CS 31: Intro to Systems C Programming

L03: C programming & Data representation

Vasanta Chaganti & Kevin Webb

Swarthmore College

September 12, 2023

Announcements

- HW1 is due Thursday before class
  - up to groups of four
  - invitations sent from gradescope

- Lab 1 is due Thursday, 11.59 PM

- Clickers will count for credit from this week

Reading Quiz

- Note the red border!
- 1 minute per question
- No talking, no laptops, phones during the quiz

Class today…let's try something different

- reading quiz (5 mins)
- content block /recap (15 mins)
- group discussions (15 mins)
- content block 2 (10 mins)
- group discussions (15 mins)

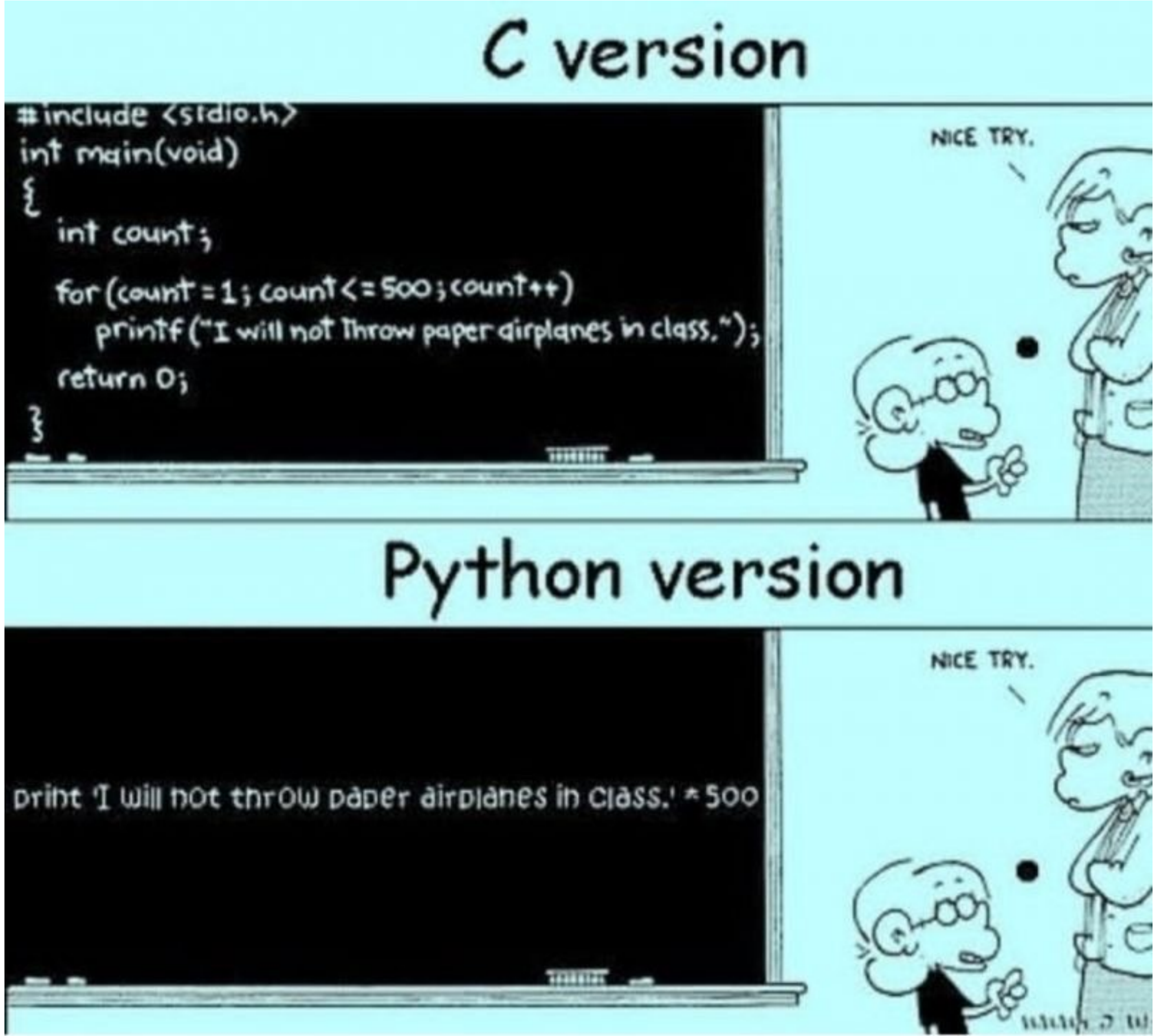
---end of class---

Agenda

- C programming
  - arrays, strings
  - functions and stack diagrams
  - structs
  - C is NOT the focus of this course: ask questions if you have them!

