CS 208: Introduction to Computer Systems Name: _____
Practice Exam 1
January 29, 2023

This is a closed-everything-other-than-your-brain **practice exam**. No books, no notes, no electronic devices. Please put your name on each sheet of paper you use that is not already stapled with the **practice exam**. There are two questions that start with "what happens when you compile and run the following C code?" Answers to such a question might include: "it doesn't compile for reason X" or "it prints text Y" or "it crashes for reason Z".

1. (4 points) Let N = 47. Write N in hexadecimal and binary.

2.(1 point) Let N = 0b110110010011. Write N in hexadecimal.

3. (3 points) Let N = 0xFFFFFF32. Write N in decimal. (hint: two's complement at work here)

4. (3 points) The character 𝟏 has the Unicode codepoint U+127F. Write this character's UTF-8 encoding as a sequence of one-byte hexadecimal numbers.

Code point ↔ UTF-8 conversion

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0xxxxxxx			
U+0080	U+07FF	110xxxxx	10xxxxxx		
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
U+10000	[b]U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

5. (3 points) `E2 9B A9` is the hex UTF-8 for the character ⛩ (aka Shinto Shrine). Write this character's Unicode codepoint.

6. (14 points) Suppose the C compiler has placed these variables at the specified address on a 64-bit big-endian computer:

- a 32-bit integer (`int num`) at address 0x62
- an array of three pointers `char *arr[3]` at address 0x47
- an `int` pointer (`int *ptr`) at address 0x6F

64-bit big-endian machine (pointers are 64-bits wide)								
address								
0x27	44	65	61	72	20	53	70	75
0x2F	74	6E	69	6B	00	61	6E	73
0x37	77	65	72	00	47	68	6F	73
0x3F	74	69	6E	67	00	42	56	67
0x47	00	00	00	00	00	00	00	27
0x4F	00	00	00	00	00	00	00	3B
0x57	00	00	00	00	00	00	00	34
0x5F	6F	69	00	FF	FF	FF	E7	39
0x67	00	00	00	3E	62	61	65	00
0x6F	00	00	00	00	00	00	00	67

For each of the following printf statements, what is the output?

(a) `printf("%s", arr[0]);`

(b) `printf("0x%X", num);`

(c) `printf("%d", num);`

(d) `printf("%p", arr[1]);`

(e) `printf("%p", &ptr);`

(f) `printf("0x%X", *ptr);`

(g) `printf("%c", *(arr[2] + 3));`

(ASCII chart on the next page)

Dec	Oct	Hex	C	Dec	Oct	Hex	C	Dec	Oct	Hex	C	Dec	Oct	Hex	C
0	0	0	^@	32	40	20		64	100	40	@	96	140	60	`
1	1	1	^A	33	41	21	!	65	101	41	A	97	141	61	a
2	2	2	^B	34	42	22	"	66	102	42	B	98	142	62	b
3	3	3	^C	35	43	23	#	67	103	43	C	99	143	63	c
4	4	4	^D	36	44	24	$	68	104	44	D	100	144	64	d
5	5	5	^E	37	45	25	%	69	105	45	E	101	145	65	e
6	6	6	^F	38	46	26	&	70	106	46	F	102	146	66	f
7	7	7	^G	39	47	27	'	71	107	47	G	103	147	67	g
8	10	8	^H	40	50	28	(	72	110	48	H	104	150	68	h
9	11	9	^I	41	51	29	)	73	111	49	I	105	151	69	i
10	12	a	^J	42	52	2a	*	74	112	4a	J	106	152	6a	j
11	13	b	^K	43	53	2b	+	75	113	4b	K	107	153	6b	k
12	14	c	^L	44	54	2c	,	76	114	4c	L	108	154	6c	l
13	15	d	^M	45	55	2d	-	77	115	4d	M	109	155	6d	m
14	16	e	^N	46	56	2e	.	78	116	4e	N	110	156	6e	n
15	17	f	^O	47	57	2f	/	79	117	4f	O	111	157	6f	o
16	20	10	^P	48	60	30	0	80	120	50	P	112	160	70	p
17	21	11	^Q	49	61	31	1	81	121	51	Q	113	161	71	q
18	22	12	^R	50	62	32	2	82	122	52	R	114	162	72	r
19	23	13	^S	51	63	33	3	83	123	53	S	115	163	73	s
20	24	14	^T	52	64	34	4	84	124	54	T	116	164	74	t
21	25	15	^U	53	65	35	5	85	125	55	U	117	165	75	u
22	26	16	^V	54	66	36	6	86	126	56	V	118	166	76	v
23	27	17	^W	55	67	37	7	87	127	57	W	119	167	77	w
24	30	18	^X	56	70	38	8	88	130	58	X	120	170	78	x
25	31	19	^Y	57	71	39	9	89	131	59	Y	121	171	79	y
26	32	1a	^Z	58	72	3a	:	90	132	5a	Z	122	172	7a	z
27	33	1b	^[	59	73	3b	;	91	133	5b	[	123	173	7b	{
28	34	1c	^\	60	74	3c	<	92	134	5c	\	124	174	7c	|
29	35	1d	^]	61	75	3d	=	93	135	5d	]	125	175	7d	}
30	36	1e	^^	62	76	3e	>	94	136	5e	^	126	176	7e	~
31	37	1f	^_	63	77	3f	?	95	137	5f	_	127	177	7f	

7. (4 points) What happens when you compile and run the following C code?

```
char buffer[20];
strcpy(buffer, "farewell, neverland");
char *k = buffer;
for (int i = 0; i < 3; i++) {
      k += 2;
      printf("%c\n", *k);
}
printf("%s\n", k);
```

8. (5 points) Let a = 6 and b = 11. What are the results of the following print statements?

(a) `printf("0x%X\n", a | ~b);`

(b) `printf("%d\n", a | ~b);`

(c) `printf("%d\n", a & b);`

(d) `printf("0x%X\n", a ^ b);`

(e) `printf("%d\n", ((a >> 1) | (b << 1)) & 15)`

9. (5 points) What happens when you compile and run the following C code?

```
char *pointers[3];
pointers[0] = "name";
pointers[1] = "chapter";
pointers[2] = "temptation";

char first = *pointers[0];
char *last_p = pointers[2];

printf("%c\n", first);
printf("%s\n", last_p);

for (int i = 0; i < 3; i++) {
     first++;
     last_p++;
}

printf("%c\n", first);
printf("%s\n", last_p);
printf("%s\n", pointers[2]);
```

10. (3 points) Suppose that I want to extract the byte ***n*** from a 32-bit int ***x*** that has been pre-initialized by earlier code. Write C code that will check for valid input, and if valid, returns the byte. Note: when *n = 0*, return the rightmost byte (endian type does not matter).
Bonus: assuming the input is valid, is it possible to return the byte using only bitwise operations?

```
getByte (int x, int n) {
     // TODO
}
```