Information and Data

CPSC 231: Introduction to Computer Science for Computer Science Majors I
Spring 2021

Jonathan Hudson, Ph.D.
Instructor
Department of Computer Science
University of Calgary

Friday, 30 April 2021

UNIVERSITY OF CALGARY

What is Information?

Etymology: Latin, “to give form to” or “to form an idea of”

Definition: The state of being of an object or system of interest

What is Data?

Data: raw facts, representation of information, no context

Encoding: The translation of information into data

(Decoding the other direction

Data represents information

Information Processing

A change of information in any manner detectable by an observer

Using a computer?

Encode information into data	Process the data	Translate data back into information

Moral: computers process **data**, not information – **it is our responsibility to interpret the data correctly.**

Storing Data

All data in a computer is either a 0 or 1

Called a bit (binary digit)

Electrically, this is a switch that is either open or closed

Encoding schemes translate integers, real numbers, letters, pictures, … into bits

Boolean Data

- Has two possible values
  - False
  - True
- Easily encoded using a single bit
  - 0: False
  - 1: True

Integer Data

How do we represent the numbers 5, 24, or 367 using only ones and zeros?

Simplest idea:

`11111 = 5`

`11111 11111 11111 11111 1111 = 24`

Not practical for large integers!

Other ideas?

Number Systems

- Decimal (Base 10)
  - 10 distinct symbols (0,1,2,3,4,5,6,7,8,9)
  - Each digit is a factor of 10 larger than the digit to its right

- Examples:

```
5 = 5 x 1
24 = 2 x 10 + 4 x 1
367 = 3 x 100 + 6 x 10 + 7 x 1
```

UNIVERSITY OF CALGARY

Number Systems

- Decimal (Base 10)
    - 10 distinct symbols (0,1,2,3,4,5,6,7,8,9)
    - Each digit is a factor of 10 larger than the digit to its right

- Examples:

$5 = \mathbf{5} \times 10^{0}$

$24 = \mathbf{2} \times 10^{1} + \mathbf{4} \times 10^{0}$

$367 = \mathbf{3} \times 10^{2} + \mathbf{6} \times 10^{1} + \mathbf{7} \times 10^{0}$

Number Systems

THIS IS A POSITIONAL SYSTEM – THE POSITION WITHIN THE NUMBER IMPACTS THE FACTOR BY WHICH THE DIGIT IS MULTIPLIED.

CHOICE OF BASE 10 IS (SOMEWHAT) ARBITRARY – **CAN USE ANY INTEGER BASE >= 1**

NOTE: **THERE IS NOTHING SPECIAL ABOUT BASE 10** – IT'S JUST WHAT WE ARE USED TO!

Binary Data

Number Systems

Binary (Base 2)

- 2 distinct symbols (0,1)
- Each digit is a factor of 2 larger than the digit to its right

Base 10: hundreds, tens, ones

Base 2: eights, fours, twos, ones

Counting in Binary

```
   0   == 0
   1   == 1
  10   == 2
  11   == 3
 100   == 4
 101   == 5
 110   == 6
 111   == 7
1000   == 8
```

- You can see how when we have a single 1 in a column (ones, two, fours, eights) that it’s equivalent to that number in decimal (base 10)

Binary Numbers

- Consider the base 2 number 1001101_2

```
1:   ones (2^0)
0:   twos (2^1)
1:   fours (2^2)
1:   eights (2^3)
0:   sixteens (2^4)
0:   thirty-twos (2^5)
1:   sixty-fours (2^6)
```


UNIVERSITY OF CALGARY

Binary Numbers

- Consider the base 2 number 1001101_2

```
1:   ones (2^0)
0:   twos (2^1)
1:   fours (2^2)
1:   eights (2^3)
0:   sixteens (2^4)
0:   thirty-twos (2^5)
1:   sixty-fours (2^6)
```