- Data representation
  - number systems + conversion
  - data types, storage
  - sizes, representation
  - signedness

Python versus C: Paradigms



https://devrant.com/rants/1755638/c-vs-python

Recap

Recap: Types in C

- All variables have an explicit type!
  - <variable type> <variable name>;
- Examples:

```
int humidity;            float temperature;
humidity = 20;           temperature = 32.5
```

Recap: An Example with Local Variables

```c
/* a multiline comment:
   anything between slashdot and dotslash */

#include <stdio.h> // C's standard I/O library (for printf)

int main() {
   // first: declare main's local variables
   int x, y;
   float z;

   // followed by: main function statements
   x = 6;
   y = (x + 3)/2; //x and y are both ints
   z = x; //z is a float, value of x is converted to a float
   z = (z + 3)/2;

   printf(…)  // Print x, y, z
}
```

Clicker choices

	X	Y	Z
A	4	4	4
B	6	4	4
C	6	4.5	4
D	**6**	**4**	**4.5**
E	6	4.5	4.5

Recap: Boolean values in C

- **Zero (0) is false, any non-zero value is true**
- **Logical** (operands int "boolean"->result int "boolean"):
  - ! (not): inverts truth value
  - && (and): true if both operands are true
  - || (or): true if either operand is true

Do the following statements evaluate to True or False?

#1: (!10) || (5 > 2)

#2: (-1) && ((!5) > -1)

Clicker choices

	#1	#2
A	True	True
B	True	False
C	False	True
D	False	False

Recap: Conditional Statements

Chaining if-else if

```
if(<boolean expr1>) {
  if-expr1-true-body
} else if (<bool expr2>){
  else-if-expr2-true-body
  (expr1 false)
}
...
} else if (<bool exprN>){
  else-if-exprN-true-body
}
```

With optional else:

```
if(<boolean expr1>) {
  if-expr1-true-body
} else if (<bool expr2>){
  else-if-expr2-true-body
}
...
} else if (<bool exprN>){
  else-if-exprN-true-body
} else {
  else body
  (all exprX's false)
}
```

Very similar to Python, just remember { } are blocks

Recap: For loops: different than Python's

```
for (<init>; <cond>; <step>) {
    for-loop-body-statements
}
<next stmt after loop>;
```

1. Evaluate <init> one time, when first eval **for** statement
2. Evaluate <cond>, if it is false, drop out of the loop (<next stmt after>)
3. Evaluate the statements in the for loop body
4. Evaluate <step>
5. Goto step (2)

```
for(i=1; i <= 10; i++) {  // example for loop
    printf("%d\n", i*i);
}
```

What does this for loop print?

Recap: While Loops

Basically identical to Python while loops:

```
while(<boolean expr>) {
    while-expr-true-body
}
```

```
x = 20;
while (x < 100) {
  y = y + x;
  x += 4;  //  x = x + 4;
}
<next stmt after loop>;
```

```
x = 20;
while(1) {
  y = y + x;
  x += 4;
  if(x >= 100) {
     break;  // break out of loop
  }
}
<next stmt after loop>;
```

Data Collections in C

- Many complex data types out there (CS 35)

- C has a few simple ones built-in:
  - Arrays
  - Strings (arrays of characters)
  - Structures (`struct`)

- Often combined in practice, e.g.:
  - An array of structs
  - A struct containing strings

Arrays and Strings

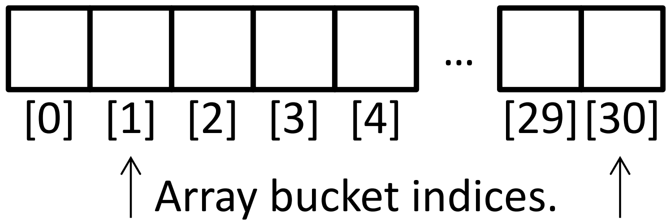
- C's support for collections of values
  - Array buckets store a single type of value
  - There is no "string" data type ☹
  - Specify max capacity (num buckets) when you declare an array variable (single memory chunk)

```
<type> <var_name>[<num buckets>];
int arr[5];  // an array of 5 integers
float rates[40]; // an array of 40 floats
```

Array Characteristics

```
int january_temps[31];  // Daily high temps
```

“january_temps” Location of [0] in memory.

