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.

NAM_____

ID: _____

Problem 1: _

Problem 2: _

Problem 3: _____

Total: _____

⌘

1. ***C If You Can Solve This (28 points)***: The following problem assumes the following declarations:

```
int x = rand();
int y = rand();
int z = rand();
unsigned ux = (unsigned) x;
unsigned uy = (unsigned) y;
```

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

	Always True?	
a. x+y>0 ⇒ (x>0 || y>0) x, y = −1 500 000 000 fails	Y	(N)
b. (ux+uy) == (x+y) implicit cast to unsigned on RHS	(Y)	N
c. ((x&16) | y) == y) ⇒ (x<<27)>0 x = 0 fails	Y	(N)
d. (x^y)^x + z == y+z ~~y^x^x~~ + z == y+z y + z == y + 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. ***Complete Dis-Array (30 points)***: C code contains three different two dimensional arrays, listed in no particular order: orange, purple, and green.

Array *orange* is statically sized (i.e. size is known at compile time) and is a nested array. There is a function *setorange*() which puts a value *val* into array *orange* as indicated by row position *i* and column position *j*:

int orange[SIZE][SIZE];
void setorange(int i, int j, int val);

Array *green* is dynamically sized (i.e. size is not known at compile time) and is a nested array. There is a function *setgreen*() which puts a value *val* into array *green* as indicated by *i* and *j*:

*int *green;*
void setgreen(int i, int j, int val);

Array *purple* is statically sized (i.e. size is known at compile time) and is a multi-level array. There is a function *setpurple* () which puts a value *val* into array *purple* as indicated by *i* and *j*:

*int *purple[SIZE];*
void setpurple(int i, int j, int val);

For example, *setorange(i,j,val)* will set *orange[i][j]* to the value *val*. *SIZE* is defined as:

#define SIZE 10

Based on the information above, you need to identify the disassembled code on the next page. Three functions are listed below, labeled `set__A___`, `set__B___`, and `set__C___`. These each represent either setorange, setgreen, or setpurple. Fill in the blanks on the answer sheet to identify each function below.

~ = compile-time constant

c_n = runtime constant

```
set_A_:
    lea     (%rdi,%rdi,4),%rax
    lea     (%rsi,%rax,2),%rax
    mov     %rdx,0x600c60(,%rax,8)
    retq
```

orange

rax = 5 * rdi
rax = rsi + 2 * rax
} rax = 10 * rdi + rsi

*(~ + rax * 8) = rdx

*(~ + 8 * (10 * rdi + rsi)) = rdx

```
set_B_:
    movslq 0x200a14(%rip),%rax
    imul    %rax,%rdi
    add     %rdi,%rsi
    mov     0x200a0e(%rip),%rax
    mov     %rdx,(%rax,%rsi,8)
    retq
```

green

rax = *(~) + rip =: c_1
rdi *= rax
rsi += rdi
} rsi += c_1 * rdi

rax = *(~) + rip =: c_2
*(rax + 8 * rsi) = rdx

(c_2 + 8 * (c_1 * rdi + rsi)) = rdx

```
set_C_:
    mov     0x600c00(,%rdi,8),%rax
    mov     %rdx,(%rax,%rsi,8)
    retq
```

purple

rax = *(~ + 8 * rdi)
*(rax + 8 * rsi) = rdx

↖ location of nested array

rdi	i
rsi	j
rdx	val

x86-64

3. ***This Problem is a Pain in My Big Endian (42 points):*** Below we show the disassembled function *func0()* – compiled on an ~~ia32~~ machine. The function reads one integer, using scanf(). Your job is to ***provide the input to scanf() that will result in this function returning the value 42.***

Consider the following C code:

Size 24

```
struct node_t {
  short key;
  union {
    long val;
    char label[8];
  } payload;
  struct node_t * next;
};

struct node_t * a[8];

int hash(int key)
{
  return (key&7)^((key>>3)&7);
}

void search(int key, int k)
{
  int i=hash(key);
  struct node_t *ptr;
  int j=0;

  ptr=a[i];
  while (ptr)
    {
      if (ptr->key==key && j==k)
        printf("%s", ptr->payload.label);
      j++;
      ptr=ptr->next;
    }
}
```

(231 & 7) ^ ((231 >> 3) & 7)
= 7 ^ (28 & 7)
= 7 ^ 4
= 3

0x600fa0 + 3 * 8B = 0x600fa0 + 24B
= 0x600fa0 + 0x18
= 0x600fb8

4f 64 69 6e 0a
O d i n \n

You need to figure out what is printed out when the following function call is performed:

search(231, 4)

The next three pages contain everything you need to solve this problem. Fill in all blanks on the answer sheet for partial credit.