- $1 \times 2^0 + 1 \times 2^2 + 1 \times 2^3 + 1 \times 2^6 = 1 + 4 + 8 + 64 = \mathbf{77_{10}}$ (base specified as a subscript)

Binary <-> Decimal

Binary to Decimal

- Convert 1111_2 to base 10:

- Convert 100010_2 to base 10:

- Convert 0_2 to base 10:

Binary to Decimal

- Convert 1111_2 to base 10:

$1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 = 1 + 2 + 4 + 8 = 15_{10}$

- Convert 100010_2 to base 10:

$1 \times 2^1 + 1 \times 2^5 = 2 + 32 = 34_{10}$

- Convert 0_2 to base 10:

0_{10}

The Division Algorithm

- Allows us to convert from Decimal to Binary

```
Let Q represent the number to convert
Repeat
   Divide Q by 2, recording the Quotient, Q, and the remainder, R
Until Q is 0
Read the remainders from bottom to top
```

- Divide by the base to which we want to convert (algorithm works for conversion from decimal to **any** base)

Decimal to Binary

- Convert 191_{10} to Binary:

```
191 / 2 = 95, remainder 1
95 / 2 = 47, remainder 1
47 / 2 = 23, remainder 1
23 / 2 = 11, remainder 1
11 / 2 = 5, remainder 1
5 / 2 = 2, remainder 1
2 / 2 = 1, remainder 0
1 / 2 = 0, remainder 1
```

Decimal to Binary

- Convert 191_{10} to Binary:

```
191 / 2 = 95, remainder 1
95 / 2 = 47, remainder 1
47 / 2 = 23, remainder 1
23 / 2 = 11, remainder 1
11 / 2 = 5, remainder 1
5 / 2 = 2, remainder 1
2 / 2 = 1, remainder 0
1 / 2 = 0, remainder 1
```

- Reading from bottom to top: $1011\ 1111_2$

- ***Check:*** *$1 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^7 = 1 + 2 + 4 + 8 + 16 + 32 + 128 = 191_{10}$*

Integer Data

Integer Data

- Base 10 integers can be represented using sequences of bits
  - Common sizes:
    - 8 bits (referred to as a **byte**)
    - 32 bits (referred to as a **word**)
    - 64 bits (referred to as a **double word** / **long**)
    - 16 bits (referred to as a **half word** / **short**)

  - N bits of data, each bit stores 2 things
  - 2 * 2 * 2 * … * 2   (N times)
  - 2^N different things can be represented by N bits (generally numbers 0 to $2^N - 1$)

Integer Data

- Base 10 integers can be represented using sequences of bits

- **Byte** [8 bits]: 0000 0000 – 1111 1111 (0 to 2^8 – 1)
- **Word** [32 bits]: 0 to 2^{32} – 1
- **Double word** (**long**) [64 bits]: 0 to 2^{64} – 1
- **Half word** (**short**) [16 bits]; 0 to 2^{16} - 1

Negative Numbers

- **Simple idea is called "Signed Magnitude".**
- Idea (SM byte): right-most 7 bits represent the magnitude, first **8^{th} bit represents the sign.**

- Example:

$65_{10} = 100\ 0001_2$

```
+65 as a byte:  0100 0001
-65 as a SM byte:  1100 0001
```

Negative Numbers

- **Simple idea is called "Signed Magnitude".**
- Idea (SM byte): right-most 7 bits represent the magnitude, first **8th bit represents the sign.**

- Example:

```
65₁₀ = 100 0001₂
```

65_{10} = $100\ 0001_2$

```
+65 as a byte:  0100 0001
-65 as a SM byte:  1100 0001
```

Losing 8th bit means we can only represent half as many positive numbers. We gain most back as negative numbers but…

what is 1000 0000?   -0?

