Computer Architecture & Real-Time Operating System

4. Data Representation (2/2)

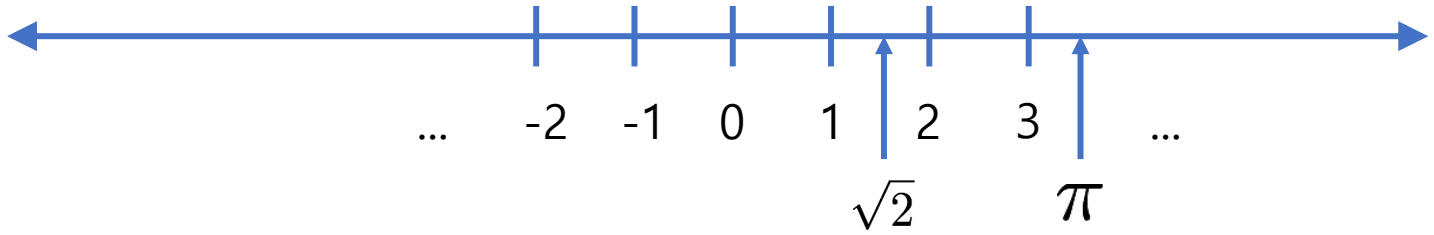
Prof. Jong-Chan Kim

Dept. Automobile and IT Convergence

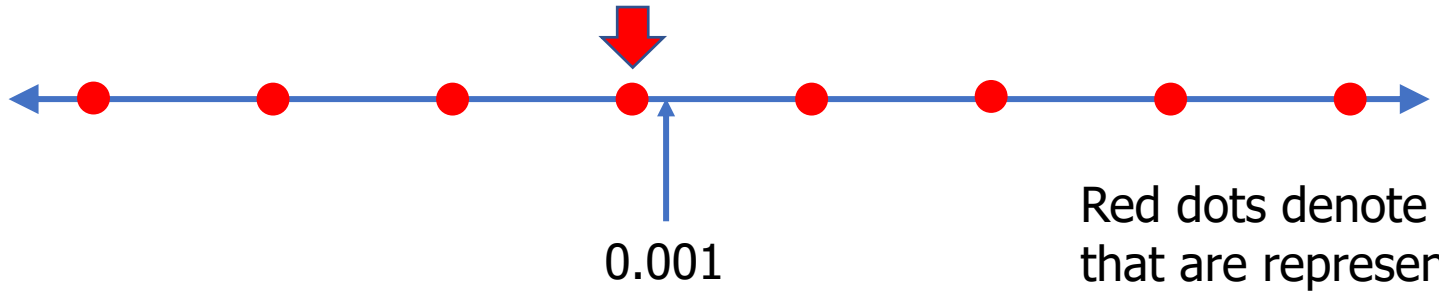
KMU 국민대학교 KOOKMIN UNIVERSITY

Real Numbers

- Quantities along a continuous numerical axis



- In any short range, there are an infinite number of real numbers
  - With *n* bits, you can only express 2^*n* accurate numbers
  - Thus, it is impossible to represent every possible real number with limited bits
  - Only a limited number of real numbers can be represented in computers
  - For example, when we put 0.001 in code, its approximation is stored instead



Red dots denote the selected real numbers that are represented without error

Two Ways to Express Real Numbers in Computer

- Fixed-Point Numbers
  - The number of digits in the fractional part is fixed (limited precision)
  - Fixed-point arithmetic can be implemented with integers
  - Pros: speed and accuracy (within the limited precision)
  - Cons: small range and limited precision

Fixed: 1234.56, 123.45, 12.34, 1.23, 0.12

- Floating-Point Numbers
  - The radix point can float arbitrarily on the numbers
  - Floating-point arithmetic is difficult to implement
  - Pros: large range and better precision
  - Cons: slow speed and inaccuracy

