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 – do not cheat.

NAME: ____

ID: ____

Problem 1: ___________

Problem 2: ___________

Problem 3: ___________

Problem 4: ___________

Total: ________________

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

   int x = func();
   int y = func2();
   unsigned ux = (unsigned) x;

   Where func() and func2() could return any arbitrary value.

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

	Always True?	
a. (x >> 1) < 0 ⇒ ux > TMax	***Y***	N
b. ((x > 0) && (y > 0) && ((x – y) < 0)) ⇒ ((y - x) > 0)	***Y***	N
c. ((x & 255) == 255) ⇒ (((x << 24) | (x << 16) **|** (x << 8) | x) == (unsigned) -1)	***Y***	N
d. ((x * y) < 0) ⇒ ((x | y) < 0)	Y	***N***

   Note that UMax and TMax are as defined in class.

   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 (12 points)***: Evaluate the following statement:

```
(~(4|3)+2)^1
```

What is the resulting value? ***`-5`***

3. ***This Problem is a Pain in My Big Endian (30 points):***

Consider the following C declaration:

```
char * mysticarray[20];
```

This array is initialized by further C code, and then the following statement is executed:

```
printf("%s\n", mysticarray[9]);
```

Your job is to figure out what is printed by this printf statement. The next two pages contain everything you need to solve this problem. ***Fill in all blanks*** on the answer sheet for credit.

The array is a multilevel array – not nested – so it is an array of pointers. From gdb interaction below - the first level array starts at address `0x7fffffffe1d0`. If my offset is 9 - that is 9*8 or 72 (decimal) offset from that base address. That is 0x7fffffffe218. That address contains pointer 0x4006ae. That pointer directs us to a string containing chars: 0x62 0x72 0x6f 0x6e 0x7a 0x65 0x00. In ASCII - this is "bronze".

Final answer: bronze

Here is some gdb interaction (on an x86-64 machine) which should prove useful – a breakpoint is set after the invocation of the function search:

```
(gdb) print &mysticarray
$1 = (char *(*)[20]) 0x7fffffffe1d0

(gdb) x/256bx 0x4006a0
0x4006a0:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x4006a8:    0x62    0x72    0x61    0x73    0x73    0x00    0x62    0x72
0x4006b0:    0x6f    0x6e    0x7a    0x65    0x00    0x63    0x6f    0x70
0x4006b8:    0x70    0x65    0x72    0x00    0x69    0x72    0x6f    0x6e
0x4006c0:    0x00    0x6e    0x69    0x63    0x6b    0x65    0x6c    0x00
0x4006c8:    0x74    0x69    0x6e    0x00    0x61    0x6c    0x75    0x6d
0x4006d0:    0x69    0x6e    0x75    0x6d    0x00    0x73    0x69    0x6c
0x4006d8:    0x76    0x65    0x72    0x00    0x67    0x6f    0x6c    0x64
0x4006e0:    0x00    0x74    0x69    0x74    0x61    0x6e    0x69    0x75
0x4006e8:    0x6d    0x00    0x73    0x74    0x65    0x65    0x6c    0x00
0x4006f0:    0x70    0x6c    0x61    0x74    0x69    0x6e    0x75    0x6d
0x4006f8:    0x00    0x63    0x68    0x72    0x6f    0x6d    0x69    0x75
0x400700:    0x6d    0x00    0x76    0x61    0x6e    0x61    0x64    0x69
0x400708:    0x75    0x6d    0x00    0x70    0x61    0x6c    0x6c    0x61
0x400710:    0x64    0x69    0x75    0x6d    0x00    0x7a    0x69    0x6e
0x400718:    0x63    0x00    0x63    0x6f    0x62    0x61    0x6c    0x74
0x400720:    0x00    0x72    0x68    0x6f    0x64    0x69    0x75    0x6d
0x400728:    0x00    0x00    0x00    0x00    0x01    0x1b    0x03    0x3b
0x400730:    0x20    0x00    0x00    0x00    0x03    0x00    0x00    0x00
0x400738:    0xd8    0xfd    0xff    0xff    0x3c    0x00    0x00    0x00
0x400740:    0x84    0xfe    0xff    0xff    0x64    0x00    0x00    0x00
0x400748:    0x94    0xfe    0xff    0xff    0x7c    0x00    0x00    0x00
0x400750:    0x14    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x400758:    0x01    0x7a    0x52    0x00    0x01    0x78    0x10    0x01
0x400760:    0x1b    0x0c    0x07    0x08    0x90    0x01    0x00    0x00
0x400768:    0x24    0x00    0x00    0x00    0x1c    0x00    0x00    0x00
0x400770:    0x94    0xfd    0xff    0xff    0xa6    0x00    0x00    0x00
0x400778:    0x00    0x41    0x0e    0x10    0x86    0x02    0x43    0x0d
0x400780:    0x06    0x5e    0x83    0x03    0x02    0x83    0x0c    0x07
0x400788:    0x08    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x400790:    0x14    0x00    0x00    0x00    0x44    0x00    0x00    0x00
0x400798:    0x18    0xfe    0xff    0xff    0x02    0x00    0x00    0x00
```

```
(gdb) x/256bx 0x7fffffffe1d0
0x7fffffffe1d0: 0xae    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe1d8: 0x1a    0x07    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe1e0: 0xe1    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe1e8: 0xd5    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe1f0: 0xc8    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe1f8: 0xd5    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe200: 0xea    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe208: 0xf9    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe210: 0x15    0x07    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe218: 0xae    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe220: 0x0b    0x07    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe228: 0x02    0x07    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe230: 0x0b    0x07    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe238: 0xd5    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe240: 0xc8    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe248: 0xc1    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe250: 0xcc    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe258: 0xa8    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe260: 0x1a    0x07    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe268: 0xea    0x06    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe270: 0x00    0x00    0x00    0x00    0x14    0x00    0x00    0x00
0x7fffffffe278: 0x20    0x04    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe280: 0x70    0xe3    0xff    0xff    0xff    0x7f    0x00    0x00
0x7fffffffe288: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe290: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe298: 0x1d    0xed    0x41    0x24    0x33    0x00    0x00    0x00
0x7fffffffe2a0: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe2a8: 0x78    0xe3    0xff    0xff    0xff    0x7f    0x00    0x00
0x7fffffffe2b0: 0x00    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffffffe2b8: 0x04    0x05    0x40    0x00    0x00    0x00    0x00    0x00
0x7fffffffe2c0: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe2c8: 0x55    0x0a    0xfa    0x42    0x2a    0xf0    0x7e    0x68
```