Other Bases

Other Bases

- A number system can have any base
  - **Decimal: Base 10 (0,1,2,3,4,5,6,7,8,9)**
  - **Binary: Base 2 (0,1)**
  - Octal: Base 8 (0,1,2,3,4,5,6,7)
  - **Hexadecimal: Base 16 (0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f)**
  - Vigesimal: Base 20 (0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f,g,h,I,j)
  - Base 6 (0,1,2,3,4,5)
  - Any other number we choose…

Hexadecimal

- Convert 0xA1 to decimal:

- Convert 44 base 16 to decimal:

- Convert $CAFE_{16}$ to base 10:

Hexadecimal

- Convert 0xA1 to decimal:

  $\mathbf{A \times 16^1 + 1 \times 16^0}$ =

  $10 \times 16^1 + 1 \times 16^0$ =

  $160 + 1$ =

  161_{10}

- Convert 44 base 16 to decimal:

  $\mathbf{4 \times 16^1 + 4 \times 16^0}$ =

  $64 + 4$ =

  68_{10}

- Convert CAFE_{16} to base 10:

  $\mathbf{C \times 16^3 + A \times 16^2 + F \times 16^1 + E \times 16^0}$ =

  $12 \times 16^3 + 10 \times 16^2 + 15 \times 16^1 + 14 \times 16^0$ =

  $12 \times 4095 + 10 \times 256 + 15 \times 16 + 14 \times 1$ =

  51966_{10}

Hexadecimal

- Convert 507_{10} to base 16:
- Use division method with 16 instead of 2:

Hexadecimal

- Convert 507_{10} to base 16:
- Use division method with 16 instead of 2:

```
507/16 = 31, remainder 11 = B
31/16 = 1, remainder 15 = F
1/16 = 0, remainder 1
```

Hexadecimal

- Convert 507_{10} to base 16:
- Use division method with 16 instead of 2:

```
507/16 = 31, remainder 11 = B
31/16 = 1, remainder 15 = F
1/16 = 0, remainder 1
```

- Reading from bottom to top: $1FB_{16}$
- **Check your work:**

$1 \times 16^2 + F \times 16^1 + B \times 16^0 = 1 \times 16^2 + 15 \times 16^1 + 11 \times 16^0 = 256 + 240 + 11 = 507_{10}$

Utility of Hexadecimal

- Common to have groups of 32 bits
  - 32 bits is cumbersome to write
  - easy to make mistakes

- Use hexadecimal as a shorthand
  - 8 hex digits instead of 32 bits
  - Group bits from the right
  - Memorize mapping from binary to hex for values between 0 and F

Utility of Hexadecimal

Convert 0xF51A to binary

Convert 1001001010101011010100 from binary to hex

Utility of Hexadecimal

Convert 0xF51A to binary

F=1111_2, 5 = 0101_2, 1 =0001_2, A=1010_2

1111 0101 0001 1010_2

Convert 1001001010101011010100 from binary to hex

10 0100 1010 1010 1101 0100

0010=2 0100=4 1010=10 1010=10 1101=13 0100=4

0010=2 0100=4 1010=a 1010=a 1101=d 0100=4

0x24aad4

Character Data

Representing Characters

- **Standard encoding scheme called ASCII**
  - **American Standard Code for Information Interchange**
    - **7 bits per character** (2^7 = 128 possible characters)
  - Includes printable characters
  - Includes “control characters” that impact formatting (tab, newline), data transmission (mostly obsolete)
  - **Layout seems arbitrary, but actually contains some interesting patterns**

Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char
0	0000 0000	00	[NUL]	32	0010 0000	20	space	64	0100 0000	40	@	96	0110 0000	60	`
1	0000 0001	01	[SOH]	33	0010 0001	21	!	65	0100 0001	41	A	97	0110 0001	61	a
2	0000 0010	02	[STX]	34	0010 0010	22	"	66	0100 0010	42	B	98	0110 0010	62	b
3	0000 0011	03	[ETX]	35	0010 0011	23	#	67	0100 0011	43	C	99	0110 0011	63	c
4	0000 0100	04	[EOT]	36	0010 0100	24	$	68	0100 0100	44	D	100	0110 0100	64	d
5	0000 0101	05	[ENQ]	37	0010 0101	25	%	69	0100 0101	45	E	101	0110 0101	65	e
6	0000 0110	06	[ACK]	38	0010 0110	26	&	70	0100 0110	46	F	102	0110 0110	66	f
7	0000 0111	07	[BEL]	39	0010 0111	27	'	71	0100 0111	47	G	103	0110 0111	67	g
8	0000 1000	08	[BS]	40	0010 1000	28	(	72	0100 1000	48	H	104	0110 1000	68	h
9	0000 1001	09	[TAB]	41	0010 1001	29	)	73	0100 1001	49	I	105	0110 1001	69	i
10	0000 1010	0A	[LF]	42	0010 1010	2A	*	74	0100 1010	4A	J	106	0110 1010	6A	j
11	0000 1011	0B	[VT]	43	0010 1011	2B	+	75	0100 1011	4B	K	107	0110 1011	6B	k
12	0000 1100	0C	[FF]	44	0010 1100	2C	,	76	0100 1100	4C	L	108	0110 1100	6C	l
13	0000 1101	0D	[CR]	45	0010 1101	2D	-	77	0100 1101	4D	M	109	0110 1101	6D	m
14	0000 1110	0E	[SO]	46	0010 1110	2E	.	78	0100 1110	4E	N	110	0110 1110	6E	n
15	0000 1111	0F	[SI]	47	0010 1111	2F	/	79	0100 1111	4F	O	111	0110 1111	6F	o
16	0001 0000	10	[DLE]	48	0011 0000	30	0	80	0101 0000	50	P	112	0111 0000	70	p
17	0001 0001	11	[DC1]	49	0011 0001	31	1	81	0101 0001	51	Q	113	0111 0001	71	q
18	0001 0010	12	[DC2]	50	0011 0010	32	2	82	0101 0010	52	R	114	0111 0010	72	r
19	0001 0011	13	[DC3]	51	0011 0011	33	3	83	0101 0011	53	S	115	0111 0011	73	s
20	0001 0100	14	[DC4]	52	0011 0100	34	4	84	0101 0100	54	T	116	0111 0100	74	t
21	0001 0101	15	[NAK]	53	0011 0101	35	5	85	0101 0101	55	U	117	0111 0101	75	u
22	0001 0110	16	[SYN]	54	0011 0110	36	6	86	0101 0110	56	V	118	0111 0110	76	v
23	0001 0111	17	[ETB]	55	0011 0111	37	7	87	0101 0111	57	W	119	0111 0111	77	w
24	0001 1000	18	[CAN]	56	0011 1000	38	8	88	0101 1000	58	X	120	0111 1000	78	x
25	0001 1001	19	[EM]	57	0011 1001	39	9	89	0101 1001	59	Y	121	0111 1001	79	y
26	0001 1010	1A	[SUB]	58	0011 1010	3A	:	90	0101 1010	5A	Z	122	0111 1010	7A	z
27	0001 1011	1B	[ESC]	59	0011 1011	3B	;	91	0101 1011	5B	[	123	0111 1011	7B	{
28	0001 1100	1C	[FS]	60	0011 1100	3C	<	92	0101 1100	5C	\	124	0111 1100	7C	|
29	0001 1101	1D	[GS]	61	0011 1101	3D	=	93	0101 1101	5D	]	125	0111 1101	7D	}
30	0001 1110	1E	[RS]	62	0011 1110	3E	>	94	0101 1110	5E	^	126	0111 1110	7E	~
31	0001 1111	1F	[US]	63	0011 1111	3F	?	95	0101 1111	5F	_	127	0111 1111	7F	[DEL]