255 − 231 = 24 = 8+16 = 2^3+2^4

1110 0111 (231) & 0111 (7) = 0111 (7)

0001 1100 (231 >> 3) & 0111 (7) = 0100 (4)

0111 (7) ^ 0100 (4) = 0011 (3)

24 = 16+8

Here is some gdb interaction which should prove useful – a breakpoint is set after the invocation of the function search:

```
(gdb) break *0x40097d
Breakpoint 1 at 0x40097d
(gdb) run
Starting program

Breakpoint 1, 0x00000000004009d7 in main ()
(gdb) print &a
$1 = (<data variable, no debug info> *) 0x600fa0

(gdb) x/64bx 0x600fa0
0x600fa0: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x600fa8: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x600fb0: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x600fb8: 0x10    0x13    0x60    0x00    0x00    0x00    0x00    0x00
0x600fc0: 0x50    0x13    0x60    0x00    0x00    0x00    0x00    0x00
0x600fc8: 0x30    0x13    0x60    0x00    0x00    0x00    0x00    0x00
0x600fd0: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x600fd8: 0x70    0x13    0x60    0x00    0x00    0x00    0x00    0x00

(gdb) x/512bx 0x601200
0x601200:    0xd0    0x11    0x60    0x00    0x00    0x00    0x00    0x00
0x601208:    0x21    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x601210:    0xe9    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x601218:    0x52    0x61    0x0a    0x00    0x00    0x00    0x00    0x00
0x601220:    0x50    0x11    0x60    0x00    0x00    0x00    0x00    0x00
0x601228:    0x21    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x601230:    0xe9    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x601238:    0x4f    0x73    0x69    0x72    0x69    0x73    0x0a    0x00
0x601240:    0x10    0x12    0x60    0x00    0x00    0x00    0x00    0x00
0x601248:    0x21    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x601250:    0xea    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x601258:    0x41    0x6e    0x75    0x62    0x69    0x73    0x0a    0x00
0x601260:    0xb0    0x11    0x60    0x00    0x00    0x00    0x00    0x00
0x601268:    0x21    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x601270:    0xe7    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x601278:    0x4f    0x64    0x69    0x6e    0x0a    0x00    0x00    0x00
0x601280:    0xf0    0x11    0x60    0x00    0x00    0x00    0x00    0x00
0x601288:    0x21    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x601290:    0xe8    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x601298:    0x42    0x61    0x73    0x74    0x0a    0x00    0x00    0x00
0x6012a0:    0x90    0x11    0x60    0x00    0x00    0x00    0x00    0x00
0x6012a8:    0x21    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x6012b0:    0xe8    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x6012b8:    0x54    0x68    0x6f    0x72    0x0a    0x00    0x00    0x00
0x6012c0:    0x90    0x12    0x60    0x00    0x00    0x00    0x00    0x00
```

print out of bounds

```
0x6012c8:  0x21  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x6012d0:  0xe9  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x6012d8:  0x4c  0x6f  0x6b  0x69  0x0a  0x00  0x00  0x00
0x6012e0:  0x30  0x12  0x60  0x00  0x00  0x00  0x00  0x00
0x6012e8:  0x21  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x6012f0:  0xea  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x6012f8:  0x46  0x72  0x65  0x79  0x6a  0x61  0x0a  0x00
0x601300:  0x50  0x12  0x60  0x00  0x00  0x00  0x00  0x00
0x601308:  0x21  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x601310:  0xe7  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x601318:  0x42  0x61  0x6c  0x64  0x75  0x72  0x0a  0x00
0x601320:  0x70  0x12  0x60  0x00  0x00  0x00  0x00  0x00
0x601328:  0x21  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x601330:  0xe8  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x601338:  0x48  0x65  0x6c  0x0a  0x00  0x00  0x00  0x00
0x601340:  0xb0  0x12  0x60  0x00  0x00  0x00  0x00  0x00
0x601348:  0x21  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x601350:  0xe9  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x601358:  0x46  0x72  0x65  0x79  0x72  0x0a  0x00  0x00
0x601360:  0xd0  0x12  0x60  0x00  0x00  0x00  0x00  0x00
0x601368:  0x21  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x601370:  0xea  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x601378:  0x46  0x72  0x69  0x67  0x67  0x0a  0x00  0x00
0x601380:  0xf0  0x12  0x60  0x00  0x00  0x00  0x00  0x00
0x601388:  0x81  0x0c  0x02  0x00  0x00  0x00  0x00  0x00
```

ASCII Table

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

58/58

28/42

Answer Sheet

Name:

1. Circle the correct responses:

(N) Y

(Y) N

Y (N)

(Y) N

28

2. Fill in the three blanks below with the name of a color (i.e. green, orange, or purple) corresponding to the correct function call.

set **A** : set orange

set **B** : set green

30

set **C** : set purple

3. Fill in all blanks below

The value of variable *i* in function search():

3

The value of *a[i]* in function search():

0x600fb8

28

The value printed by the printf statement in function search():

Odin\n

⌘