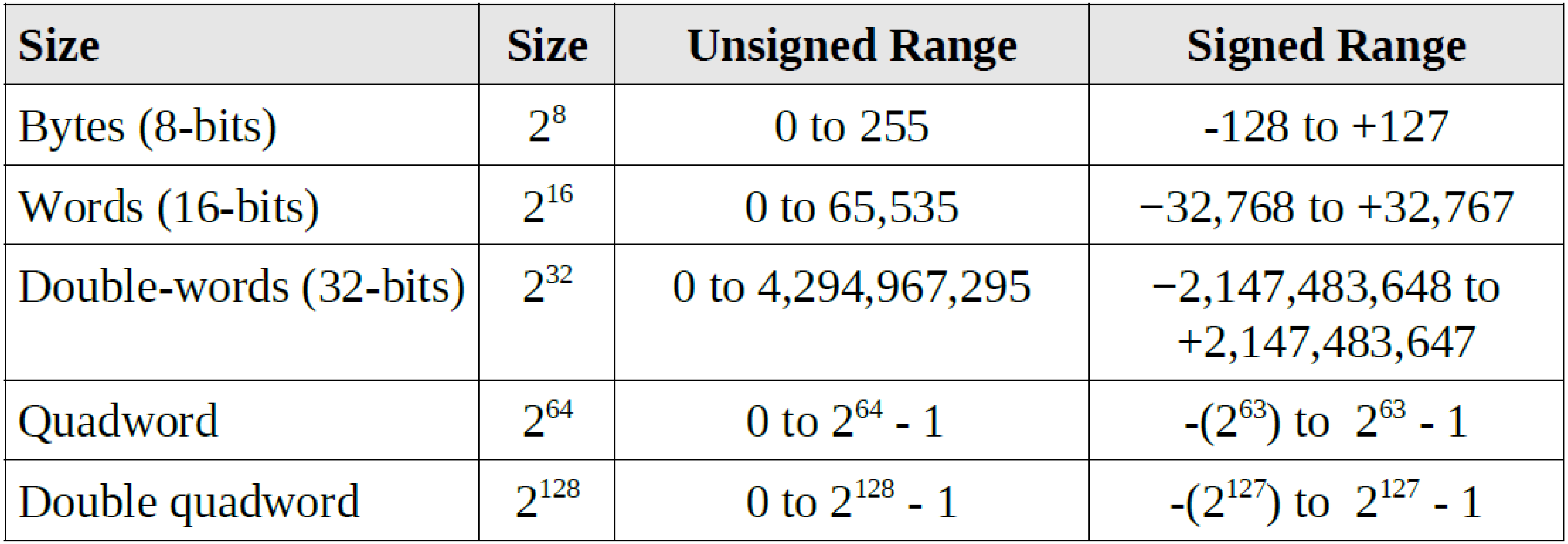
Data Representation

Data representation → how integers, floating-points and strings are stored in computer

Integer representation

- Integers are stored as 8, 16, 32, 64 and 128 bit binary
- Negative numbers are represented in **2’s complement** format
- Binary sizes and integer ranges are as shown below



Size	Size	Unsigned Range	Signed Range
Bytes (8-bits)	2^{8}	0 to 255	-128 to +127
Words (16-bits)	2^{16}	0 to 65,535	−32,768 to +32,767
Double-words (32-bits)	2^{32}	0 to 4,294,967,295	−2,147,483,648 to +2,147,483,647
Quadword	2^{64}	0 to 2^{64} - 1	-(2^{63}) to 2^{63} - 1
Double quadword	2^{128}	0 to 2^{128} - 1	-(2^{127}) to 2^{127} - 1

Data Representation

Binary Fraction

Fraction can be expressed in binary as

$$F = f_0\cdot 2^{-1} + f_1\cdot 2^{-2} + f_2\cdot 2^{-3} + f_3\cdot 2^{-4} + \ldots \quad ; f_0, f_1, f_2, \ldots = \{0, 1\}$$

$$= 0.5f_0 + 0.25f_1 + 0.125f_2 + 0.0625f_3 + \ldots$$

Ex $0.375 = 0\text{x}0.5 + 1\text{x}0.25 + 1\text{x}0.125 + 0\text{x}0.0625$

$= \mathbf{0.0110_2}$

For 8-bit fraction, the resolution is $\mathbf{2^{-8}}$ = 0.00390625

Ex $4.625 = 100.101_2$

$= \mathbf{1.00101_2 \text{ x } 2^2}$ ←Normalized form (1 digit non-leading zero)