Floating: 1.23456, 12.3456, 123.456, 1234.56, 12345.6

For precision and accuracy, refer to Appendix. A

Inaccuracy of Floating-Point Numbers

Try precision.c

- Exact precision is not always possible

```
float sum = 0.0f;
for (int i = 0; i < 1000; i++) {
    sum += 0.001f;
}
printf("%f\n", sum);
```

```
0.999991
```

Try equality.c

- Equality check does not work

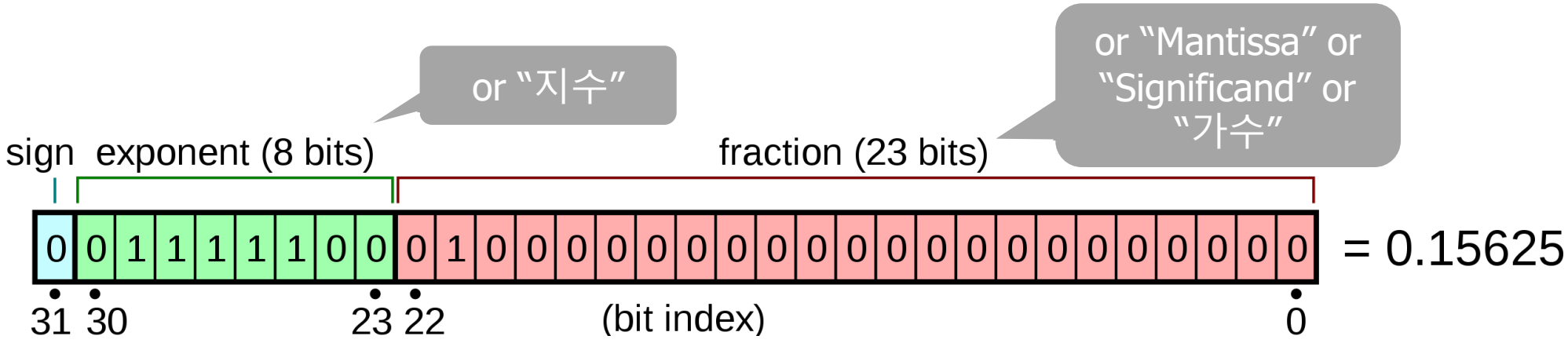
```
float a = 1.0f;
float b = a / 3.0f - 1.0f;
float c = (b + 1.0f) * 3.0f;

printf("%f %f\n", a, c);
if (a == c)
    printf("Equal\n");
else
    printf("Not Equal\n");
```

```
Not Equal
```

IEEE 754 Standard for Floating-Point Numbers

- Defines floating-point data types (float: 32 bits, double: 64 bits)
  - No unsigned data types
  - Number of bits are fixed (no changes across compilers)
- Memory layout and its interpretation for the 32-bit float data type



