Information Representation

How do we convey written ideas?

Example: How do we convey the concept of a greeting in words?

Information Representation

We might write one of several things...

Hello,

Hola,

Привет,

こんにちは,

Hallo,

Hallo,

...

Information Representation

Which is *almost* enough to know what it means and what it is...

English	Hello,
Spanish	Hola,
Russian	Привет,
Japanese	こんにちは,
????	Hallo,
????	Hallo,

Information Representation

But we sometimes need context to figure it out.

English	Hello,	my name is Donald.
Spanish	Hola,	me llamo Donald.
Russian	Привет,	меня зовут Дональд.
Japanese	こんにちは,	私の名前はドナルドです
????	Hallo,	*mein name* ist Donald.
????	Hallo,	*mijn naam* is Donald.

Information Representation

But we sometimes need context to figure it out.

English	Hello,	my name is Donald.
Spanish	Hola,	me llamo Donald.
Russian	Привет,	меня зовут Дональд.
Japanese	こんにちは,	私の名前はドナルドです
German	Hallo,	*mein name* ist Donald.
Dutch	Hallo,	*mijn naam* is Donald.

Information Representation

English	Hello,	my name is Donald.
Spanish	Hola,	me llamo Donald.
Russian	Привет,	меня зовут Дональд.
Japanese	こんにちは,	私の名前はドナルドです
German	Hallo,	*mein name* ist Donald.
Dutch	Hallo,	*mijn naam* is Donald.

Conclusion: With words, the Shape, structure, and context give us the information, not the characters themselves. In computing, we can't just look at the number and assume it means what we think it means. **We often need more context and understanding to get the value being described.**

~~Information~~ Representation
Data

Let's learn how to recognize and use the following things today!

Binary

Decimal

2's Complement

Hex

char

int

uintX_t

float

And more!

Data Representation:
Number Formats

Name	Base	Digits (written as a set)	"Definition"
Decimal *eg. 10, 524*	10	{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }	Each place value **x** represents **x**$*10^n$, where is ***n*** is the index of the value in the number (0 idx-ed).
Hexadecimal *eg. 0x11, 0x449, 0xF1*	16	{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }	Each place value **x** represents **x**$*16^n$, where is ***n*** is the index of the value in the number (0 idx-ed).
Binary *eg. 0b1101 0b0001*	2	{ 0, 1, }	Each place value **x** represents **x**$*2^n$, where is ***n*** is the index of the value in the number (0 idx-ed).

Data Representation:
Number Formats

Name	Base	Digits (written as a set)	"Definition"
Decimal *eg. 10, 524*	**10**	**{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }**	Each place value **x** represents **x**$*10^n$, where is ***n*** is the index of the value in the number (0 idx-ed).
Hexadecimal *eg. 0x11, 0x449, 0xF1*	**16**	**{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }**	Each place value **x** represents **x**$*16^n$, where is ***n*** is the index of the value in the number (0 idx-ed).
Binary *eg. 0b1101 0b0001*	**2**	**{ 0, 1, }**	Each place value **x** represents **x**$*2^n$, where is ***n*** is the index of the value in the number (0 idx-ed).

Did you notice? The number of digits in the digit set is equal to the base! What does this imply, and why is it like that?

Data Representation:
Number Formats

Name	Base	Digits (written as a set)	"Definition"
Decimal *eg. 10, 524*	10	{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }	Each place value **x** represents **x**$*10^n$, where is ***n*** is the index of the value in the number (0 idx-ed).
Hexadecimal *eg. 0x11, 0x449, 0xF1*	16	{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }	Each place value **x** represents **x**$*16^n$, where is ***n*** is the index of the value in the number (0 idx-ed).
Binary *eg. 0b1101 0b0001*	2	{ 0, 1, }	Each place value **x** represents **x**$*2^n$, where is ***n*** is the index of the value in the number (0 idx-ed).

Did you notice? Even if two numbers use the same digits, eg. 110 and 0b110, hex and binary have prefixes to indicate what the digits represent.

Data Representation:
Data Types (Containers)

Name	Width	Range	Signed?
char	8	-128 → 127 OR 0 → 255	unspecified
short	16	-32,768 → 32,767	yes
int	32	-2,147,483,648 → 2,147,483,647	yes
float	32	1.2E-38 → 3.4E+38 , Prec: 6	yes
double	64	2.3E-308 → 1.7E+308, Prec: 15	yes
long	64	-9223372036854775808 → 9223372036854775807	yes
uintX_t , $\{ X \in 2^N, 3 \leq N \leq 6 \}$	X	$0 \rightarrow 2^X - 1$	no

Data Representation:
Special Cases, etc.