4. ***Now That's a Switch (30 points)***: Consider the following code fragment:

```
short table[8][8];

short func0 (int i, int j, int a)
{
  switch(a){
     // CODE NOT SHOWN…
  }

  fprintf (stderr,"Lookup 0x%hx\n", table[i][j]);

  return a;
}
```

The switch statement code is not shown above, but here the complete assembly code (x86-64) for function *func0*:

```
0000000000400544 <func0>:
  400544:  55                      push   %rbp
  400545:  48 89 e5                mov    %rsp,%rbp
  400548:  48 83 ec 10             sub    $0x10,%rsp
  40054c:  89 7d fc                mov    %edi,-0x4(%rbp)
  40054f:  89 75 f8                mov    %esi,-0x8(%rbp)
  400552:  89 55 f4                mov    %edx,-0xc(%rbp)
  400555:  8b 45 f4                mov    -0xc(%rbp),%eax
  400558:  83 e8 64                sub    $0x64,%eax
  40055b:  83 f8 06                cmp    $0x6,%eax
  40055e:  77 34                   ja     400594 <func0+0x50>
  400560:  89 c0                   mov    %eax,%eax
  400562:  48 8b 04 c5 18 08 40    mov    0x400818(,%rax,8),%rax
  400569:  00
  40056a:  ff e0                   jmpq   *%rax
  40056c:  83 45 fc 01             addl   $0x1,-0x4(%rbp)
  400570:  83 45 f8 01             addl   $0x1,-0x8(%rbp)
  400574:  eb 2c                   jmp    4005a2 <func0+0x5e>
  400576:  83 45 fc 02             addl   $0x2,-0x4(%rbp)
  40057a:  83 45 f8 02             addl   $0x2,-0x8(%rbp)
  40057e:  eb 22                   jmp    4005a2 <func0+0x5e>
  400580:  83 45 fc 01             addl   $0x1,-0x4(%rbp)
  400584:  83 45 f8 02             addl   $0x2,-0x8(%rbp)
  400588:  eb 18                   jmp    4005a2 <func0+0x5e>
  40058a:  83 45 fc 02             addl   $0x2,-0x4(%rbp)
  40058e:  83 45 f8 01             addl   $0x1,-0x8(%rbp)
  400592:  eb 0e                   jmp    4005a2 <func0+0x5e>
  400594:  c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)
```

```
40059b:  c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%rbp)
4005a2:  8b 55 fc                mov    -0x4(%rbp),%edx
4005a5:  8b 45 f8                mov    -0x8(%rbp),%eax
4005a8:  48 98                   cltq
4005aa:  48 63 d2                movslq %edx,%rdx
4005ad:  48 c1 e2 03             shl    $0x3,%rdx
4005b1:  48 8d 04 02             lea    (%rdx,%rax,1),%rax
4005b5:  0f b7 84 00 60 0b 60    movzwl 0x600b60(%rax,%rax,1),%eax
4005bc:  00
4005bd:  0f bf d0                movswl %ax,%edx
4005c0:  b9 08 08 40 00          mov    $0x400808,%ecx
4005c5:  48 8b 05 74 05 20 00    mov    0x200574(%rip),%rax
4005cc:  48 89 ce                mov    %rcx,%rsi
4005cf:  48 89 c7                mov    %rax,%rdi
4005d2:  b8 00 00 00 00          mov    $0x0,%eax
4005d7:  e8 74 fe ff ff          callq  400450 <fprintf@plt>
4005dc:  8b 45 f4                mov    -0xc(%rbp),%eax
4005df:  c9                      leaveq
4005e0:  c3                      retq
```

