Agenda

Data representations in binary (Introduction)

Conversions between hexadecimal, decimal, octal, and binary

Binary data type representations
- ASCII characters
- unsigned integers
- signed integers
- floating-point

Byte Layouts
- padding
- endianness

Writing/Reading binary files, Hexedit

Binary and Data Representation

Data is stored as **binary** signals

e.g. they can either be **on** or **off**

Each signal corresponds to a single **bit**

All data can be represented with bits

more complicated data -> needs more bits

Binary and data representation

Smallest unit of addressable memory is a **byte**

Memory address 0: 01010101
Memory address 1: 10101010
Memory address 2: 00001111

A byte is **8 bits**

A **word** is the default size of memory that the hardware moves around

either 32 bits or 64 bits

Why define variables? Why have types?

Variable types in C

- Different Types have different number of bytes:

```
1 byte:  char, unsigned char (no negative values)
2 bytes: short, unsigned short
4 bytes: int, unsigned int, float
8 bytes: long long, unsigned long long, double
4 or 8 bytes: long, unsigned long
```

```
NOTE: On a 64 bit machine, pointers are ___ bytes
```

Example: Memory can be interpreted in different ways depending on the context

Consider the data

0b01001100011011111101001100 (0x4C6F4C)

The above bits can mean any of the following

LOL

5,009,228

Example: ASCII

American Standard Code for Information Interchange

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

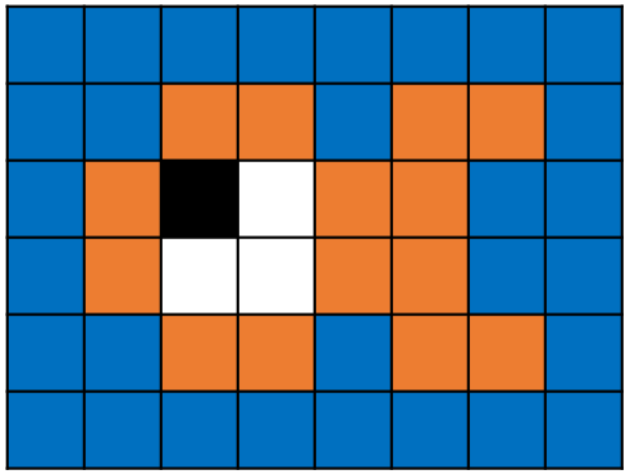
What is decimal number 1 in binary?

What is the ‘1’ in binary?

Example: Simple image

Binary Interpretation:

00	White	01	Orange
10	Blue	11	Black



(a)

10	10	10	10	10	10	10	10
10	10	01	01	10	01	01	10
10	01	11	00	01	01	10	10
10	01	00	00	01	01	10	10
10	10	01	01	10	01	01	10
10	10	10	10	10	10	10	10

(b)

10101010	10101010
10100101	10010110
10011100	01011010
10010000	01011010
10100101	10010110
10101010	10101010

(c)

Figure 2. The (a) image representation, (b) two-bit cell representation, and (c) byte representation of a simple fish image. *Dive Into Systems*

Number bases and unsigned integers

Recall: Decimal numbers

$$5163 = 5 * 10^3 + 1 * 10^2 + 6 * 10^1 + 3 * 10^0$$

What is the general formula?

Notation

0b (zero-b) denotes a binary number, e.g 0b 1001

0x (zero-x) denotes a hexadecimal number, e.g. 0xE3

0 (zero) denotes an octal number, e.g. 0644

d_0 denotes the lowest order bit

d_{N-1} denote the highest order bit

Other popular notation: $74_{10} = 7 * 10^1 + 4 * 10^0$

Hexadecimal

Base 16

Compact way to represent binary numbers

Octal (base 8) is sometimes used

Dec	Bin	Hex	Dec	Bin	Hex	Dec	Bin	Hex	Dec	Bin	Hex
0			4			8			12		
1			5			9			13		
2			6			10			14		
3			7			11			15		

Exercise: Hexadecimal

Convert this binary number to hexadecimal

0b01001100110011101000100

Convert this hexadecimal number to binary

0x80A5E2

Hexadecimal to decimal

Convert 0x1203 to decimal

Octal (Base 8)