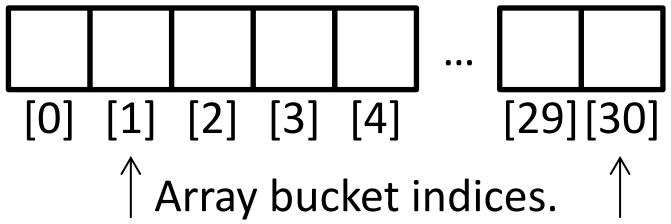


- Indices start at 0! Why?
- Array variable name means, to the compiler, the beginning of the memory chunk. (The memory address)
  - january_temps” (without brackets!) <u>Location of [0] in memory.</u>
  - Keep this in mind, we’ll return to it soon (functions).

Array Characteristics

```
int january_temps[31];  // Daily high temps
```

"january_temps"
Location of [0] in
memory.



- Indices start at 0! Why?
- The index refers to an offset from the start of the array
  - e.g., `january_temps`[3] means "three integers forward from the starting address of january_temps"

Characters and Strings

A character (type `char`) is numerical value that holds one letter.

char my_letter = ‘w’;   // Note: single quotes

What is the numerical value?

- printf(“%d   %c”, my_letter, my_letter);
- Would print: 119   w

Why is ‘w’ equal to 119?

- ASCII Standard says so.
- American Standard Code for Information Interchange

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	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	Neg. 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 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	Substitution	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	□

Characters and Strings

$ man ascii

119 = w

Characters and Strings

- A character (type `char`) is numerical value that holds one letter.
- A string is a memory block containing characters, one after another…

Hmm, suppose we used printf and %s to print name.

How does it know where the string ends and other memory begins?

- Examples:

  char food[6] = “Pizza”;

name	P	i	z	z	a	(Other memory)
	[0]	[1]	[2]	[3]	[4]	

0 is the
"Null character"

Special stuff over here in the lower values.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	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	Neg. 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 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	Substitution	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	□

Characters and Strings

$ man ascii

Characters and Strings

- A character (type `char`) is numerical value that holds one letter.
- A string is a memory block containing characters, one after another, with a null terminator (numerical 0) at the end.
- Examples:

  char name[20] = “Pizza”;

P	i	z	z	a	\0			…		
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]		[18]	[19]

Strings in C

- C String library functions: `#include <string.h>`
  - Common functions (strlen, strcpy, etc.) make strings easier
  - Less friendly than Python strings

- More on strings later, in labs.

- For now, remember about strings:
  - Allocate enough space for null terminator!
  - If you're modifying a character array (string), don't forget to set the null terminator!
  - If you see crazy, unpredictable behavior with strings, check these two things!

Functions and Stack Diagrams

Functions: Specifying Types

Need to specify the return type of the function, and the type of each parameter:

```
<return type> <func name> ( <param list> ) {
    // declare local variables first
    // then function statements
    return <expression>;
}

// my_function takes 2 int values and returns an int
int my_function(int x, int y) {
  int result;
  result = x;
  if(y > x) {
    result = y+5;
  }
  return result*2;
}
```

Compiler will yell at you if you try to pass the wrong type!

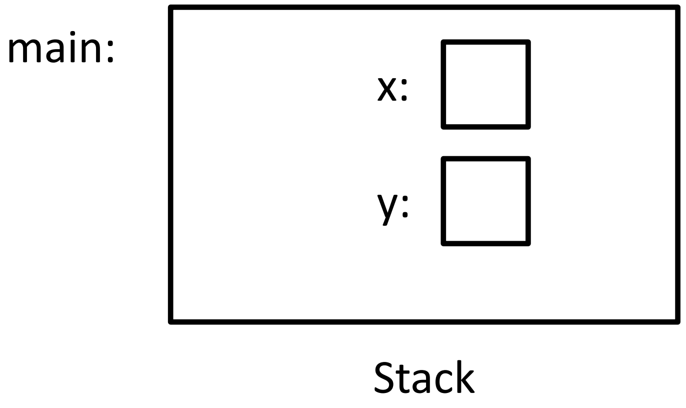
Function Arguments

Arguments are **passed by value**

- The function gets a separate <u>copy</u> of the passed variable

```
int func(int a, int b) {
    a = a + 5;
    return a - b;
}

int main() {
    // declare two integers
    int x, y;
    x = 4;
    y = 7;
    y = func(x, y);
    printf("%d, %d", x, y);
}
```