The movzwl instruction at address 0x4005b5 sets the value of %eax to table[i][j]. This will ultimately be printed out by the fprintf() invocation at address 0x4005d7. At the time of the call to func0, the value of register rdx is 103 (in decimal). Here is some further interaction with gdb:

```
(gdb) x/104bx 0x400800
0x400800:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x400808:     0x4c    0x6f    0x6f    0x6b    0x75    0x70    0x20    0x30
0x400810:     0x78    0x25    0x68    0x78    0x0a    0x00    0x00    0x00
0x400818:     0x6c    0x05    0x40    0x00    0x00    0x00    0x00    0x00
0x400820:     0x94    0x05    0x40    0x00    0x00    0x00    0x00    0x00
0x400828:     0x76    0x05    0x40    0x00    0x00    0x00    0x00    0x00
0x400830:     0x80    0x05    0x40    0x00    0x00    0x00    0x00    0x00
0x400838:     0x84    0x05    0x40    0x00    0x00    0x00    0x00    0x00
0x400840:     0x94    0x05    0x40    0x00    0x00    0x00    0x00    0x00
0x400848:     0x8a    0x05    0x40    0x00    0x00    0x00    0x00    0x00
0x400850:     0x74    0x65    0x73    0x74    0x69    0x6e    0x67    0x20
0x400858:     0x20    0x25    0x68    0x69    0x0a    0x00    0x00    0x00
0x400860:     0x01    0x1b    0x03    0x3b    0x2c    0x00    0x00    0x00

(gdb) x/128b 0x600b60
0x600b60:    0x67    0x45    0xc6    0x23    0x69    0x98    0x73    0x48
0x600b68:    0x51    0xdc    0xff    0x5c    0x4a    0x94    0xec    0x58
0x600b70:    0x29    0x1f    0xcd    0x7c    0xba    0x58    0xab    0xd7
0x600b78:    0xf2    0x41    0xfb    0x1e    0xe3    0xa9    0x46    0xe1
0x600b80:    0x7c    0x00    0xc2    0x62    0x54    0x08    0xf8    0x27
0x600b88:    0x1b    0x23    0xe8    0xe9    0xe7    0xcd    0x8d    0x43
0x600b90:    0x76    0x0f    0x5a    0x25    0x2e    0xf9    0x63    0x72
```

```
0x600b98:    0x33    0xc2    0x9f    0xd7    0xc9    0xc4    0x9a    0x07
0x600ba0:    0x66    0xfb    0x32    0x5d    0x0d    0x50    0xb7    0xd7
0x600ba8:    0x31    0xba    0x58    0xe4    0xa3    0x30    0x5a    0xd9
0x600bb0:    0x25    0x61    0x5d    0x89    0x05    0xb1    0x17    0xa3
0x600bb8:    0x58    0xa8    0xe9    0x5a    0x5e    0x84    0xd4    0xa8
0x600bc0:    0xab    0xbd    0xb2    0x8c    0xcd    0xd0    0xc6    0xe0
0x600bc8:    0x9b    0x76    0xb4    0x9e    0x54    0x24    0x11    0x86
0x600bd0:    0x0e    0xc4    0x82    0x1d    0x74    0xf8    0x41    0x86
0x600bd8:    0x21    0xf5    0x3d    0xbd    0xdc    0x8d    0x87    0xf0
```

(some bolding and italics added above to help you search for specific hex values that may be useful)

In one invocation of func0, the value printed is:

