may be useful to know that $1/2 = 0.5, 1/4 = 0.25, 1/8 = 0.125, 1/16 = 0.0625, 1/32 = 0.03125$. Your resulting number should be formatted something like $1.234 * 2^{56}$, but you should obviously write out the proper value represented by `1000001010`.

   ______ . ______________________________ $*2^{------}$

   (c) **[10 Points]**

   The 10-bit float type is specified using `float10`. Jane is concerned about the limited precision of the `float10` data type. She has to add two `float10` numbers together. She wants to have a check to determine if a loss of precision has occured because of the limited range of representable values and the problems that occur when adding very large numbers to very small numbers.

   She has a program fragment that appears as below:

```
float10 x = 256;
float10 y = ......;

if ( y <= ______________ ) {
   printf("This value of Y will not change x\n");
} else {
  x = x + y;
}
```

   What is the largest value for `y` that will cause the `printf` to be executed? You should give your value as a decimal number. Remember to account for any needed rounding using the standard round-to-even mode.

4. **[18 Points]** Consider the following code for a C loop

Translate this code	into C
`quiz1:`	
`.LFB22:`	
`  pushl   %ebx`	
`  movl    8(%esp), %eax`	
`  movzwl  12(%esp), %ebx`	`__________ quiz1( ___________ a, __________ salt)`
`  movzbl  (%eax), %edx`	`{`
`  movl    %ebx, %ecx`	`  ________ x = __________;`
`  testb   %dl, %dl`	
`  je      .L2`	`  while ( ________________ ) {`
`.L3:`	
`  sall    $4, %ecx`	`    _______ = __________;`
`  movzbl  %dl, %edx`	
`  addl    %edx, %ecx`	`    _______ = __________;`
`  addl    $1, %eax`	
`  movzbl  (%eax), %edx`	`    ____________;`
`  testb   %dl, %dl`	`  }`
`  jne     .L3`	
`.L2:`	`  return ________________`
`  movzwl  %cx, %eax`	`}`
`  movzwl  %bx, %ebx`	
`  subl    %ebx, %eax`	
`  popl    %ebx`	
`  ret`	

You need to indicate the types for each variable, the value to which 'y' is initialized, what is returned and the equivalent computation. If you need to use more space, use the back of the page – don't put in lots of circles and arrows that makes it hard to grade.

You should be able to represent your program using the template on right hand side of the table above; if you feel you can't, you can use the back of the page, but you're strongly advised to use that template as a guide to the structure of your program.

5. **[14 Points]** Assume you've been given the following program fragment with a struct that contains only scalars except for one entry.

```
struct foo {

______________________

______________________

______________________

______________________

______________________ E[3];

______________________
};

int
bar(struct foo *f)
{
  f[1].C = f[0].B > f[2].C;
  f[0].E[-1] = f[0].D;
  f[0].A = f[1].C;
}
```

which produces this assembly code:

```
bar:
movl 4(%esp), %eax
movzbl 65(%eax), %edx
cmpl %edx, 12(%eax)
setg %dl
movb %dl, 33(%eax)
movl 28(%eax), %ecx
movl %ecx, 12(%eax)
movb %dl, (%eax)
ret
```

Knowing the struct contains precisely five scalar variables (A, B, C, D, F) and one array (E), use the C alignment rules and variable sizes to determine both the order in which the variables are declared and the types of those variables. One of the variables (F) is not mentioned in the program, but you should be able to determine the needed information anyway. If you can not determine if a variable is signed or unsigned from the code, assume it is signed. But, if it's clearly unsigned, you must indicate that.

3. **[18 Points]** The next problem concerns the following C code. This program reads a string on standard input and prints an integer in hexadecimal format based on the input string it read.

```
#include <stdio.h>

/* Read a string from stdin into buf */
int evil_read_string()
{
    int buf[2];

    scanf("%s",buf);
    return buf[1];
}

int main()
{
    printf("0x%x\n", evil_read_string());
}
```

Here is the corresponding machine code on a Linux/x86 machine (note: the compile generates some odd code, but it's all correct):

```
08048414 <evil_read_string>:
 8048414:  55                      push   %ebp
 8048415:  89 e5                   mov    %esp,%ebp
 8048417:  83 ec 14                sub    $20,%esp
 804841a:  53                      push   %ebx
 804841b:  83 c4 f8                add    $-8,%esp
 804841e:  8d 5d f8                lea    -8(%ebp),%ebx
 8048421:  53                      push   %ebx                 address arg for scanf
 8048422:  68 b8 84 04 08          push   $0x80484b8           format string for scanf
 8048427:  e8 e0 fe ff ff          call   804830c <_init+0x50>  call scanf
 804842c:  8b 43 04                mov    0x4(%ebx),%eax
 804842f:  8b 5d e8                mov    -24(%ebp),%ebx
 8048432:  89 ec                   mov    %ebp,%esp
 8048434:  5d                      pop    %ebp
 8048435:  c3                      ret

08048438 <main>:
 8048438:  55                      push   %ebp
 8048439:  89 e5                   mov    %esp,%ebp
 804843b:  83 ec 08                sub    $0x8,%esp
 804843e:  83 c4 f8                add    $-8,%esp
 8048441:  e8 ce ff ff ff          call   8048414 <evil_read_string>
 8048446:  50                      push   %eax                 integer arg for printf
 8048447:  68 bb 84 04 08          push   $0x80484bb           format string for printf
 804844c:  e8 eb fe ff ff          call   804833c <_init+0x80>  call printf
 8048451:  89 ec                   mov    %ebp,%esp
 8048453:  5d                      pop    %ebp
 8048454:  c3                      ret
```

This problem tests your understanding of the stack discipline and byte ordering. Here are some notes to help you work the problem:

- `scanf("%s", buf)` reads an input string from the standard input stream (stdin) and stores it at address `buf` (including the terminating `'\0'` character). It does **not** check the size of the destination buffer.
- `printf("0x%x", i)` prints the integer i in hexadecimal format preceded by "`0x`".
- Recall that Linux/x86 machines are Little Endian.
- You will need to know the hex values of the following characters:

Character	Hex value	Character	Hex value
`'d'`	`0x64`	`'v'`	`0x76`
`'r'`	`0x72`	`'i'`	`0x69`
`'.'`	`0x2e`	`'l'`	`0x6c`
`'e'`	`0x65`	`'\0'`	`0x00`
		`'s'`	`0x73`

(a) **[5 Points]** Suppose we run this program on a Linux/x86 machine, and give it the string "`dr.evil`" as input on stdin.

Here is a template for the stack, showing the locations of `buf[0]` and `buf[1]`. Fill in the value of `buf[1]` (in hexadecimal) and indicate where `ebp` points just **after** `scanf` returns to `evil_read_string`.

```
                  |<- buf[0]->|<-buf[1] ->|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

(b) Suppose now we give it the input "`dr.evil.lives`" (again on a Linux/x86 machine).

i. **[8 Points]** List the contents of the following memory locations just **after** `scanf` returns to `evil_read_string`. Each answer should be an unsigned 4-byte integer expressed as 8 hex digits.

`buf[0] = 0x`________________________

`buf[3] = 0x`________________________

ii. **[5 Points]** Immediately **before** the `ret` instruction at address `0x08048435` executes, what is the value of the frame pointer register `%ebp`?

`%ebp  = 0x`________________________

You can use the following template of the stack as *scratch space*. *Note:* this does **not** have to be filled out to receive full credit.

```
              <- buf[0] -><- buf[1] ->
--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--
  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--
```