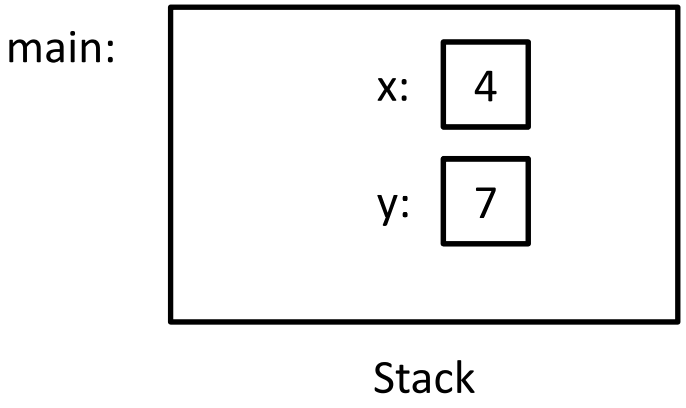
Function Arguments

Arguments are **passed by value**

- The function gets a separate copy of the passed variable

```c
int func(int a, int b) {
    a = a + 5;
    return a - b;
}

int main() {
    // declare two integers
    int x, y;
    x = 4;
    y = 7;
    y = func(x, y);
    printf("%d, %d", x, y);
}
```



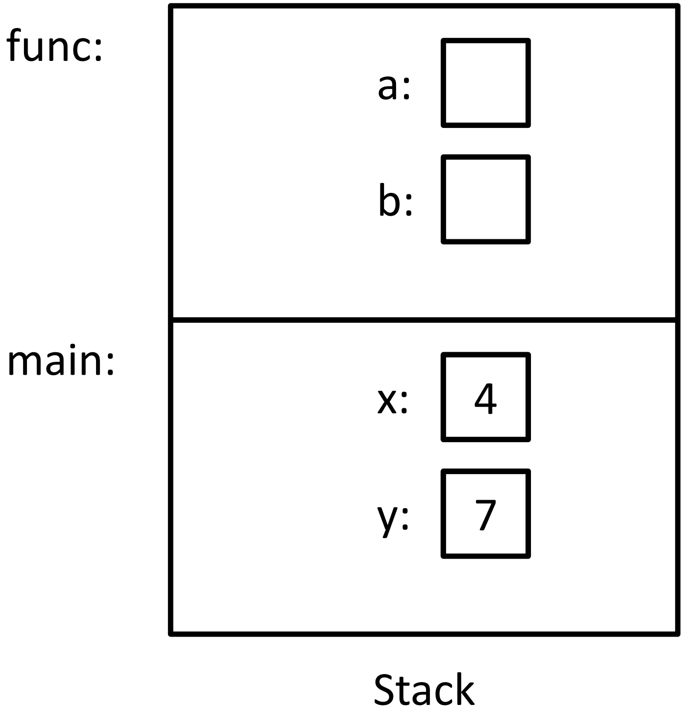
Function Arguments

Arguments are **passed by value**

- The function gets a separate copy of the passed variable

```
int func(int a, int b) {
    a = a + 5;
    return a - b;
}

int main() {
    // declare two integers
    int x, y;
    x = 4;
    y = 7;
    y = func(x, y);
    printf("%d, %d", x, y);
}
```