$$\text{Number} = (-1)^s \times (1.m)_2 \times 2^{(e-127)}$$

Mantissa is thought as placed below the binary point in binary notation

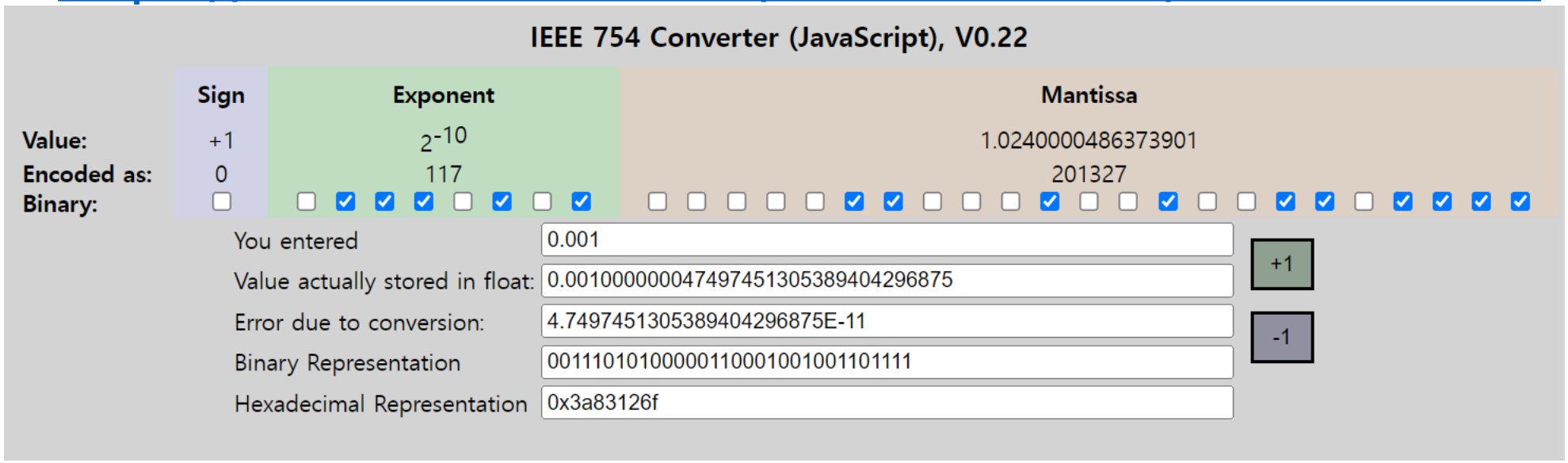
$(\ldots)_2$ means binary numbers

Figure source: https://en.wikipedia.org/wiki/IEEE_754-1985

IEEE 754 Converter

Try ieee754.c

- https://www.h-schmidt.net/FloatConverter/IEEE754.html



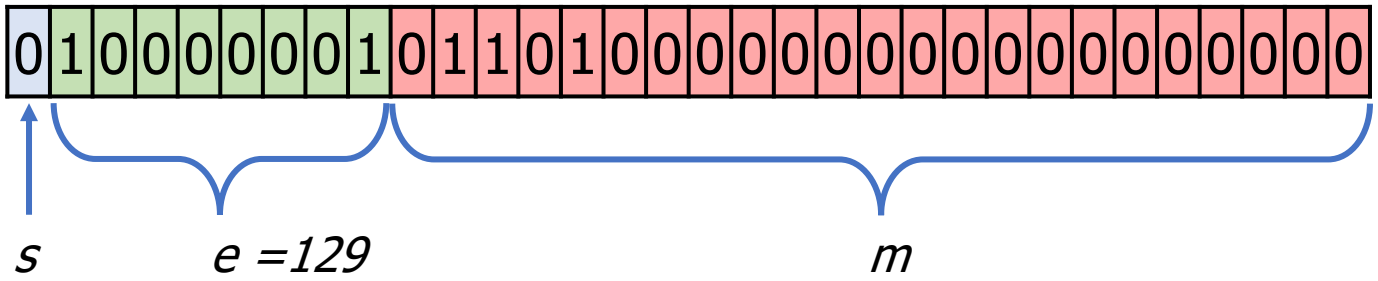

```
float a = 0.001f;
printf("%f\n", a);
printf("%35.33f\n", a);
printf("%02x %02x %02x %02x\n",
    ((unsigned char *)&a)[0],
    ((unsigned char *)&a)[1],
    ((unsigned char *)&a)[2],
    ((unsigned char *)&a)[3]);
```

Convert Decimal 5.625 to Float Type

- $5.625_{10} = 101.101_2$

$$\text{Number} = (-1)^s \times (1.m)_2 \times 2^{(e-127)}$$

$$101.101 = (-1)^0 \times (1.01101)_2 \times 2^2$$



Floating Point Data Types

- C data types

C data type	IEEE 754 name	Data size	Min (+)	Max (+)
float	Single precision	32 bits (4 bytes)	$3.4 * (10^{-38})$	$3.4 * (10^{+38})$
double	Double precision	64 bits (8 bytes)	$1.7 * (10^{-308})$	$1.7 * (10^{+308})$
long double	Quadruple precision	128 bits (16 bytes)	$3.4 * (10^{-4932})$	$1.1 * (10^{+4932})$

