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, Spring 2014

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.
- You can use a *single* sheet of your own notes and the provided notes page. 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	20	
2	2	13	
3	3	16	
4	4	15	
5	5	18	
6	7	18	
Total		100	

1. **[20 Points]** We are running programs on a machine with the following characteristics:

   - Values of type `int` are 32 bits. They are represented in two's complement, and they are right shifted arithmetically. Values of type `unsigned` are 32 bits.
   - The `random()` function returns successive pseudo-random numbers in the range from 0 to $2^{31} - 1$.

   We generate arbitrary values `x`, `y`, and `z`, and convert them to other forms as follows:

```
/* Create some arbitrary values */
int x = random();
int y = random();
int z = random();
/* Convert to other forms */
unsigned ux = (unsigned) x;
unsigned uy = (unsigned) y;
```

   For each of the following C expressions, you are to indicate whether or not the expression *always* yields 1. If so, circle "Y". If not, circle "N". You will be graded on each problem as follows:

   - If you circle no value, you get 0 points.
   - If you circle the right value, you get 4 points.
   - If you circle the wrong value, you get -2 points (so don't just guess wildly).

Expression	Always True?
`(x<y) == (-x>-y)`	Y N
`((x+y)<<4) + y-x == 17*y+15*x`	Y N
`~x+~y+1 == ~(x+y)`	Y N
`ux-uy == -(y-x)`	Y N
`(x >= 0) || (x < ux)`	Y N

2. **[13 Points]** Assume we are running code on a 5-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 = 9
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?
–		0x0d	–
–	-7		–
y	9		–
x+y			
x-y	-2		
TMax + x			
TMin-x			

3. **[16 Points]** One of the most disgusting scenes in movie history is the dinner of Mr. Creosote, a grossly crass diner appearing in the Monty Python Movie, *"The Meaning of Life"*. Famously, after eating four dinners with a case of wine and a six crates of beer, Mr. Creosote is offered a "wafer thin mint" that leads to disasterous consequences.

   In all the following questions, assume you have a 12-bit IEEE-compatible floating point format with a sign-bit, five exponent bits, six mantissa bits and an exponent bias of 15 (*i.e.* the normal IEEE bias, specified as $2^{5-1}-1$ for a 5-bit exponent)).

   Answer the following questions:

   (a) **[4 Points]** Assume the mass M_C of Mr. Creosote is 236kg. Represent this as a binary floating point number.

   _ . _____ x 2**_____

   (b) **[4 Points]** Represent the mass of Mr. Creosote in kg using an IEEE 12-bit floating point number. Indicate the portions that are sign, exponent and mantissa by drawing lines between the labeled parts.

   (c) **[4 Points]**

   Determine the **largest mass** of the "wafer thin mint" M_{wtm} such that, when added to the mass M_C of Mr. Crosote, it does *not* change his mass when stored in the 12-bit IEEE format. In other words, determine the largest M_{wtm} such that $M_{wtm} + M_C = M_C$. You should assume the standard "round to even" rounding mode. It may be useful to determine what values can be represented *precisely* around the value of M_C .

   _ . _____ x 2**_____

   (d) **[4 Points]** What is the **smallest possible non-zero positive mass** that can be represented in this IEEE representation. Write down that mass below.

4. **[15 Points]** Condider the following assembly code for a C `for` loop:

```
loop:
        pushl %ebp
        movl %esp,%ebp
        movl 0x8(%ebp),%edx
        movl %edx,%eax
        addl 0xc(%ebp),%eax
        leal 0xffffffff(%eax),%ecx
        cmpl %ecx,%edx
        jae .L4
.L6:
        movb (%edx),%al
        xorb (%ecx),%al
        movb %al,(%edx)
        xorb (%ecx),%al
        movb %al,(%ecx)
        xorb %al,(%edx)
        incl %edx
        decl %ecx
        cmpl %ecx,%edx
        jb .L6
.L4:
        movl %ebp,%esp
        popl %ebp
        ret
```

Based on the assembly code above, fill in the blanks below in its corresponding C source code. (Note: you may only use the symbolic variables `h`, `t` and `len` in your expressions below — *do not use register names.*)

```
void loop(char *h, int len)
{
   char *t;

   for (_______________; _____________; h++,t--) {

      _________________;

      _________________;

      _________________;
   }

   return;
}
```

5. **[18 Points]** Consider the following C declaration:

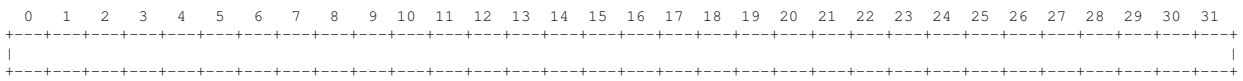
```
struct Node{
   char c;
   double value;
   struct Node* next;
   int flag;
   struct Node* left;
   struct Node* right;
};

typedef struct Node* pNode;

/* NodeTree is an array of N pointers to Node structs */
pNode NodeTree[N];
```

A. **[6 Points]** Using the template below (allowing a maximum of 32 bytes), indicate the allocation of data for a `Node` struct. Mark off and label the areas for each individual element (there are 6 of them). Cross hatch the parts that are allocated, but not used (to satisfy alignment).

