NOT SCARY BINARY - EXERCISES

NOT SCARY PRACTICE #1

YOU CAN DO THIS!

BINARY TO DECIMAL

Translate the following to decimal:

A. 10011011

B. 01101100

C. 10101010

D. 01110111

E. 11111111

DECIMAL TO BINARY

Convert these numbers to binary:

A. 250

B. 127

C. 99

D. 131

E. 197

BINARY IS NOT JUST FOR NUMBERS

ASCII is a way of encoding letters as numeric values. The table shown here shows the numeric code for each character. Using this table, translate the message below:

```
01000010 01101001 01101110 01100001
01110010 01111001 00100000 01101001
01110011 00100000 01101110 01101111
01110100 00100000 01110011 01100011
01100001 01110010 01111001 00100001
```

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

NOT SCARY PRACTICE #1

THE ANSWERS

BINARY TO DECIMAL

Translate the following to decimal:

A. 10011011 = 155

B. 01101100 = 108

C. 10101010 = 170

D. 01110111 = 119

E. 11111111 = 255

DECIMAL TO BINARY

Convert these numbers to binary:

A. 250 = 11111010

B. 127 = 01111111

C. 99 = 01100011

D. 131 = 10000011

E. 197 = 11000101

BINARY IS NOT JUST FOR NUMBERS

ASCII is a way of encoding letters as numeric values. The table shown here shows the numeric code for each character. Using this table, translate the message below:

```
01000010 01101001 01101110 01100001
01110010 01111001 00100000 01101001
01110011 00100000 01101110 01101111
01110100 00100000 01110011 01100011
01100001 01110010 01111001 00100001
```

Binary is not scary!

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

NOT SCARY PRACTICE #2

SIGN ME UP!

MORE CONVERSION

Provide both the unsigned and signed values for the binary below:

A. `10000000`

B. `01110000`

C. `10000111`

D. `11110000`

E. `11111111`

HARD LIMITS

IPv6 uses 128-bit addresses. What is the maximum number of unique IP addresses that can be represented?

DOS partition tables use 32-bit unsigned values to store the number of sectors in the partition.

A. What is the maximum number of sectors in a partition?
B. If each sector is 512 bytes, what is the maximum partition size?

MY RETIREMENT PLAN

Unix traditionally stores timestamps as 32-bit signed integers representing the number of seconds from Jan 1, 1970. If there are 86400 seconds per day and 365 days in a year, what is the largest year that can be represented?

NOT SCARY PRACTICE #2

THE ANSWERS

MORE CONVERSION

	Unsigned:	Signed:
A. 10000000	128	-128
B. 01110000	70	70
C. 10000111	135	-121
D. 11110000	240	-16
E. 11111111	255	-1

HARD LIMITS

IPv6 uses 128-bit addresses. What is the maximum number of unique IP addresses that can be represented?

$2^{128} - 1$ (more than 3.4 x 10^{38} unique addresses)

DOS partition tables use 32-bit unsigned values to store the number of sectors in the partition.

A. What is the maximum number of sectors in a partition?

B. If each sector is 512 bytes, what is the maximum partition size?

$2^{32} - 1 = 4{,}294{,}967{,}295$ sectors

Multiply by 512 bytes/sector = 2,147,483,647 KB, or 2TB

MY RETIREMENT PLAN

Unix traditionally stores timestamps as 32-bit signed integers representing the number of seconds from Jan 1, 1970. If there are 86400 seconds per day and 365 days in a year, what is the largest year that can be represented?

$2^{32-1} - 1$= 2,147,483,647 seconds

2,147,483,647 / (86400 * 365) = 68 years

Largest possible year: 1970 + 68 = 2038

NOT SCARY PRACTICE #3

BE SURE TO COVER ALL YOUR BASES

BINARY TO HEX

In a previous exercise, we translated the following ASCII bytes to "Binary is not scary!". Now convert them to hex.

```
01000010 01101001 01101110 01100001 01110010
01111001 00100000 01101001 01110011 00100000
01101110 01101111 01110100 00100000 01110011
01100011 01100001 01110010 01111001 00100001
```

HEX TO DECIMAL

When looking at low-level packet captures, IP addresses are sometimes displayed as four bytes in hex. Translate the following hex numbers as IP addresses (e.g. 10.37.128.16).

A. `0x0a01010a`

B. `0xc0a80a89`

C. `0xac100c63`

ASKING PERMISSION

Convert Unix permissions "`rw-r--r--`" to a set of three octal digits

What Unix permissions are represented by the octal string 711?