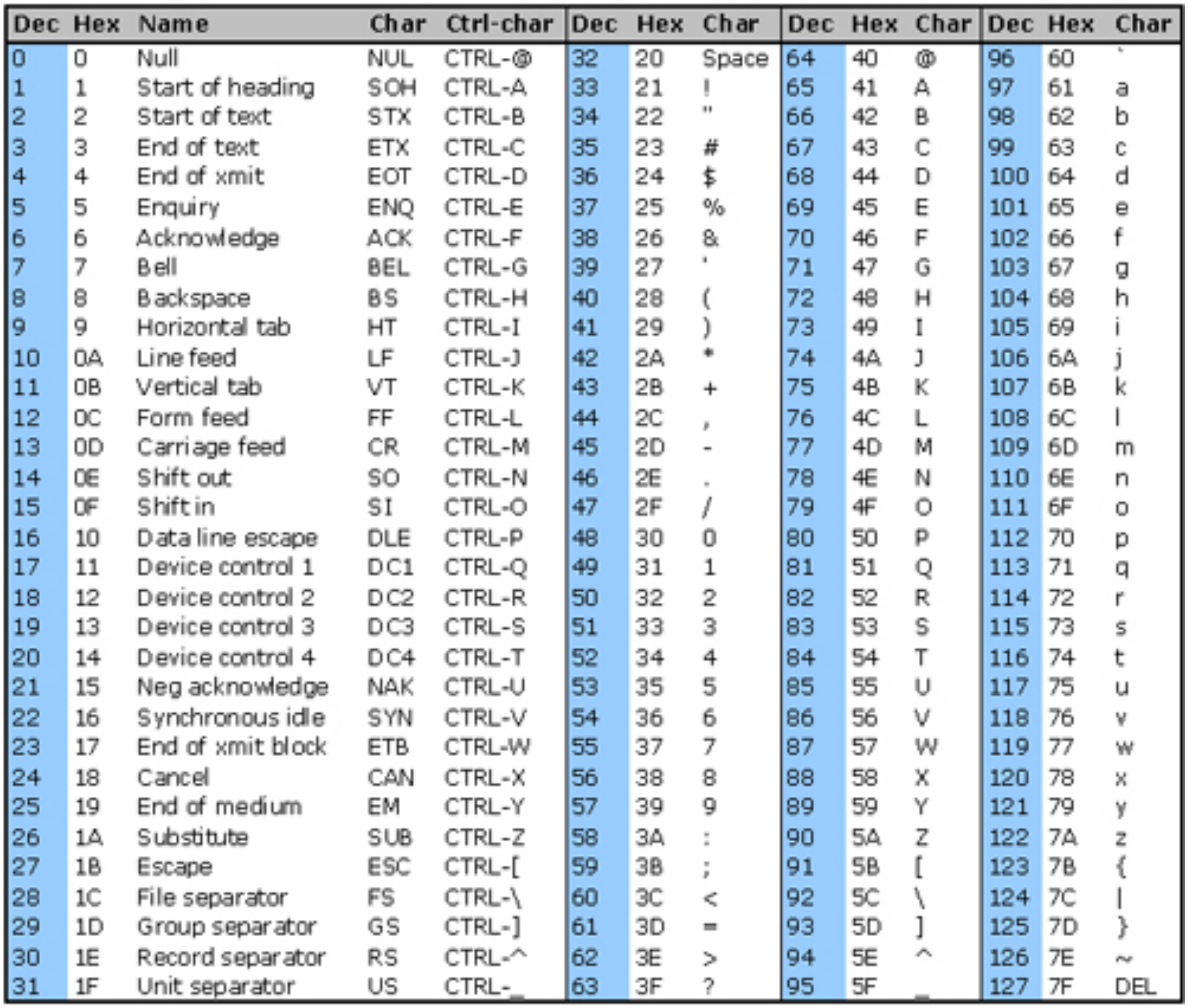
```
Lookup 0x231b
```

What are the values of *i* and *j* ***that are passed as parameters to func0***?

If A==103, then the jump table setup code will subtract 100 from this, and then multiply 3 * 8 to offset the jump table. This would be address 0x400818 + 0x18 = 0x400830. In the jump table, that entry points to address 0x400580. If you look at that code, and trace back to the input params to the function, you will see it does i+=1 and j+=2.

So now with that in mind, let's go from the answer at the end – Lookup 0x231b. That answer is based off of the table, and is at address 0x600b88. The base address of the table is 0x600b60. So the displacement here is 0x28 or 40 in decimal. The displacement equation for a nested array is i*C*K + j*K. Here, K=2 (short) and C=8 (8 cols). So simplifying, we would have i*8+j=20. There are multiple solutions of i and j that could work here – but let's take i=2 and j=4. If we now consider the impact of the switch statement (i+=1 and j+=2), that means the original values of i and j (upon entry to the function func0) would have been 1 and 2 respectively.

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

Answer Sheet

Name:______________________________

1. Circle the correct responses:

Y N

Y N

Y N

Y N

2. Resulting value: ___________________

3. Fill in all blanks below:

What is the starting address of the string at `mysticarray[9]`:____________________

The string printed by the printf statement: ______________________

4. Fill in all blanks below:

What value is written to %rax by the instruction at address 0x4005b5? ______________

The value of %rax at the execution of the jmpq at address 0x40056a: ______________

The value of i: ______________ and j: ________________ at the invocation of the func0 function.

⌘