00101 is called fraction part of floating-point

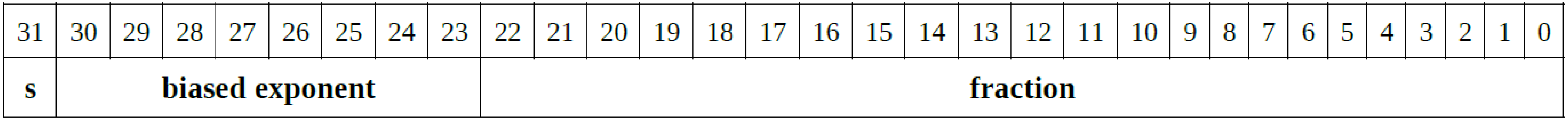
2 is called exponent part of floating-point

. is called binary point

Data Representation

Floating-point representation

- IEEE 754 (32-bit) Standard
- S = sign (0 → positive, 1 → negative)
- e = exponent
- E = biased exponent
- F = fraction or mantissa



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
s	**biased exponent**								**fraction**																						

$$N = (-1)^s \text{ x } 1.F \text{ x } 2^e \quad \text{where } e = E\text{-}127$$

Data Representation

Example 1: **-7.75**

- determine sign -7.75 => 1 (since negative)
- convert to binary -7.75 = -0111.11_2
- normalized scientific notation = 1.1111×2^2
- compute biased exponent $2_{10} + 127_{10} = 129_{10}$
  - and convert to binary $= 10000001_2$
- write components in binary:

  sign exponent mantissa
  1 10000001 11110000000000000000000
- convert to hex (split into groups of 4)

  11000000111110000000000000000000
  1100 0000 1111 1000 0000 0000 0000 0000
  C 0 F 8 0 0 0 0
- final result: **$\mathbf{C0F8\ 0000_{16}}$**

Data Representation

Example 2: **-0.125**

- determine sign -0.125 => 1 (since negative)
- convert to binary -0.125 = -0.001_2
- normalized scientific notation = 1.0×2^{-3}
- compute biased exponent $-3_{10} + 127_{10} = 124_{10}$
    - and convert to binary $= 01111100_2$
- write components in binary:

```
sign  exponent   mantissa
  1   01111100  00000000000000000000000
```

- convert to hex (split into groups of 4)

```
10111110000000000000000000000000
1011 1110 0000 0000 0000 0000 0000 0000
  B    E    0    0    0    0    0    0
```

- final result: **$\mathbf{BE00\ 0000_{16}}$**

Data Representation

Example 3: **41440000_{16}**

- convert to binary

  $0100\ 0001\ 0100\ 0100\ 0000\ 0000\ 0000\ 0000_2$
- split into components

  $0\ 10000010\ 10001000000000000000000_2$
- determine exponent $10000010_2 = 130_{10}$
  - and remove bias $130_{10} - 127_{10} = 3_{10}$
- determine sign 0 => positive
- write result $+1.10001 \times 2^3 = +1100.01 =$ **+12.25**

Data Representation

Floating-point representation

- IEEE 754 (64-bit) Standard
- S = sign (0 → positive, 1 → negative)
- e = exponent
- E = biased exponent
- F = fraction or mantissa