- Floating-point literals
  - Single precision (e.g., `0.001f`)
  - Double precision (e.g., `0.001`)
  - Quadruple precision (e.g., `0.001l`)

Default precision

How Computers Store Characters

- Computers can represent only numbers

http://computertutorinc.net/computer-maintenance-safety-tips/

- How can we teach characters to computers?
  - Assign each character a unique number (code)

'A' ⇄ 65

Character Code

ASCII (American Standard Code for Information Interchange) Code

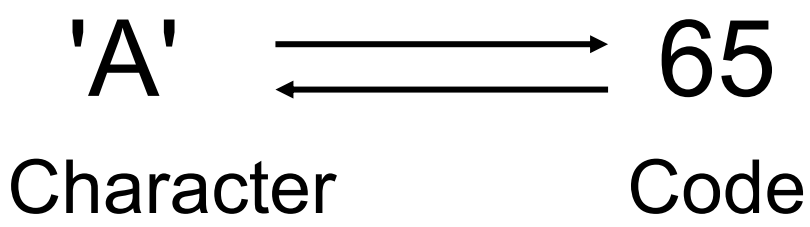
- Mapping between 128 characters and 7-bit unsigned integers (0 ~ 127)

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

ASCII Code

Try ascii.c

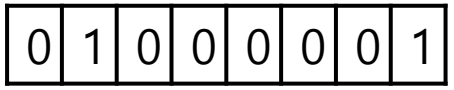



```
printf("%d = %c\n", 65, 65);
printf("%d = %c\n", 'A', 'A');
if (65 == 'A') {
    printf("A and \'65\' are the same thing\n");
}
```

```
65 = A
65 = A
'A' and 65 are the same thing
```

Character Data Type

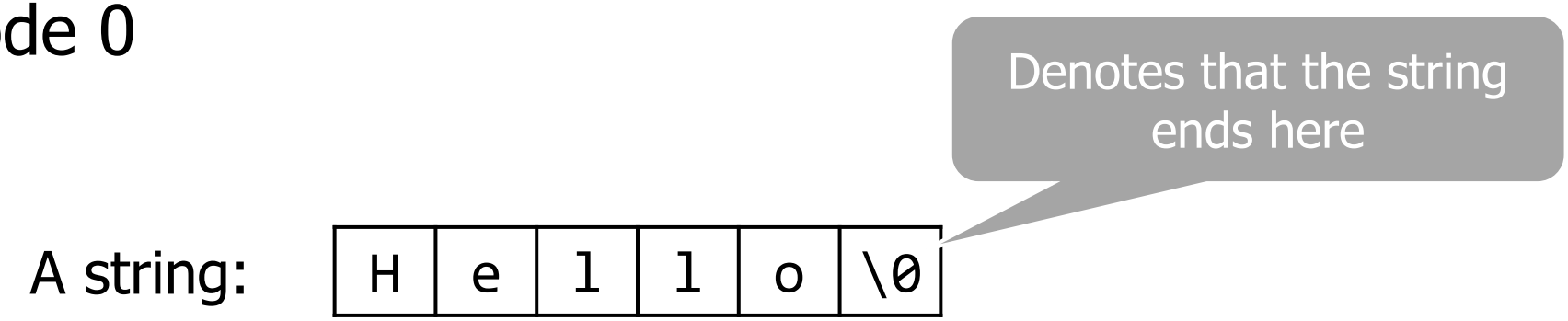
- The 8-bit `char` integer type is commonly used for storing ASCII codes
- The `char` type can represent -128 ~ 127, which is enough for ASCII




```
char c = 65;
printf("%d = %c\n", c, c);
```

String

- A sequence of characters with a terminating NULL character (`'\0'`) with its ASCII code 0




```
char s[] = {'H', 'e', 'l', 'l', 'o', '\0'};
printf("%c\n", *s);
printf("%s\n", s);
printf("%s\n", s + 1);
```

s is pointing to 'H'