NOT SCARY PRACTICE #3

THE ANSWERS

BINARY TO HEX

In a previous exercise, we translated the following ASCII bytes to "Binary is not scary!". Now convert them to hex.

```
01000010 01101001 01101110 01100001 01110010 42 69 6E 61 72
01111001 00100000 01101001 01110011 00100000 79 20 69 73 20
01101110 01101111 01110100 00100000 01110011 6E 6F 74 20 73
01100011 01100001 01110010 01111001 00100001 63 61 72 79 21
```

HEX TO DECIMAL

When looking at low-level packet captures, IP addresses are sometimes displayed as four bytes in hex. Translate the following hex numbers as IP addresses (e.g. 10.37.128.16).

A. `0x0a01010a` **10.1.1.10**

B. `0xc0a80a89` **192.168.10.137**

C. `0xac100c63` **172.16.12.99**

ASKING PERMISSION

Convert Unix permissions "`rw-r--r--`" to a set of three octal digits

644

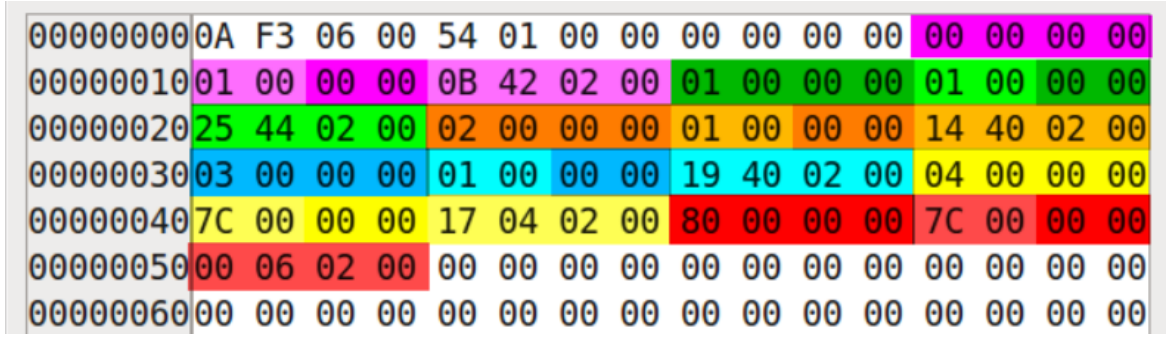
What Unix permissions are represented by the octal string 711?

rwx--x--x

NOT SCARY PRACTICE #4

WHICH END IS UP?

THAT'S THE EXTENT OF IT



00000000	0A	F3	06	00	54	01	00	00	00	00	00	00	00	00	00	00
00000010	01	00	00	00	0B	42	02	00	01	00	00	00	01	00	00	00
00000020	25	44	02	00	02	00	00	00	01	00	00	00	14	40	02	00
00000030	03	00	00	00	01	00	00	00	19	40	02	00	04	00	00	00
00000040	7C	00	00	00	17	04	02	00	80	00	00	00	7C	00	00	00
00000050	00	06	02	00	00	00	00	00	00	00	00	00	00	00	00	00
00000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

EXT4 extents consist of four little endian fields:

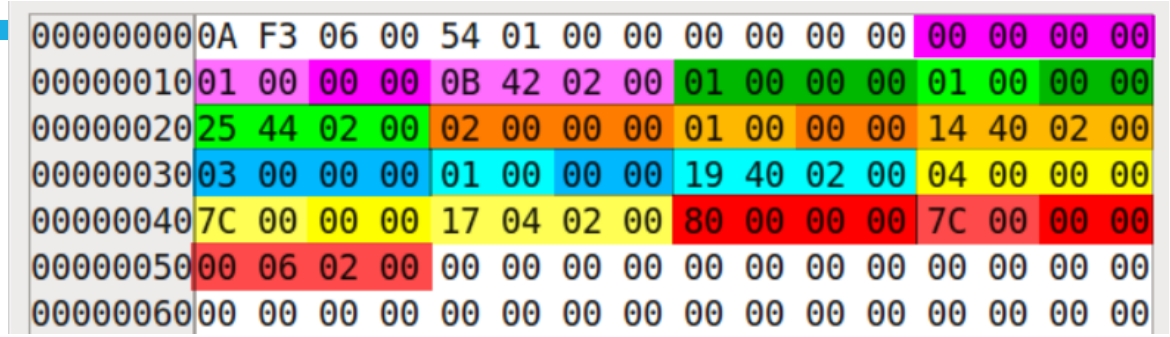
1. Four byte logical offset
2. Two bytes for the length of the extent in blocks
3. Two bytes for the upper 16 bits of the 48 bit starting block address
4. Four bytes for the lower 32 bits of the starting block address

There are six extents shown in the picture. Convert the bytes in each field from little-endian to big-endian, and then convert the values to decimal.