Representing More Characters

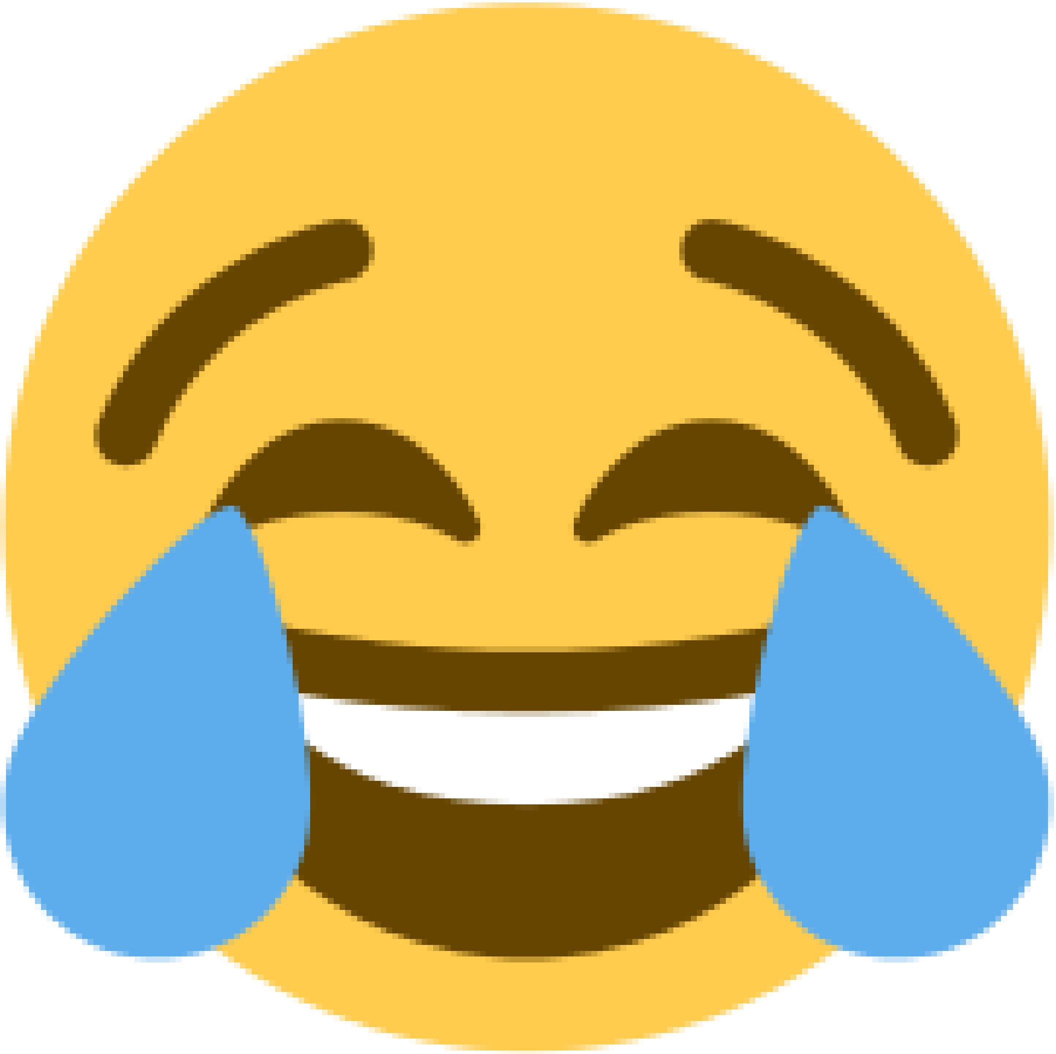
- Limitation of ASCII?
  - Only supports Latin character set
  - No support for accents, additional character sets
  - Solutions?