Convert this binary number to octal

0b010011001100111101000100

Convert this octal number to binary: 04513

Convert 01203 to decimal

Decimal to hexadecimal

Idea: Compute digits from highest order to lowest order

Example: divide by 16^4, then 16^3, then by 16^2, etc

Each step i from N-1 to 0 do

i^{th} digit = floor(input/16^i)

input = input % 16^i

Exercise: Convert 542_{10} to hexadecimal

Digit Place	Input	Divisor	Digit (/)	Remainder (%)
d4		16^4		
d3		16^3		
d2		16^2		
D1		16^1		
d0		16^0		

Decimal to binary

Can use the same approach, but there is an easier way: **repeated division**

Idea: Repeatedly divide by 2 and check the parity

Each step i from 0 to N-1 do

i^{th} digit = input % 2

input = floor(input / 2)

Exercise: Convert 542_{10} to binary

Unsigned integers

Numbers ranging from 0 to a positive max value

What is the largest number that can be stored in 4 bits?

What is the largest number that can be stored in N bits?

Overflow

When we try to store a value too large to fit into a data type, we get **overflow**

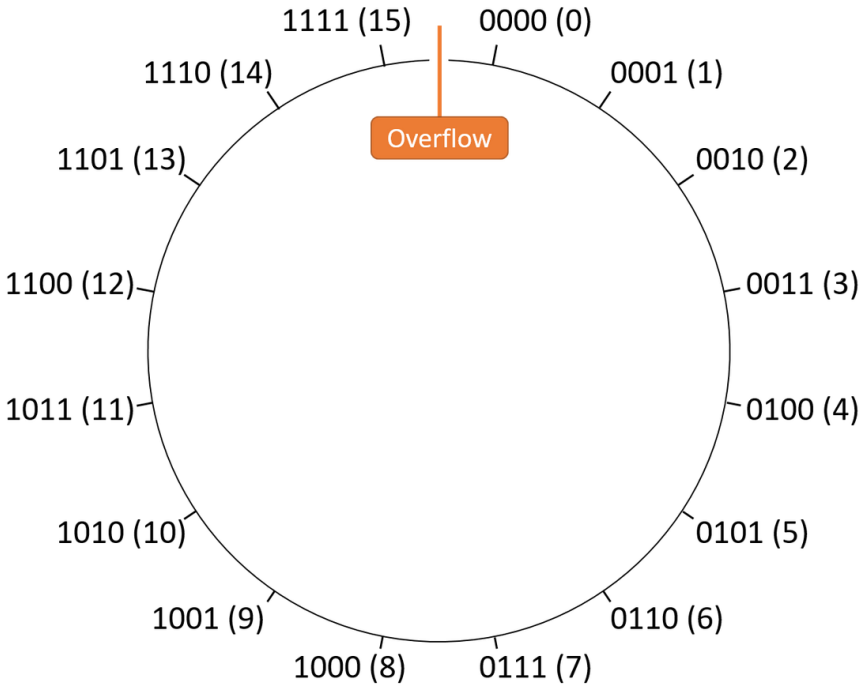


Figure 3. An arrangement of four-bit unsigned values into a modular space. All arithmetic is modular with respect to 2^4 (16).

What is the output of this program?

```
#include <stdio.h>

int main() {
   unsigned int a = 0;
   for (a = 5; a >= 0; a--) {
      printf("Message!\n");
   }
   return 0;
}
```

Signed integers

Modern systems use a method called **two's compliment**

highest order bit encodes the sign (0 -> positive; 1 -> negative)

advantage: allows pos/neg numbers to be treated the same in hardware

Two’s compliment

Suppose we have N bits to represent a signed integer. The formula is

$$-(d_{N-1} \times 2^{N-1}) + (d_{N-2} \times 2^{N-2}) + \ldots + (d_2 \times 2^2) + (d_1 \times 2^1) + (d_0 \times 2^0)$$

^ note the leading negative sign for just the first term!

Example: What is 0b1001 interpreted as a *signed* integer? As an *unsigned* integer?