Name	"Definition"
2's Complement	An interpretation of the binary format where given a binary number of N bits, instead of the highest bit N-1 having the value of 2^{N-1}, we say that it has the value -2^{N-1}. This gives us many useful properties and allows us to store negative values without requiring an extra symbol (the - symbol).
long, static, volatile, unsigned, etc.	We can append different keywords to certain data types to increase their width, increase their scope, change their accessibility, decide if they are 2'sC or not, and maybe other things (?)
IEEE format	An interpretation of the binary format where a binary number of N bits is divided into three groups: the **highest bit N-1** (or **sign bit**), some number of **exponent bits E**, and some number of **mantissa bits M**. Notice that 1 + E + M = N. We use these groups to store values of an equation, which generates a high precision fractional value.

Data Representation:
Special Cases, etc.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Data Representation

There are realistically too many examples to cover in a simple recitation, but let's try some basic ones that should help you get yourself ready to work in C.

Convert each of the following numbers from their starting base to the other two (i.e. dec → bin & hex).

Assume binary numbers are <u>not</u> in two's complement.

0b11010110

154

0xD4

0x45F1

0b11010110 10110101

Data Representation

There are realistically too many examples to cover in a simple recitation, but let's try some basic ones that should help you get yourself ready to work in C.

Invert the following numbers (neg to positive or vice versa), which are in two's complement, and then convert them to hex, and then decimal.

0b11010110

0b11010110 10110101

0b11111111 11111111 11111111 11111111

0b00010101

0b01111111

Data Representation

There are realistically too many examples to cover in a simple recitation, but let's try some basic ones that should help you get yourself ready to work in C.

Write out the bits that would be stored for each of the following type definitions.

char a = 0x54;

uint32_t b = 156;

int c = 0b11001001;

Challenge:

int oof = 0xF4A1 - 'z';

float ouch = 65.345

Data Representation

There are realistically too many examples to cover in a simple recitation, but let's try some basic ones that should help you get yourself ready to work in C.

Write out the bits that would be stored for each of the following type definitions.

int array[3] = { 123, 234, 345 };

uint8_t d = 400;

float e = 3645.4243

char name[10] = "Joe Biden";

Data Representation: Answers

There are realistically too many examples to cover in a simple recitation, but let's try some basic ones that should help you get yourself ready to work in C.

Convert each of the following numbers from their starting base to the other two (i.e. dec → bin & hex).

Assume binary numbers are not in two's complement.

0b11010110 : 0xD6, 214

154 : 0x9A, 0b10011010

0xD4 : 212, 11010100

0x45F1 : 17905, 0b10001011 1110001

0b11010110 10110101 : 54965, 0xD6B5

Data Representation: Answers

There are realistically too many examples to cover in a simple recitation, but let's try some basic ones that should help you get yourself ready to work in C.

Invert the following numbers (neg to positive or vice versa), which are in two's complement, and then convert them to hex, and then decimal.

0b11010110	: 0b00101010, 42
0b11010110 10110101	: 0b00101001 01001011, 10571
0b11111111 11111111 11111111 11111111	: 0b00000000 00000000 00000000 00000000, 1
0b00010101	: 0b11101011 , -21
0b01111111	: 0b10000001, -127

Data Representation: Answers

There are realistically too many examples to cover in a simple recitation, but let's try some basic ones that should help you get yourself ready to work in C.

Write out the bits that would be stored for each of the following type definitions.

char a = 0x54; : 01010100 //note the lack of 0b here; these are bits so they don't have a prefix

uint32_t b = 156; : 00000000 00000000 00000000 10011100

int c = 0b11001001; : 00000000 00000000 00000000 11001001

^^^ There is some argument here that certain compilers might automatically interpret this as a signed char, but according to what I know it is a literal, whose default type int (?) which means that the leading bit is a 0 after all. Try asking chat or compiling this one your own in a simple C program.

int oof = 0xF4A1 - 'z'; : 00000000 00000000 11110100 00100111

float ouch = 65.345 : 01000010 10000010 10110000 10100100

If you need help learning about Floating Point, you can use this calculator: https://www.h-schmidt.net/FloatConverter/IEEE754.html

Data Representation: Answers

There are realistically too many examples to cover in a simple recitation, but let's try some basic ones that should help you get yourself ready to work in C.

Write out the bits that would be stored for each of the following type definitions.

int array[3] = { 123, 234, 345 };

00000000 00000000 00000000 01111011 00000000 00000000 00000000 11101010 00000000 00000000 00000001 01011001

uint8_t d = 400; : 10010000 // note the overflow

float e = 3645.4243 : 01000101 01100011 11010110 11001010

char name[10] = "Joe Biden"; :

01001010 01101111 01100101 00100000 01000010 01101001 01100100 01100101 01101110