Representing More Characters

- **UTF-8**
  - Another encoding scheme for characters
    - **Variable length – 1, 2, 3 or 4 bytes per character**
  - **Compatible with ASCII**
  - Consider each byte
    - **Left most bit is 0? Usual ASCII Character**
    - Left most bits are 110? 2 byte character
    - Left most bits are 1110? 3 byte character
    - Left most bits are 11110? 4 byte character

    - \xF0\x9F\x98\x82 → tears of joy

    - (\x indicates hexadecimal bytes here)



UTF-8

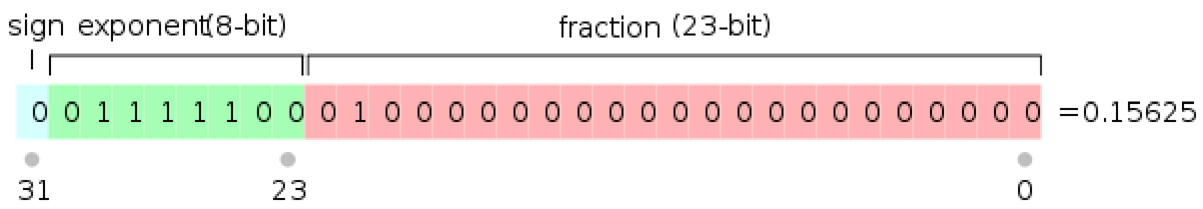
Number of bytes	Bits for code point	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	7	U+0000	U+007F	0xxxxxxx			
2	11	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	16	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	21	U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Decimal Point Numbers

Representing Real Numbers

- Standard Representation: IEEE 754 Floating Point
  - **Express the number in scientific notation**
  - **-0.0002589 becomes -2.589 * 10^{-4}**
- Need to store **sign**, **exponent**, and **mantissa** (the fraction)
- 32-bit floating point representation:
- **sign (1 bit), exponent (8 bits), mantissa (23 bits)**
- 64-bits:
- sign (1 bit), exponent (11 bits), mantissa (52 bits)

IEEE 754 – 32 Bit



Problems with Real Numbers

- How many real numbers are there? **Infinity**
- How many real numbers are there between 0 and 1? **Infinity**
- How many values can be represented by 32 or 64 bits?
- **2^{32} = 4.2 billion,**
- **2^{64} = 1.8 x 10^{19}**
- **Largest values: 2^{32} – 1 and 2^{64} – 1**
- What's the problem?

UNIVERSITY OF CALGARY

Problems with Real Numbers

- Problem: some real numbers exist that cannot be represented exactly in floating point
- (eg. 1/3 = 0.3333333…., sqrt(2) = 1.414213…).

- Thus floating point numbers only **approximate** real numbers (and maintaining accuracy is a very important concern!).

Image Data

Encoding Images

- Common Techniques
  - **Vector Images**
    - Vector images: “line work” Image is encoded as a collection of geometric primitives such as points, lines, curves.
  - **Raster Images**
    - Raster images: constructed from a grid of pixels (picture elements), where each picture is assigned a color

Representing Colors

- How do we represent a color as a sequence of bits?
- Can represent almost any color as a combination of some red, some green, and some blue. Typically use a scale from 0 (no light of that color) to 255 (full on for that color). Yields 256x256x256 = 16 million different possible colors.
  - (256 = 16*16   or two hex symbols)

- To represent an image: 3 color components for **each pixel** (becomes a lot of bytes very quickly!)

Videos

- Raster image storage formats like jpg heavily use ‘compression’ to reduce storage size
  - Basic ideas, reduce quantity of colours stored, and group idea of ‘where colours are’ to store less information
- Video compression works similar but since video is a sequence of frames where each frame is an image, they also make use of reducing data by grouping idea of ‘colours stay the same and where’ across multiple frames
  - Great example of compression failure → confetti
  - When confetti is in image, the colour of spot changes every frame and nearby spots are different each frame
  - This means more info is needed per frame, as a result at the same data rate, the image quality will go down (boxy artifacts will appear, or even decoding breaks down)
  - This is the same reasons sports struggle with compressed video

Onward to … decisions.

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~hudsonj/

UNIVERSITY OF CALGARY