Exercise

Convert the following *signed* integer to decimal: 0b 0110

Convert the following *signed* integer to decimal: 0b 1111

Two’s-compliment: 4 bits

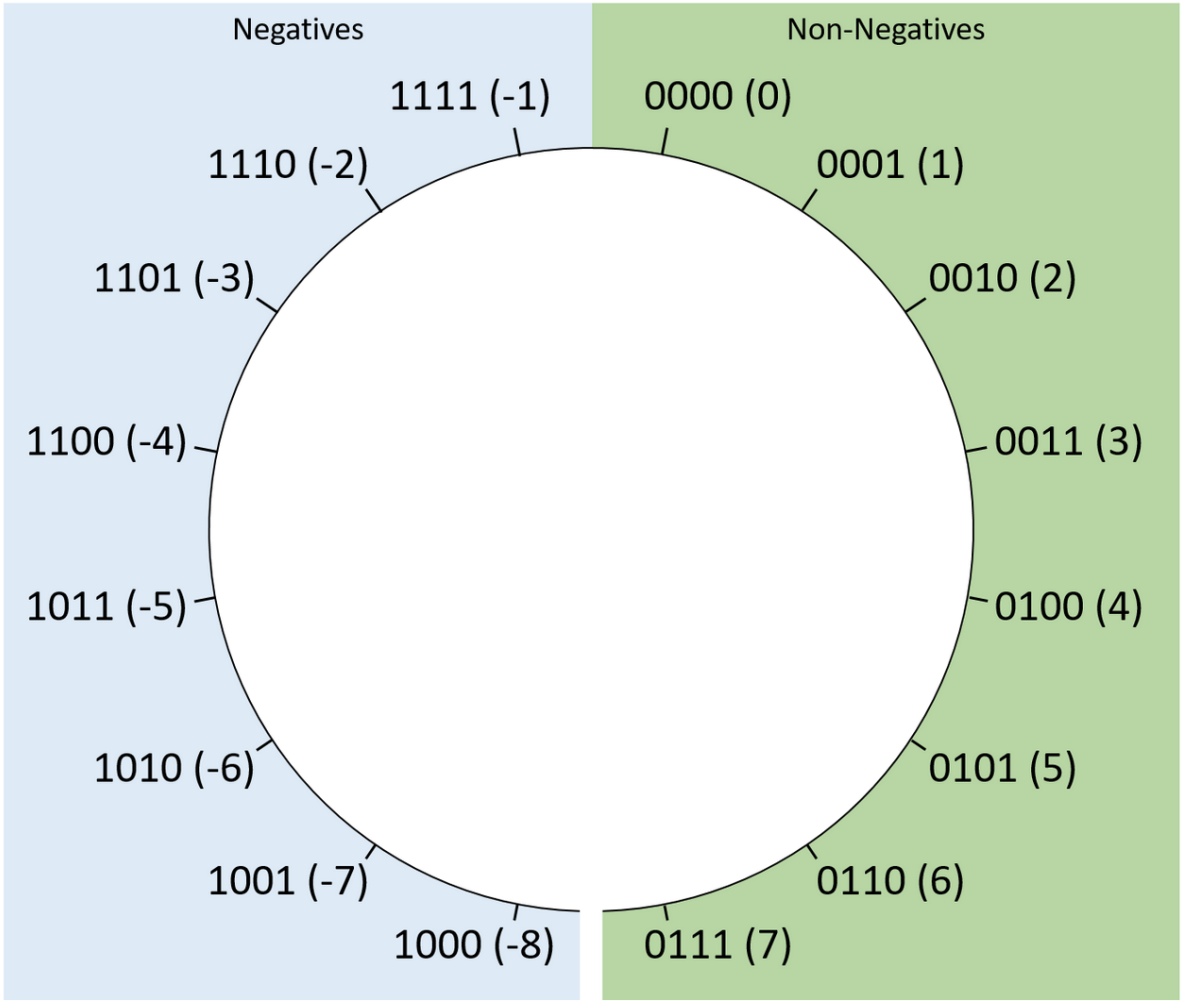


Figure 36. A logical layout of two’s complement values for bit sequences of length four.

Question

If we use 4 bits, what is the range of unsigned integers we can represent?

If we use 4 bits, what is the range of signed integers we can represent?

If we use N bits, what is the range of signed integers we can represent?