63	62		52	51		0
s	**biased exponent**			**fraction**		

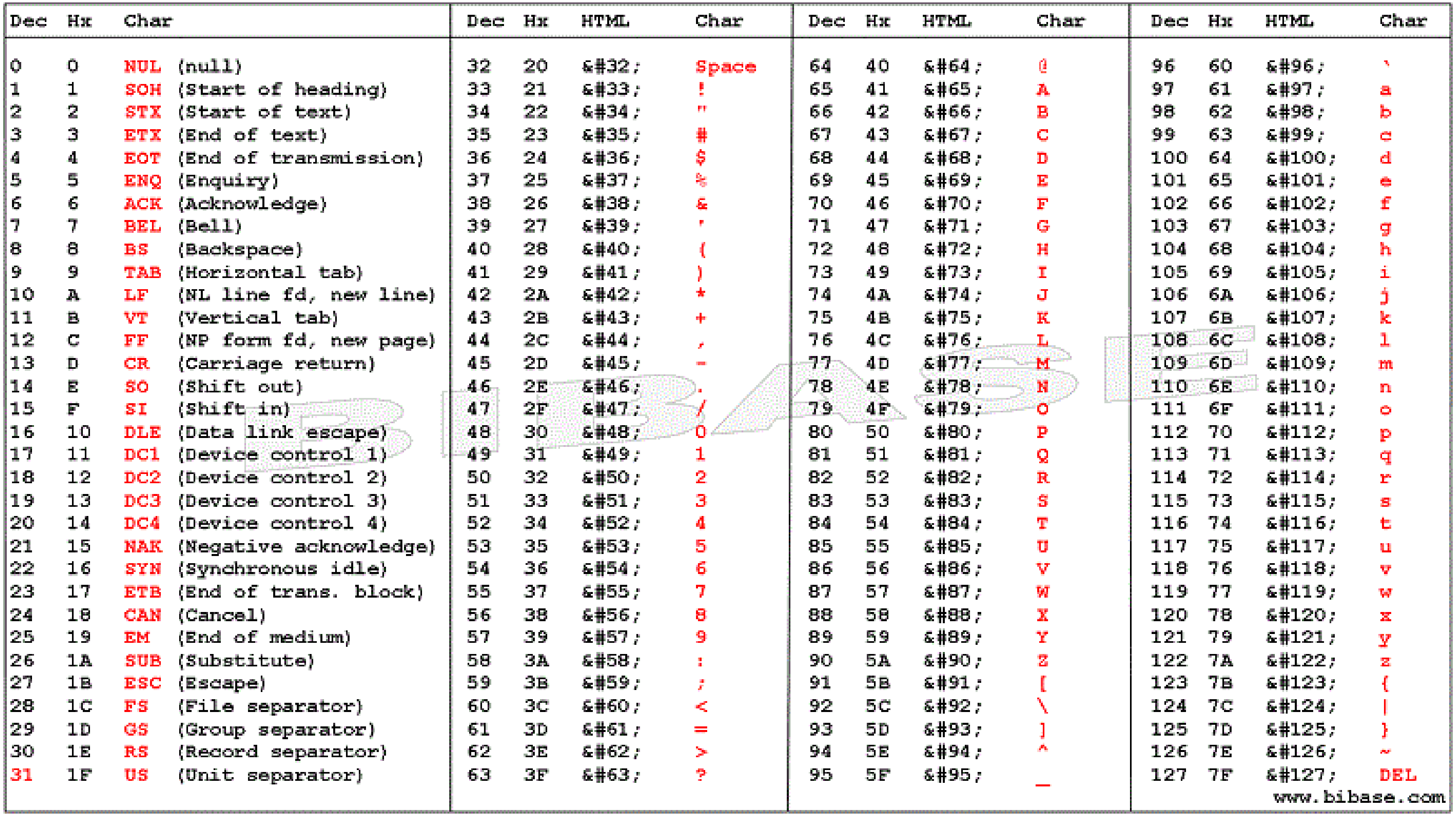
$$N = (-1)^{s} \times 1.F \times 2^{e} \quad \text{where } e = E\text{-}1023$$

When floating-point cannot fit standard format (either 32 or 64 bit), it is called **Not a Number (NaN)**

Data Representation

Character representation

- Character means a symbolic or non-numeric data
- String means a sequence of character
- Characters are represented as numbers defined by ASCII standard
- ASCII ==> American Standard Code for Information Interchange
- ASCII defines 128 characters (letters, numbers, symbols and control characters)



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

www.bibase.com

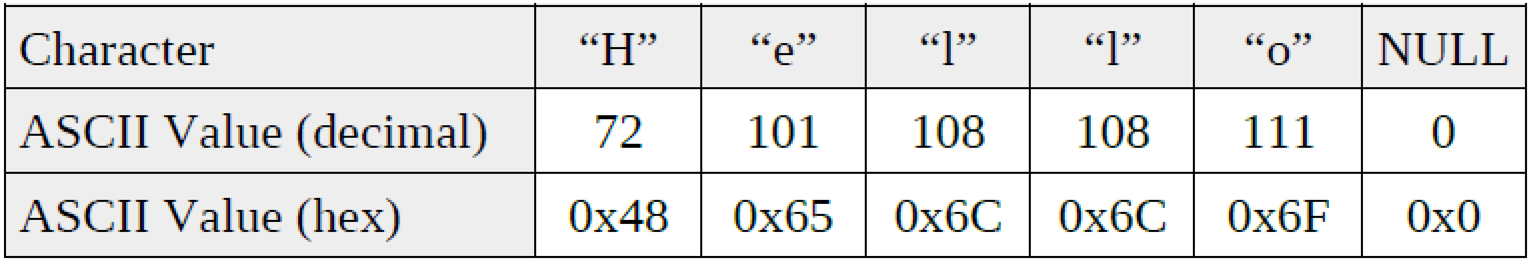
Data Representation

Character representation

- It is important that a number can be stored as characters or numeric data
- For example, 2 can be stored as ASCII code 0x32 or a number 2