%s prints next and next characters until the NULL character

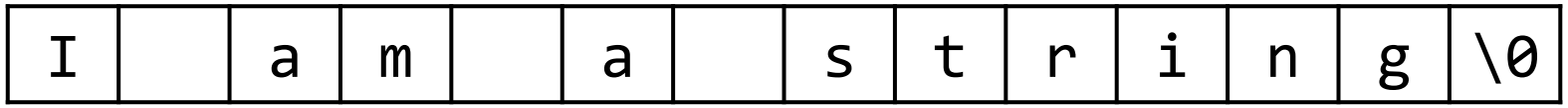
```
H
Hello
ello
```

String Literal

- A sequence of characters enclosed in double quotes

"I am a string"

- Compiler allocates (the number of characters + 1) bytes

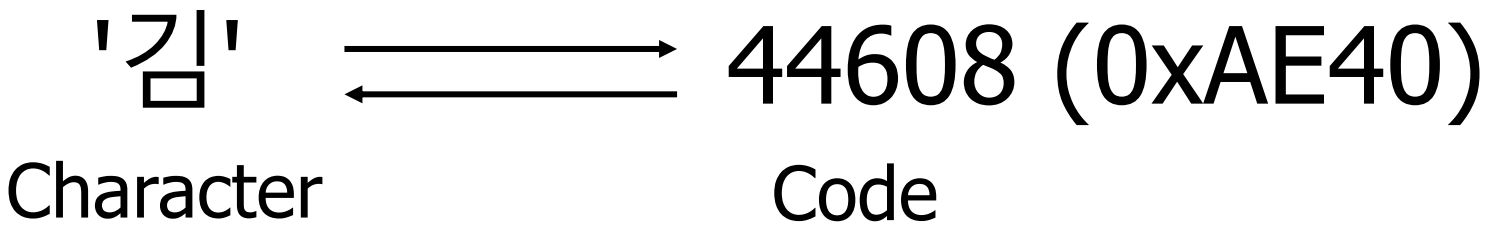


- Then **"I am a string"** means the starting address of it

```
char *s = "I am a string";
printf("%s\n", s);
```

Unicode

- ASCII was extended to UNICODE to store characters besides ASCII characters like Korean and Chinese characters

