Full Name: ____

CSCI 540, Fall 2015

Practice 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	17	
4	4	15	
5	5	15	
6	6	16	
Total		100	

1. **[24 Points]** In the following questions assume the variable `a` is a signed integer and that the machine uses two's complement representation. Also assume that `MAX_INT` is the maximum integer, `MIN_INT` 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 5 points for each correct match.

   1. `((a < 0) ? (a + 3) : a) >> 2`

   2. `(~((~0) << 1)) & a`

   3. `a > 0`

   4. `(a << 4) + (a << 2) + (a << 1)`

   a. `a * 22`

   b. `~((a ^ (~0)) | a)`

   c. `((a < 0) ? a + 1 : a) >> 1`

   d. `!((a >> W) | !a)`

   e. `~((a | (~a + 1)) >> W) & 1`

   f. `a % 2`

   g. `a / 4`

   h. `1 + (a << 3) + ~a`

2. **[13 Points]** Assume we are running code on a 6-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 = -7;
int x = 12;
```

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?
–		0x38	–
–	31		–
y	-7		
x+y			
x + TMax			
TMin-x			

3. **[17 Points]** Consider the following code for a C loop

Translate this code	into C

quiz1:
.LFB24:
movl $0, %edx
movl $43, %r9d
movsbl %dil, %edi
.L3:
movzbl %dl, %r8d
movslq %r8d, %rcx
subw -24(%rsp,%rcx,4), %r9w
movl %edi, %eax
addl %edi, %r8d
movl %r8d, -24(%rsp,%rcx,4)
addl $1, %edx
movzbl %dl, %ecx
cmpl %esi, %ecx
jl .L3
movswl %r9w, %r9d
addl %r9d, %eax
ret
	quiz1(______________ a, ________________ x)
	{
	____________________ z[3];
	____________________ y = ______________;
	____________________ q = ______________;
	do {
	__________ = _________ - ____________;
	__________ = _________ + ____________;
	__________________;
	} while (_________
	(pick: <, <=, =, =>, >) __________);
	return a + y;
	}

You need to indicate the data types (short, int, char, *etc* as well as *unsigned* if appropriate) for 'a', 'x', 'y', 'q' and 'z', the value to which 'q' is initialized, what is returned and the equivalent computation. In the condition for the the **if** statement, circle the currect operator. If you need to use more space, use the back of the page – don't put in lots of circles and arrows in the template 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.**

4. **[15 Points]** Assume you’ve been given the following program fragment with a struct that contains only scalars except for one entry.

```
struct foo {

______________________


______________________


______________________


______________  five[5];


______________________
};

int bar(struct foo *f)
{
  f[0].three = f[0].two + f[1].three;
  f[0].five[3] = f[0].five[-1];
  f[0].one = f[1].three;
}
```

which produces this assembly code:

```
bar:
     movzbl 48(%rdi), %eax
     movl %eax, %edx
     addb 32(%rdi), %dl
     movb %dl, 8(%rdi)
     movl 8(%rdi), %edx
     movl %edx, 24(%rdi)
     cbtw                          #convert byte to word AL -> AX
     movw %ax, 10(%rdi)
     ret
```

Knowing the struct contains precisely five scalar variables (one, two, three, four and five) 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 (four) 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.

5. **[15 Points]** Consider the source code below, where M, N are constants declared with #define and struct s is a structure containing some variable declarations. Functions one and two are compiled into the code at the right.

```
struct s x[M];
struct s y[N][M];

one(int i, int j)
{
  x[i].a[j] = i + j;
}

two(int i, int j)
{
  y[i][j].c = i + j;
}
```

```
one:
        movslq %esi, %rdx
        movslq %edi, %rax
        leaq (%rax,%rax,2), %rax
        addq %rdx, %rax
        addl %esi, %edi
        movl %edi, x+4(,%rax,4)
        ret
two:
        movslq %esi, %rax
        movslq %edi, %rcx
        leaq (%rax,%rax,2), %rdx
        imulq $204, %rcx, %rax
        addl %esi, %edi
        movb %dil, y(%rax,%rdx,4)
        ret
```

(a) **[5 Points]** What is the size of the structure struct s? If it's impossible to determine from the information given, say that.

(b) **[5 Points]** What is the value of the constant N? If it's impossible to determine from the information given, say that.

6. **[16 Points]** The following problem concerns the following, low-quality code:

```
void foo(int x)
{
  int a[3];
  char buf[4];
  a[0] = 0xF0F1F2F3;
  a[1] = x;
  gets(buf);
  printf("a[0] = 0x%x, a[1] = 0x%x, buf = %s\n", a[0], a[1], buf);
}
```

In a program containing this code, procedure `foo` has the following disassembled form on an IA32 machine:

```
080485d0 <foo>:
80485d0: 55              pushl  %ebp
80485d1: 89 e5           movl   %esp,%ebp
80485d3: 83 ec 10        subl   $0x10,%esp
80485d6: 53              pushl  %ebx
80485d7: 8b 45 08        movl   0x8(%ebp),%eax
80485da: c7 45 f4 f3 f2  movl   $0xf0f1f2f3,0xfffffff4(%ebp)
80485df: f1 f0
80485e1: 89 45 f8        movl   %eax,0xfffffff8(%ebp)
80485e4: 8d 5d f0        leal   0xfffffff0(%ebp),%ebx
80485e7: 53              pushl  %ebx
80485e8: e8 b7 fe ff ff  call   80484a4 <_init+0x54>  # gets
80485ed: 53              pushl  %ebx
80485ee: 8b 45 f8        movl   0xfffffff8(%ebp),%eax
80485f1: 50              pushl  %eax
80485f2: 8b 45 f4        movl   0xfffffff4(%ebp),%eax
80485f5: 50              pushl  %eax
80485f6: 68 ec 90 04 08  pushl  $0x80490ec
80485fb: e8 94 fe ff ff  call   8048494 <_init+0x44>  # printf
8048600: 8b 5d ec        movl   0xffffffec(%ebp),%ebx
8048603: 89 ec           movl   %ebp,%esp
8048605: 5d              popl   %ebp
8048606: c3              ret
8048607: 90              nop
```

For the following questions, recall that: (a) `gets` is a standard C library routine; (b) IA32 machines are little-endian; (c) C strings are null-terminated; (i.e., terminated by a character with value 0x00). (d) Characters '`0`' through '`9`' have ASCII codes `0x30` through `0x39`.

Fill in the following table indicating where on the stack the following program values are located. Express these as decimal offsets (positive or negative) relative to register `%ebp`:

Program Value	Decimal Offset
`a`	
`a[2]`	
`x`	
`buf`	
`buf[3]`	
Saved value of register `%ebx`	