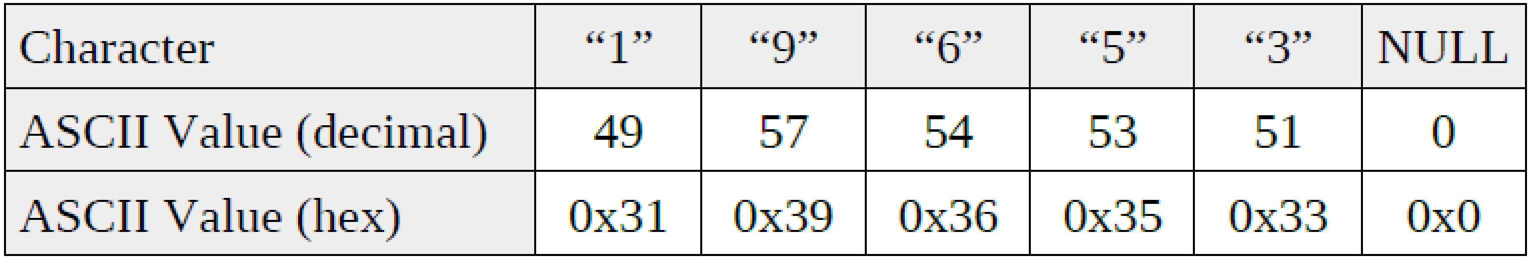
String representation

- String is a sequence of characters terminated with NULL (0x00)
- A string “HELLO” is stored as



Character	“H”	“e”	“l”	“l”	“o”	NULL
ASCII Value (decimal)	72	101	108	108	111	0
ASCII Value (hex)	0x48	0x65	0x6C	0x6C	0x6F	0x0

- A string “19653” is stored as

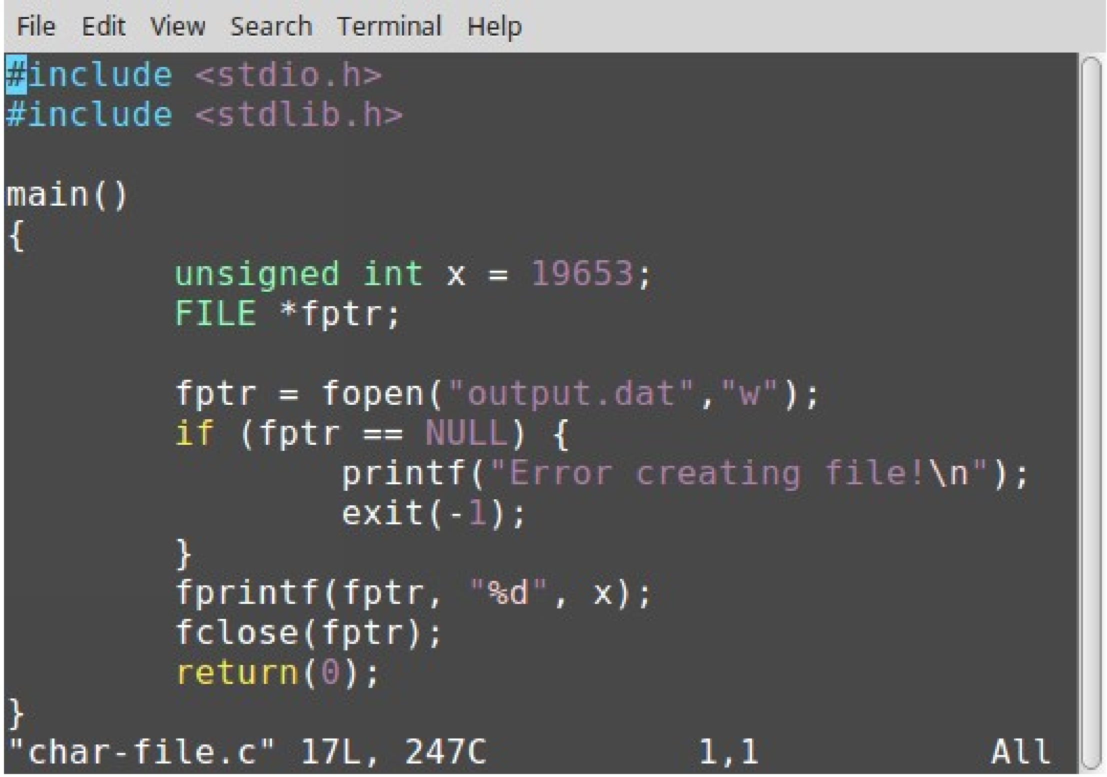


Character	“1”	“9”	“6”	“5”	“3”	NULL
ASCII Value (decimal)	49	57	54	53	51	0
ASCII Value (hex)	0x31	0x39	0x36	0x35	0x33	0x0