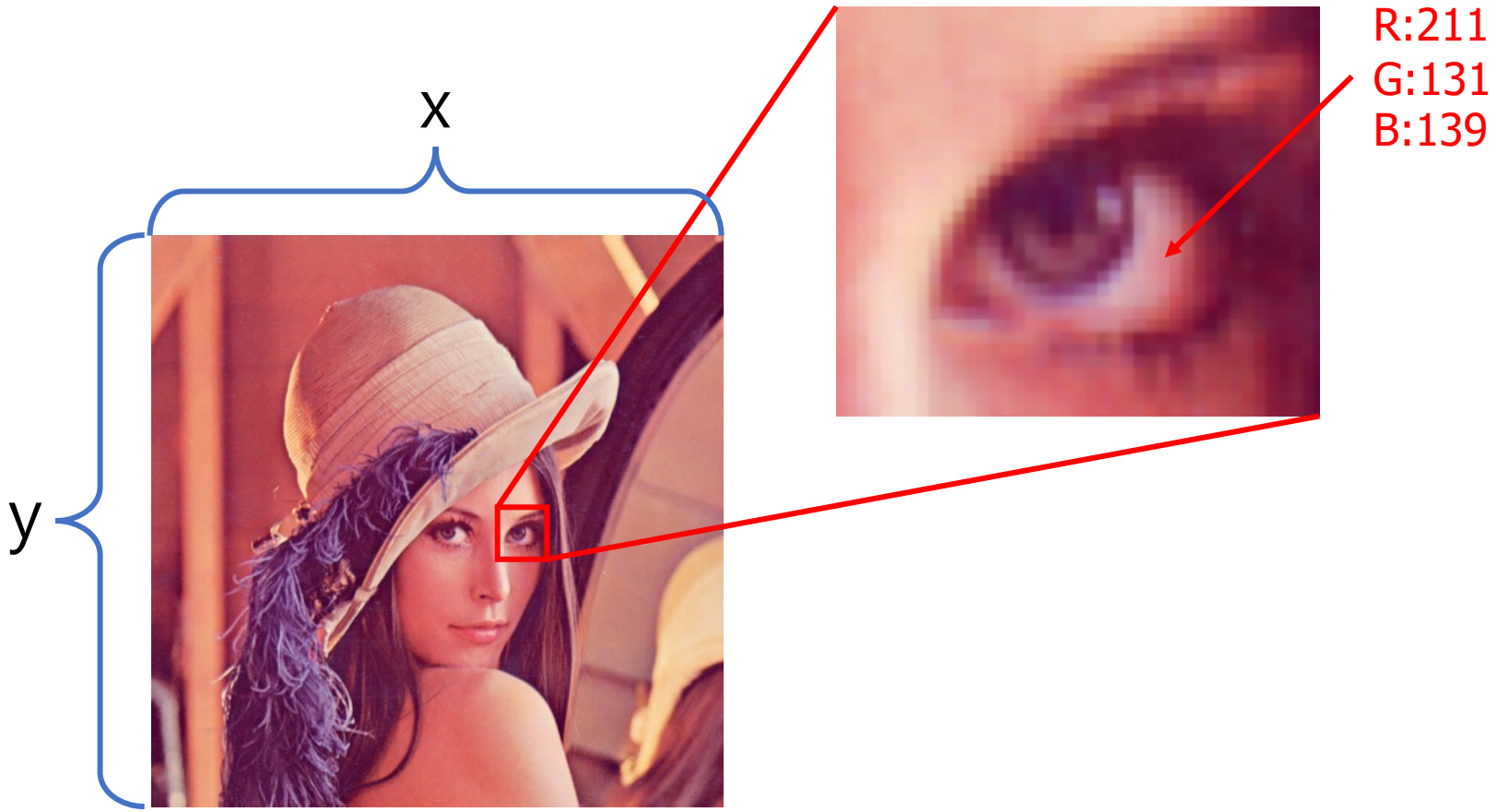


Visit http://unicode-table.com

- Since single byte is not enough for storing UNICODE characters, multibyte character types such as wchar_t are necessary

Multimedia Data

- A raster image data
  - Sequence of pixels represented by color codes with an image resolution (x, y)



Color Depth

- The number of bits allocated for each pixel
- Higher color depth enables more realistic images with larger file sizes

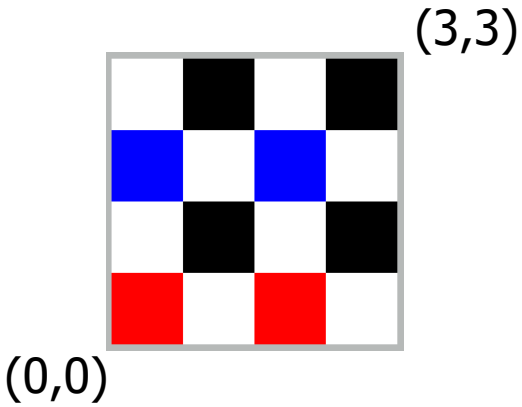
Allocate 1 bit for each pixel

Allocate 24 bits for each pixel

Bitmap File (BMP)