Negation

To negate a two's compliment signed integer

flip the bits

add one

Example: Compute -5 from 5 in binary

Signed extension

What happens when you perform an arithmetic operation on numbers with different sizes? **signed extension**

For *unsigned* values, prepend 0

For *signed* values, prepend the leftmost bit

Example: Signed extension

Add 5 and -5 as 8 bit numbers

Signed extension: example

0b 1011	
0b 1 1011	
0b 11 1011	

Floating point

Goals of representation

- Need to encode the sign
- Need to encode both very large and very small numbers

Idea: We let the placement of the decimal point change

Floating point

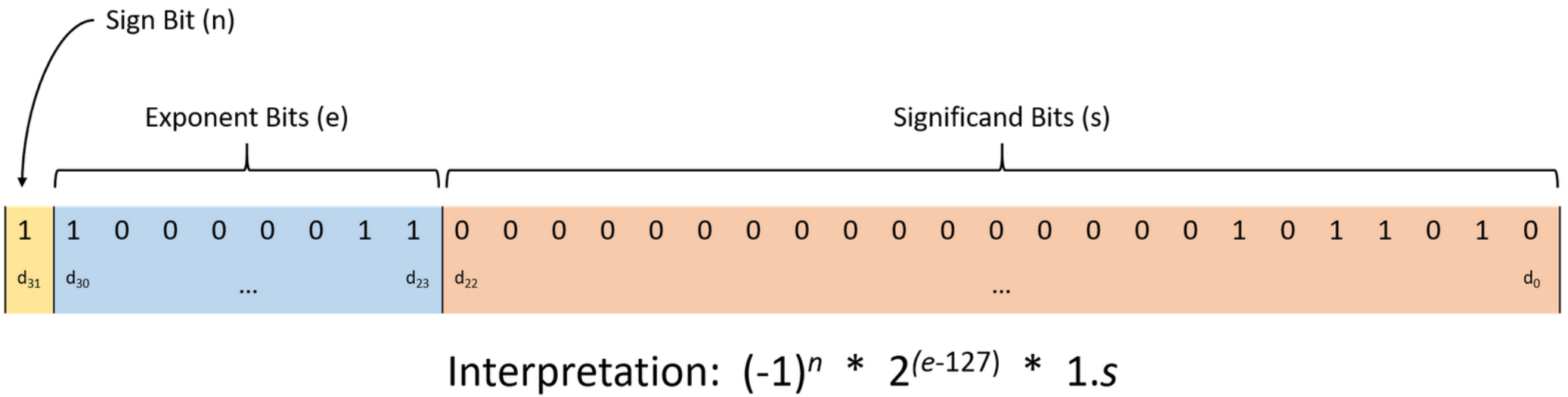


Figure 45. The 32-bit IEEE 754 floating-point standard

NOTE: e-127 called the **bias,** range is [-126,127]

Floating point example

What is the value of this binary floating point number in decimal?

1100 0001 1011 0100 0000 0000 0000 0000

Byte layouts: data type padding

Rule: data is always on a 1/4/8-byte boundary (depending on type size)

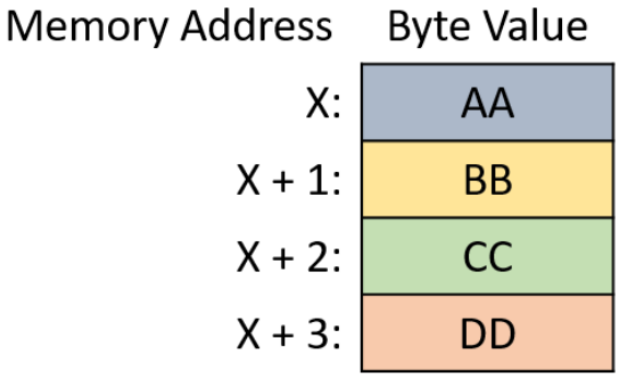
```
struct mystery {
  char c;
  float q[4];
};
```

```
sizeof(mystery} = 20 bytes
mystery.c    - 0x7ffcddd3f0e0
mystery.q[0] - 0x7ffcddd3f0e4
mystery.q[1] - 0x7ffcddd3f0e8
mystery.q[2] - 0x7ffcddd3f0ec
mystery.q[3] - 0x7ffcddd3f0f0
```