Data Representation

Practice session

- Write C programs to save an integer 19356 in a file as string (text file)



```
#include <stdio.h>
#include <stdlib.h>

main()
{
        unsigned int x = 19653;
        FILE *fptr;

        fptr = fopen("output.dat","w");
        if (fptr == NULL) {
                printf("Error creating file!\n");
                exit(-1);
        }
        fprintf(fptr, "%d", x);
        fclose(fptr);
        return(0);
}
"char-file.c" 17L, 247C          1,1           All
```

To compile source code, use command

gcc -o <executable_name> <sourcefile.c>

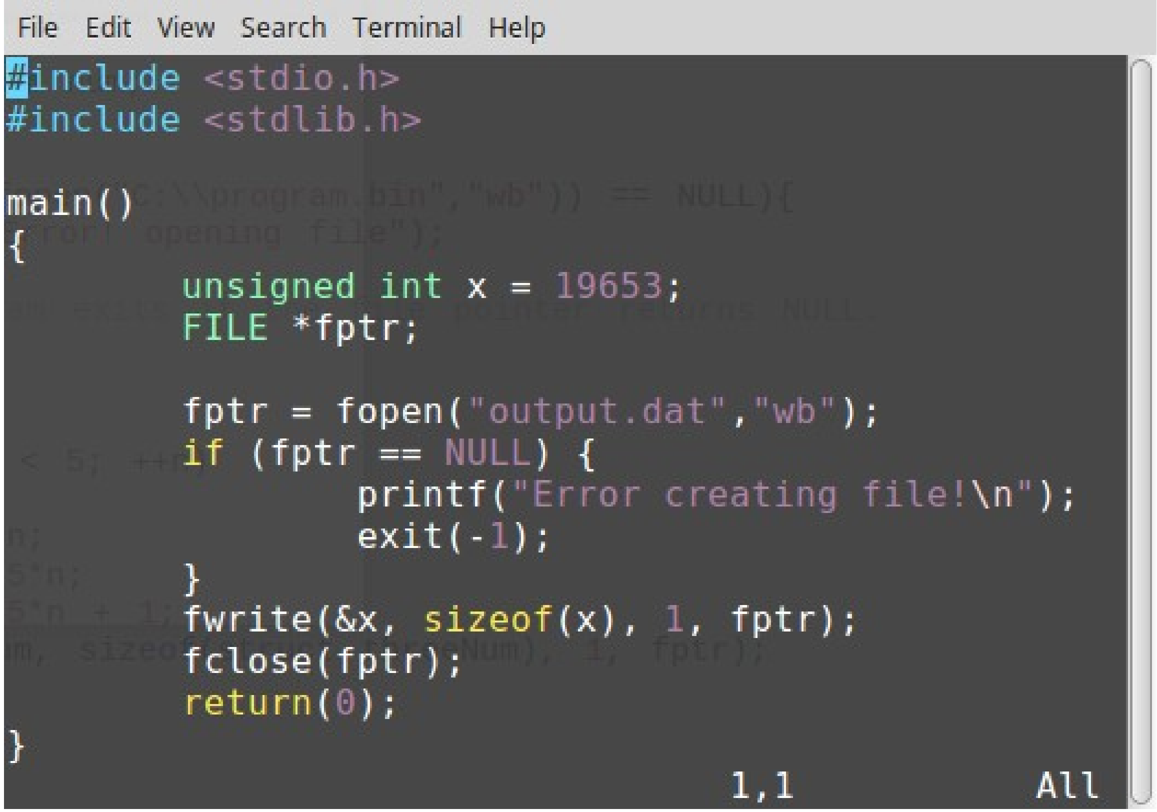
To view content of output.dat, use command

cat output.dat

Data Representation

Practice session

- Write C programs to save an integer 19356 in a binary file



```
#include <stdio.h>
#include <stdlib.h>

main()
{
        unsigned int x = 19653;
        FILE *fptr;

        fptr = fopen("output.dat","wb");
        if (fptr == NULL) {
                printf("Error creating file!\n");
                exit(-1);
        }
        fwrite(&x, sizeof(x), 1, fptr);
        fclose(fptr);
        return(0);
}
                                  1,1           All
```

To compile source code, use command

gcc -o <executable_name> <sourcefile.c>

To view content of output.dat, use command

hexdump output.dat