1.
a) N
b) Y
c) Y
d) Y

2. 1200 bytes

3.
a) 171
b) -85
c) (-1)*2^-2*1.375 = -0.34375

4.
a) Y
b) 0x100270
c) grape

5.
a.) 0x400575
c.) 0x40057F
d.) 0x4005CF
e.) 0x4005C0
d.) answers may vary (see code)

C code for problem 5

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 6
#define MAX_FIRST 10
int dud1(void)
{
  int i;
  int x;
  int a[SIZE];
  int b[SIZE+SIZE];
  for (i=0; i<SIZE; i++)
    {
      scanf("%d", &a[i]);
      b[i]=0;
      b[i+SIZE]=0;
    }
  if (a[0]>MAX_FIRST)
    return 0;
  x=a[0];
  for (i=1;i<SIZE; i++)
    switch (a[i]) {
    case 2:
      x=x-5;
    case 4:
      if (b[a[i]]==1)
        return 0;
      b[a[i]]=1;
      x=x*4;
      break;
    case 3:
      x*=2;
      if (b[a[i]]==1)
        return 0;
      b[a[i]]=1;
      break;
    case 6:
      x+=7;
      if (b[a[i]]==1)
        return 0;
      b[a[i]]=1;
      break;
    case 7:
      x+=3;
      if (b[a[i]]==1)
        return 0;
      b[a[i]]=1;
      break;
    case 8:
      x-=17;
      if (b[a[i]]==1)
        return 0;
      b[a[i]]=1;
    default:
      x=x-1;
    }
  return (x==42);
}
```

UNIVERSITY OF CALIFORNIA, LOS ANGELES UCLA

BERKELEY • DAVIS • IRVINE • LOS ANGELES • RIVERSIDE • SAN DIEGO • SAN FRANCISCO SANTA BARBARA • SANTA CRUZ

CS 33
Midterm

All answers must be written on the answer sheet (last page of the exam).

All work should be written directly on the exam, use the backs of pages if needed.

This is an open book, open notes quiz – but you cannot share books or notes. An ASCII table is on the second to last page if you need it.

I will follow the guidelines of the university in reporting academic misconduct – please do not cheat.

NAME: _____

ID: _____

Problem 1: _____

Problem 2: _____

Problem 3: _____

Problem 4: _____

Problem 5: _____

Total: _____

1. ***C If You Can Solve This (12 points)***: The following problem assumes the following declarations:

```
int x = rand();
int y = rand();
int z = rand();
float f= foo();  // f is not NaN
unsigned ux = (unsigned) x;
unsigned uy = (unsigned) y;
```

For the following C expressions, circle either Y or N (but not both).

		Always True?	
a.	(0+f)-f == 0	Y	N
b.	(ux-uy) == (x-y)	Y	N
c.	((((x<<1)^x)-x)>>1) == x ⇒ !((x&12) == 12)	Y	N
d.	(x^y)^z == y ⇒ x==z	Y	N

Note that "⇒" represents an *implication*. A ⇒ B means that you assume A is true, and your answer should indicate whether B should be implied by A – i.e. given that A is true, is B always true?

2. ***What Size Is It? (4 points):*** Consider the following structure definition:

```
struct product {
        unsigned short ID;
        union NameOrSerial {
                char name[8];
                unsigned int serial;
                } label;
} my_data[100];
```

How many bytes would my_data consume in memory on an x86-64 Linux machine?

3. ***Four Perspectives on a Bit Vector (9 points):*** Consider the following 8 bits:

   10101011

   We will interpret these bits in three different ways (assume the above is in big endian form):

   a. An 8-bit unsigned integer

   b. An 8-bit two's complement integer

   c. The 8-bit floating point format we covered in class

4. ***This Problem is a Pain in My Big Endian (35 points)***: Consider the following structure declaration:

```
struct my_struct{
  int val;
  char a;
  char b;
  short small_num;
  char * label;
  struct my_struct * left_child;
  struct my_struct * right_child;
};
```

We compile code using this struct on a 32-bit little endian machine, and insert a number of nodes into the structure. The following gdb interaction provides enough details to reverse engineer a tree built from these structs.