Try simple.bmp

- 24 bits for each pixel
  - 8 bits for B
  - 8 bits for G
  - 8 bits for R



4 rows 4 columns

(0,0) pixel

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   42 4D 7C 00 00 00 00 00 00 00 1A 00 00 00 0C 00   BM|.............
00000010   00 00 04 00 04 00 01 00 18 00 00 00 FF FF FF FF   ............ÿÿÿÿ
00000020   00 00 FF FF FF FF FF FF FF 00 00 00 FF FF FF 00   ..ÿÿÿÿÿÿÿ...ÿÿÿ.
00000030   00 00 FF 00 00 FF FF FF FF 00 00 FF FF FF FF FF   ..ÿ..ÿÿÿÿ..ÿÿÿÿÿ
00000040   FF 00 00 00 FF FF FF 00 00 00 00 00               ÿ|...ÿÿÿ.....
```

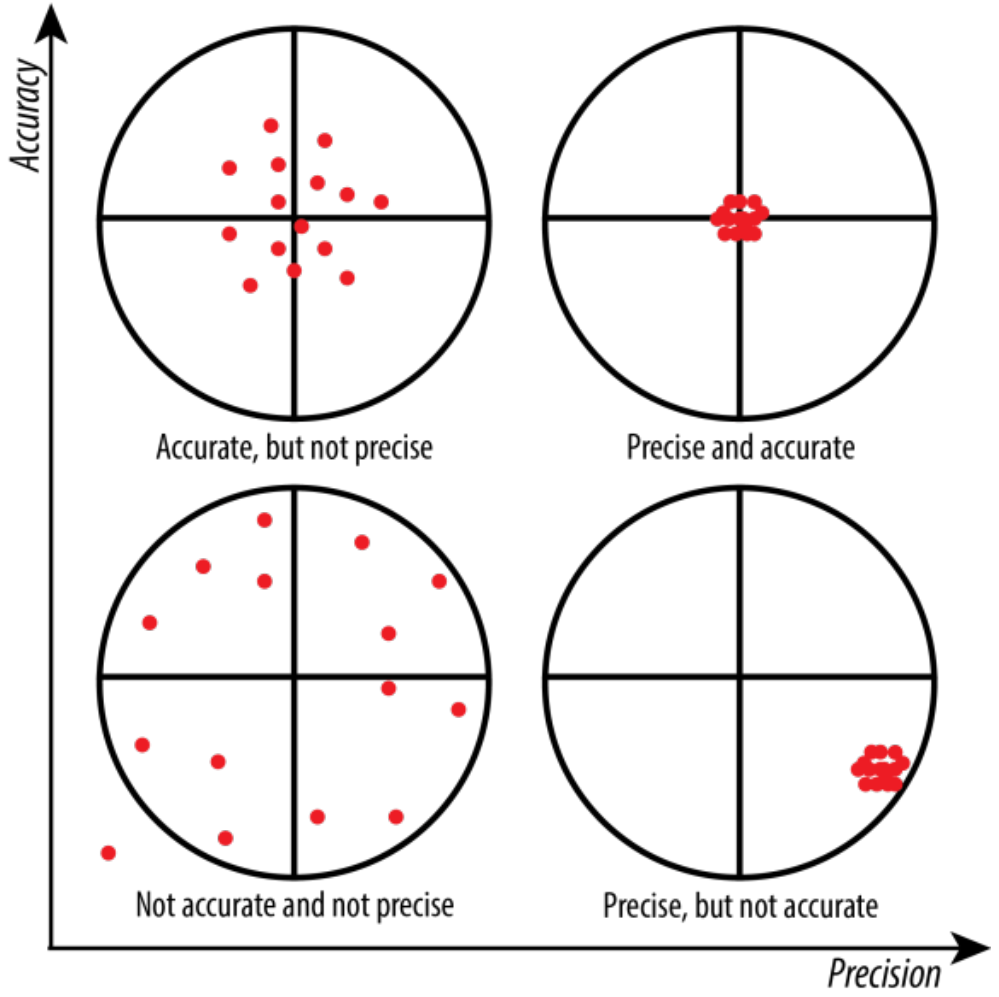
(3,3) pixel Padding

http://magazine.art21.org/2011/09/13/how-to-create-a-bitmap-image-file-by-hand-without-stencils/#.WqjK8ei5suW

Summary

- Floating-point data types
- Characters
- Strings
- Images

Appendix A. Precision and Accuracy



Source: https://wp.stolaf.edu/it/gis-precision-accuracy/