NOT SCARY PRACTICE #4

THE ANSWERS

THAT'S THE EXTENT OF IT



00000000	0A	F3	06	00	54	01	00	00	00	00	00	00	00	00	00	00
00000010	01	00	00	00	0B	42	02	00	01	00	00	00	01	00	00	00
00000020	25	44	02	00	02	00	00	00	01	00	00	00	14	40	02	00
00000030	03	00	00	00	01	00	00	00	19	40	02	00	04	00	00	00
00000040	7C	00	00	00	17	04	02	00	80	00	00	00	7C	00	00	00
00000050	00	06	02	00	00	00	00	00	00	00	00	00	00	00	00	00
00000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

1. Logical Offset: 0, Number of Blocks: 0x0001 = 1, Starting Block: 0x0000 ~ 0x0002420B = 147979
2. Logical Offset: 0x00000001 = 1, Number of Blocks: 0x0001 = 1, Starting Block: 0x0000 ~ 0x00024425 = 148517
3. Logical Offset: 0x00000002 = 2, Number of Blocks: 0x0001 = 1, Starting Block: 0x0000 ~ 0x00024014 = 147476
4. Logical Offset: 0x00000003 = 3, Number of Blocks: 0x0001 = 1, Starting Block: 0x0000 ~ 0x00024019 = 147481
5. Logical Offset: 0x00000004 = 4, Number of Blocks: 0x007C = 124, Starting Block: 0x0000 ~ 0x00020417 = 132119
6. Logical Offset: 0x00000080 = 128, Number of Blocks: 0x007C = 124, Starting Block: 0x0000 ~ 0x00020600 = 132608

NOT SCARY PRACTICE #5

SHIFTY PEOPLE WITH MASKS

HARD CIDR

CIDR notation represents a network mask as a number of bits. For example, "/24" means a 24-bit netmask, which can be represented as 0xFFFFFF00 where the upper 24 bits are all ones.

What are the hex netmasks for the following CIDR values?

A. /16

B. /20

C. /27

MY TIME IS PACKED

EXT4 uses 32-bit fields to represent the nanoseconds field in timestamps. You only need 30 bits to represent nanoseconds (bonus points if you can show why). So EXT4 uses the upper 30 bits for the nanoseconds and the lower two bits to extend the seconds field for the normal Unix epoch timestamp.

Below are four nanosecond fields from a file in EXT4 (represented in big-endian byte order). Extract the nanoseconds from the upper 30 bits as a hex value. If you have a calculator handy, convert the value to decimal.

1. `0xD86D8324`
2. `0x4A7502E4`
3. `0xEAEABC70`
4. `0xA125C840`

NOT SCARY PRACTICE #5

SHIFTY PEOPLE WITH MASKS

HARD CIDR

CIDR notation represents a network mask as a number of bits. For example, "/24" means a 24-bit netmask, which can be represented as 0xFFFFFF00 where the upper 24 bits are all ones.

What are the hex netmasks for the following CIDR values?

A. /16 0xFFFF0000

B. /20 1111 1111 1111 1111 1111 0000 0000 0000 = 0xFFFFF000

C. /27 1111 1111 1111 1111 1111 1111 1110 0000 = 0xFFFFFFE0

MY TIME IS PACKED

1. 0xD86D8324 = 11011000011011011000001100100100 >> 2 = 00110110000110110110000011001001
   = 0x361B60C9 (907763913 decimal)
2. 0x4A7502E4 = 01001010011101010000001011100100 >> 2 = 00010010100111010100000010111001
   = 0x129D40B9 (312295609 decimal)
3. 0xEAEABC70 = 11101010111010101011110001110000 >> 2 = 00111010101110101010111100011100
   = 0x3ABAAF1C (985313052 decimal)
4. 0xA125C840 = 10100001001001011100100001000000 >> 2 = 00101000010010010111001000010000
   = 0x28497210 (675901968 decimal)