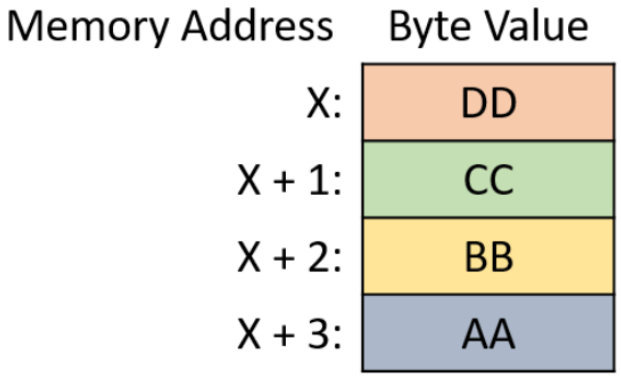
Byte order

The **smallest** addressable unit of memory is a **byte**

If a type is larger than 1 byte, how should we store it?



(a) Big-Endian



(b) Little-Endian

Byte order: demo

```
#include <stdio.h>

int main() {
  int value = 0xAABBCCDD;
  char* p = (char *) &value;

  int i;
  for (i = 0; i < sizeof(value); i++)  {
    printf("Address %p: %02hhX\n", p, *p);
    p += 1;
  }
}
```

Writing Binary Files

```
#include <stdio.h>
struct Data {
  int size;
  float p[3];
  char buff[16];
};

int main()
{
  struct Data data = {10, 1.0, -2.0, 0.5, "carrot"};
  FILE* fp = fopen("data.bin", "wb");
  fwrite(&data, sizeof(struct Data), 1, fp);
  fclose(fp);
}
```

```
00000000   0A 00 00 00  00 00 80 3F  00 00 00 C0  00 00 00 3F  .......?.......?
00000010   63 61 72 72  6F 74 00 00  00 00 00 00  00 00 00 00  carrot..........
00000020
```

Using hexedit

```
00000000   0A 00 00 00  00 00 80 3F  00 00 00 C0  00 00 00 3F  .......?.......?
00000010   63 61 72 72  6F 74 00 00  00 00 00 00  00 00 00 00  carrot..........
00000020
```

Left Column: line numbers in hex
Middle Column: File data in hex
Right Column: File data in ASCII

/ <keyword>: search

Tab: Switch between binary and ASCII mode

Ctrl-C: exit

Reading Binary Files

```
struct Data data;
 FILE* fp = fopen("data.bin", "rb");
 fread(&data, sizeof(struct Data), 1, fp);
 fclose(fp);
```

```
00000000   0A 00 00 00  00 00 80 3F  00 00 00 C0  00 00 00 3F  .......?.......?
00000010   63 61 72 72  6F 74 00 00  00 00 00 00  00 00 00 00  carrot..........
00000020
```

Helpful binary file commands

FILE* fp = fopen(filename, “rb”);

// Write binary data

size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)

// Read binary data

size_t fread(const void *ptr, size_t size, size_t nmemb, FILE *stream)

Exercise: Writing binary files

```
struct point {
  int x;
  int y;
};

int main() {
  struct point p[3];
  for (int i = 0; i < 3; i++) {
    p[i].x = i;
    p[i].y = 2*i;
  }

  FILE* fp = fopen("points.bin", "wb");
  fwrite(p, sizeof(struct point), 3, fp);
  fclose(fp);
}
```

Draw the stack diagram

What are the contents of the written file?

Reading binary files

```
struct point {
  int x;
  int y;
};

int main() {
  struct point p[3];

  FILE* fp = fopen("points.bin", "rb");
  fread(p, sizeof(struct point), 3, fp);
  fclose(fp);

  for (int i = 0; i < 3; i++) {
    printf("p[%d] = (%d,%d)\n", i, p[i].x, p[i].y);
  }
}
```

Discuss: How are binary files different from text files?

```
struct point {
  int x;
  int y;
};

int main() {
  struct point p[3];
  for (int i = 0; i < 3; i++) {
    p[i].x = i;
    p[i].y = 2*i;
  }

  FILE* fp = fopen("points.bin", "w");
  fprintf(“%.2f %.2f, %.2f\n”, p[0][0], p[0][1], p[0][2]);
  fprintf(“%.2f %.2f, %.2f\n”, p[1][0], p[1][1], p[1][2]);
  fprintf(“%.2f %.2f, %.2f\n”, p[2][0], p[2][1], p[1][2]);
  fclose(fp);
}
```

Draw the contents of the text file