Assume the Linux alignment rules. **Clearly indicate the right hand boundary of the data structure with a vertical line**.



B.**[12 Points]** For each of the four C references below, please indicate which assembly code section (labeled A – F) places the value of that C reference into register `%eax`. If no match is found, please write "NONE" next to the C reference.

The initial register-to-variable mapping for each assembly code section is:

```
%eax = starting address of the NodeTree array
%edx = i
```

References:

1. _____ `pNode[i]->flag`

2. _____ `pNode[i]->left->left->c`

3. _____ `pNode[i]->next->next->flag`

4. _____ `pNode[i]->right->left->left`

Linux/IA32 Assembly:

A.
```
sall $2, %edx
leal (%eax,%edx),%eax
movl 16(%eax),%eax
```

B.
```
sall $2,%edx
leal (%eax,%edx),%eax
movl (%eax),%eax
movl 24(%eax),%eax
movl 20(%eax),%eax
movl 20(%eax),%eax
```

C:
```
sall $2,%edx
leal (%eax,%edx),%eax
movl 20(%eax),%eax
movl 20(%eax),%eax
movsbl (%eax),%eax
```

D:
```
sall $2,%edx
leal (%eax,%edx),%eax
movl (%eax),%eax
movl 16(%eax),%eax
```

E:
```
sall $2, %edx
leal (%eax,%edx),%eax
movl (%eax),%eax
movl 16(%eax),%eax
movl 16(%eax),%eax
movl 20(%eax),%eax
```

F:
```
sall $2, %edx
leal (%eax,%edx),%eax
movl (%eax),%eax
movl 12(%eax),%eax
movl 12(%eax),%eax
movl 16(%eax),%eax
```

6. **[18 Points]** The next problem concerns the following C code:

```
/* copy string x to buf */
void foo(char *x) {
  int buf[1];
  strcpy((char *)buf, x);
}

void callfoo() {
  foo("abcdefghi");
}
```

Here is the corresponding machine code on a Linux/x86 machine:

```
080484f4 <foo>:
080484f4: 55              pushl  %ebp
080484f5: 89 e5           movl   %esp,%ebp
080484f7: 83 ec 18        subl   $0x18,%esp
080484fa: 8b 45 08        movl   0x8(%ebp),%eax
080484fd: 83 c4 f8        addl   $0xfffffff8,%esp
08048500: 50              pushl  %eax
08048501: 8d 45 fc        leal   0xfffffffc(%ebp),%eax
08048504: 50              pushl  %eax
08048505: e8 ba fe ff ff  call   80483c4 <strcpy>
0804850a: 89 ec           movl   %ebp,%esp
0804850c: 5d              popl   %ebp
0804850d: c3              ret

08048510 <callfoo>:
08048510: 55              pushl  %ebp
08048511: 89 e5           movl   %esp,%ebp
08048513: 83 ec 08        subl   $0x8,%esp
08048516: 83 c4 f4        addl   $0xfffffff4,%esp
08048519: 68 9c 85 04 08  pushl  $0x804859c        # push string address
0804851e: e8 d1 ff ff ff  call   80484f4 <foo>
08048523: 89 ec           movl   %ebp,%esp
08048525: 5d              popl   %ebp
08048526: c3              ret
```

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

- `strcpy(char *dst, char *src)` copies the string at address `src` (including the terminating '`\0`' character) to address `dst`. It does **not** check the size of the destination buffer.
- 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
'a'	0x61	'f'	0x66
'b'	0x62	'g'	0x67
'c'	0x63	'h'	0x68
'd'	0x64	'i'	0x69
'e'	0x65	'\0'	0x00

Now consider what happens on a Linux/x86 machine when `callfoo` calls `foo` with the input string "`abcdefghi`".

(a) List the contents of the following memory locations immediately after `strcpy` returns to `foo`. Each answer should be an unsigned 4-byte integer expressed as 8 hex digits.

`buf[0]` = _____

`buf[1]` = _____

`buf[2]` = _____

(b) Immediately **before** the `ret` instruction at address `0x0804850d` executes, what is the value of the frame pointer register `%ebp`?

`ebp` = _____

(c) Immediately **after** the `ret` instruction at address `0x0804850d` executes, what is the value of the program counter register `%eip`?

`eip` = _____