```
(gdb) print the_top
$3 = (struct my_struct *) 0x100130

(gdb) x/96x 0x100130
0x100130:   0x0000000a  0x00d91107  0x00001f8e  0x001001f0
0x100140:   0x00100210  0x00000000  0x00000000  0x00000000
0x100150:   0x00000005  0x00c8020a  0x00001f71  0x00100190
0x100160:   0x00100170  0x00000000  0x00000000  0x00000000
0x100170:   0x00000012  0x00431e18  0x00001f76  0x001001b0
0x100180:   0x00000000  0x00000000  0x00000000  0x00000000
0x100190:   0x00000003  0x0055180d  0x00001f7a  0x00000000
0x1001a0:   0x00000000  0x00000000  0x00000000  0x00000000
0x1001b0:   0x0000000d  0x00b3020c  0x00001f80  0x001001d0
0x1001c0:   0x00000000  0x00000000  0x00000000  0x00000000
0x1001d0:   0x00000008  0x00111707  0x00001f87  0x00000000
0x1001e0:   0x00000000  0x00000000  0x00000000  0x00000000
0x1001f0:   0x00000005  0x002f1418  0x00001f95  0x00100230
0x100200:   0x00100250  0x00000000  0x00000000  0x00000000
0x100210:   0x0000000f  0x00050d11  0x00001f9a  0x00100270
0x100220:   0x00100290  0x00000000  0x00000000  0x00000000
0x100230:   0x00000003  0x006b1518  0x00001f9f  0x00000000
0x100240:   0x00000000  0x00000000  0x00000000  0x00000000
0x100250:   0x00000008  0x00070816  0x00001fa5  0x00000000
0x100260:   0x00000000  0x00000000  0x00000000  0x00000000
0x100270:   0x0000000d  0x00481219  0x00001fab  0x00000000
0x100280:   0x00000000  0x00000000  0x00000000  0x00000000
0x100290:   0x00000012  0x00410213  0x00001fb1  0x00000000
0x1002a0:   0x00000000  0x00000000  0x00000000  0x00000000
```

(continued on next page)

```
(gdb) x/32x 0x00001f60
0x1f60:     0x00000000  0x00201868  0x1425ff00  0x90000020
0x1f70:     0x756c6200  0x65720065  0x72670064  0x006e6565
0x1f80:     0x6c6c6579  0x7000776f  0x6c707275  0x726f0065
0x1f90:     0x65676e61  0x756c7000  0x6570006d  0x6d007261
0x1fa0:     0x6e6f6c65  0x72656200  0x67007972  0x65706172
0x1fb0:     0x70706100  0x0100656c  0x1c000000  0x00000000
0x1fc0:     0x1c000000  0x00000000  0x1c000000  0x02000000
0x1fd0:     0x90000000  0x3400000b  0x34000000  0x31000000
```

Using this information, find the struct in the tree where val is equal to 13, and report the corresponding label (i.e. the actual string). Also, please fill in the blanks we have on the answer key with the requested intermediate values that would help you answer this question.

Hint – don't forget that gdb reverses byte ordering within each 4-byte chunk. So in the following dump:

```
(gdb) x/4x 0x00111110
0x111110:   0x33221100  0x77665544  0xBBAA9988  0xFFEEDDCC
```

This prints out 16 bytes of memory starting at address 0x111110. In this example, the 16 bytes of memory starting at 0x111110 would contain, in order from lowest address (0x111110) to highest address (0x11111F):

00112233445566778899AABBCCDDEEFF

So address 0x111110 contains the byte 0x00, address 0x111111 contains the byte 0x11, address 0x111112 contains the byte 0x22, and so on. So in terms of just the least significant hex place of the address, gdb is actually printing out addresses in the following order:

3 2 1 0 7 6 5 4 B A 9 8 F E D C

This is useful when reading words, but can be confusing for other values.

5. ***Revenge of the Bomb Lab (40 points):*** Here's your chance to show your bomb lab skills. Below we show the disassembled function *dud1()* – compiled on an x86-64 machine. The function does not take any parameters – but does read some number of integers, one at a time, using scanf (this is the *callq* instruction below). Your job is to provide the input that will result in this function returning the value 1 (i.e. 0x1 will be in %rax).

   To help you with this task – we provide part of the C code for the function below (note that MAX_FIRST and SIZE are macros that would be replaced with constants at compile time):

```
int dud1(void)
{
  int i;
  int x;
  int a[SIZE];
  int b[SIZE+SIZE];

  for (i=0; i<SIZE; i++)
    {
      scanf("%d", &a[i]);
      b[i]=0;
      b[i+SIZE]=0;
    }

  if (a[0]>MAX_FIRST)
    return 0;
  x=a[0];

  for (…)
    switch (…) {
    … /* some number of cases go here */
    }

  return …;
}
```

   where the …'s above represent missing code you need to figure out.

   In addition to the input string that would result in this function returning a 1, please fill in the blanks we have on the answer key with the requested intermediate values that would help you answer this question.

   The next two pages contain everything you need for this problem.

Here's the function from objdump:

```
0000000000400540 <dud1>:
  400540:       41 54                   push   %r12
  400542:       55                      push   %rbp
  400543:       53                      push   %rbx
  400544:       31 db                   xor    %ebx,%ebx
  400546:       48 83 ec 50             sub    $0x50,%rsp
  40054a:       4c 8d 64 24 30          lea    0x30(%rsp),%r12
  40054f:       48 89 e5                mov    %rsp,%rbp
  400552:       49 8d 34 1c             lea    (%r12,%rbx,1),%rsi
  400556:       31 c0                   xor    %eax,%eax
  400558:       bf 48 0b 40 00          mov    $0x400b48,%edi
  40055d:       e8 c6 fe ff ff          callq  400428 <scanf@plt>
  400562:       c7 04 2b 00 00 00 00    movl   $0x0,(%rbx,%rbp,1)
  400569:       c7 44 2b 18 00 00 00    movl   $0x0,0x18(%rbx,%rbp,1)
  400570:       00
  400571:       48 83 c3 04             add    $0x4,%rbx
  400575:       48 83 fb 18             cmp    $0x18,%rbx
  400579:       75 d7                   jne    400552 <dud1+0x12>
  40057b:       8b 4c 24 30             mov    0x30(%rsp),%ecx
  40057f:       83 f9 0a                cmp    $0xa,%ecx
  400582:       0f 8f a9 00 00 00       jg     400631 <dud1+0xf1>
  400588:       31 f6                   xor    %esi,%esi
  40058a:       66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)
  400590:       41 8b 54 b4 04          mov    0x4(%r12,%rsi,4),%edx
  400595:       83 fa 08                cmp    $0x8,%edx
  400598:       77 26                   ja     4005c0 <dud1+0x80>
  40059a:       89 d0                   mov    %edx,%eax
  40059c:       ff 24 c5 50 0b 40 00    jmpq   *0x400b50(,%rax,8)
  4005a3:       83 7c 24 20 01          cmpl   $0x1,0x20(%rsp)
  4005a8:       0f 84 83 00 00 00       je     400631 <dud1+0xf1>
  4005ae:       83 e9 11                sub    $0x11,%ecx
  4005b1:       c7 44 24 20 01 00 00    movl   $0x1,0x20(%rsp)
  4005b8:       00
  4005b9:       0f 1f 80 00 00 00 00    nopl   0x0(%rax)
  4005c0:       83 e9 01                sub    $0x1,%ecx
  4005c3:       48 83 c6 01             add    $0x1,%rsi
  4005c7:       48 83 fe 05             cmp    $0x5,%rsi
  4005cb:       75 c3                   jne    400590 <dud1+0x50>
  4005cd:       31 c0                   xor    %eax,%eax
  4005cf:       83 f9 2a                cmp    $0x2a,%ecx
  4005d2:       0f 94 c0                sete   %al
  4005d5:       48 83 c4 50             add    $0x50,%rsp
  4005d9:       5b                      pop    %rbx
  4005da:       5d                      pop    %rbp
  4005db:       41 5c                   pop    %r12
  4005dd:       c3                      retq
  4005de:       83 e9 05                sub    $0x5,%ecx
  4005e1:       48 63 c2                movslq %edx,%rax
  4005e4:       83 3c 84 01             cmpl   $0x1,(%rsp,%rax,4)
  4005e8:       74 47                   je     400631 <dud1+0xf1>
  4005ea:       c1 e1 02                shl    $0x2,%ecx
  4005ed:       c7 04 84 01 00 00 00    movl   $0x1,(%rsp,%rax,4)
  4005f4:       eb cd                   jmp    4005c3 <dud1+0x83>
  4005f6:       83 7c 24 0c 01          cmpl   $0x1,0xc(%rsp)
```

```
4005fb:       74 34                    je     400631 <dud1+0xf1>
4005fd:       01 c9                    add    %ecx,%ecx
4005ff:       c7 44 24 0c 01 00 00     movl   $0x1,0xc(%rsp)
400606:       00
400607:       eb ba                    jmp    4005c3 <dud1+0x83>
400609:       83 7c 24 18 01           cmpl   $0x1,0x18(%rsp)
40060e:       74 21                    je     400631 <dud1+0xf1>
400610:       83 c1 07                 add    $0x7,%ecx
400613:       c7 44 24 18 01 00 00     movl   $0x1,0x18(%rsp)
40061a:       00
40061b:       eb a6                    jmp    4005c3 <dud1+0x83>
40061d:       83 7c 24 1c 01           cmpl   $0x1,0x1c(%rsp)
400622:       74 0d                    je     400631 <dud1+0xf1>
400624:       83 c1 03                 add    $0x3,%ecx
400627:       c7 44 24 1c 01 00 00     movl   $0x1,0x1c(%rsp)
40062e:       00
40062f:       eb 92                    jmp    4005c3 <dud1+0x83>
400631:       48 83 c4 50              add    $0x50,%rsp
400635:       31 c0                    xor    %eax,%eax
400637:       5b                       pop    %rbx
400638:       5d                       pop    %rbp
400639:       41 5c                    pop    %r12
40063b:       c3                       retq
40063c:       0f 1f 40 00              nopl   0x0(%rax)
```

And here is some gdb interaction which should prove useful:

```
(gdb) x/112x 0x400b40
0x400b40 <>:        0x00000000      0x00000000      0x25006425      0x00000a64
0x400b50 <+16>:     0x004005c0      0x00000000      0x004005c0      0x00000000
0x400b60 <+32>:     0x004005de      0x00000000      0x004005f6      0x00000000
0x400b70 <+48>:     0x004005e1      0x00000000      0x004005c0      0x00000000
0x400b80 <+64>:     0x00400609      0x00000000      0x0040061d      0x00000000
0x400b90 <+80>:     0x004005a3      0x00000000      0x004006c0      0x00000000
0x400ba0 <+96>:     0x004006c0      0x00000000      0x004006f6      0x00000000
0x400bb0 <+112>:    0x004006a3      0x00000000      0x0040070a      0x00000000
0x400bc0 <+128>:    0x004006c0      0x00000000      0x004006de      0x00000000
0x400bd0 <+144>:    0x004006e1      0x00000000      0x0040071e      0x00000000
0x400be0 <+160>:    0x004007c0      0x00000000      0x004007c0      0x00000000
0x400bf0 <+176>:    0x004007f6      0x00000000      0x004007de      0x00000000
0x400c00 <+192>:    0x004007a3      0x00000000      0x004007c0      0x00000000
0x400c10 <+208>:    0x004007e1      0x00000000      0x0040080a      0x00000000
0x400c20 <+224>:    0x0040081d      0x00000000      0x004008c0      0x00000000
0x400c30 <+240>:    0x004008c0      0x00000000      0x004008a3      0x00000000
0x400c40 <+256>:    0x004008e1      0x00000000      0x004008de      0x00000000
0x400c50 <+272>:    0x004008c0      0x00000000      0x004008f6      0x00000000
0x400c60 <+288>:    0x00400909      0x00000000      0x0040091d      0x00000000
0x400c70 <+304>:    0x004009b1      0x00000000      0x004009b1      0x00000000
0x400c80 <+320>:    0x004009cf      0x00000000      0x004009d9      0x00000000
0x400c90 <+336>:    0x004009d2      0x00000000      0x004009b1      0x00000000
0x400ca0 <+352>:    0x004009e5      0x00000000      0x004009de      0x00000000
0x400cb0 <+368>:    0x004009af      0x00000000      0x004009ff      0x00000000
0x400cc0 <+384>:    0x004009ff      0x00000000      0x00400a1d      0x00000000
0x400cd0 <+400>:    0x00400a40      0x00000000      0x00400a36      0x00000000
0x400ce0 <+416>:    0x004009ff      0x00000000      0x00400a30      0x00000000
0x400cf0 <+432>:    0x00400a29      0x00000000      0x00400a24      0x00000000
```

Hint – don't forget gdb's trick about reversing byte ordering within each 4-byte chunk (same as problem #4).

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Answer Sheet

Name:_____

1. Y N

   Y N

   Y N

   Y N

2.

3. a.

   b.

   c.

4. Fill in all blanks below

Is the struct with the requested value a leaf node in the tree (i.e. no children)? (Y/N)

_____

Base address (in hex) of struct with the requested value:

_____

String (i.e. label field of the struct with the requested value):

_____

5. Fill in all blanks below

Address (in hex) of the assembly instruction that compares i<SIZE in the loop that calls scanf:

_____

Address (in hex) of the assembly instruction that compares a[0]>MAX_FIRST:

_____

Address (in hex) of the assembly instruction that does the comparison to produce the value returned by the return statement at the end of the C code (i.e. return(…)):

_____

Starting address for the instructions that make up the *default* case of the switch statement (absolute address in hex):

_____

Input numbers to return a 1:

_____