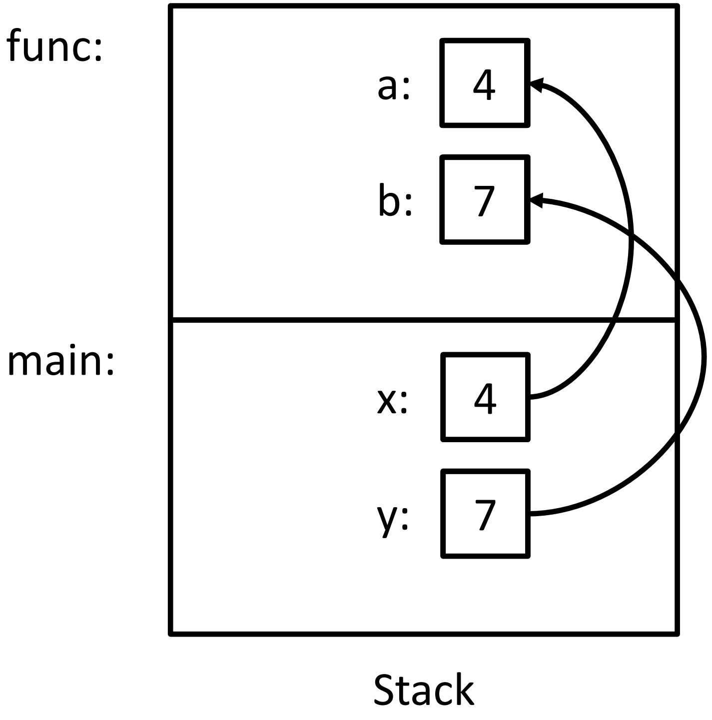
Function Arguments

Arguments are **passed by value**

– The function gets a separate copy of the passed variable

```
int func(int a, int b) {
    a = a + 5;
    return a - b;
}

int main() {
    // declare two integers
    int x, y;
    x = 4;
    y = 7;
    y = func(x, y);
    printf("%d, %d", x, y);
}
```



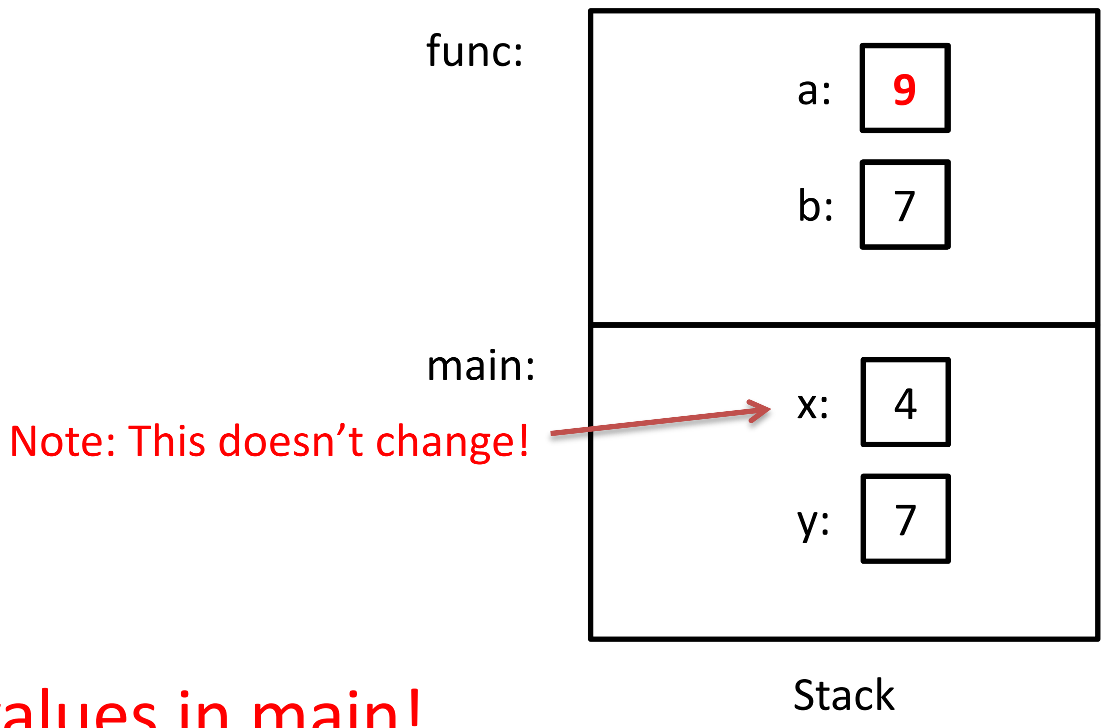
Function Arguments

Arguments are **passed by value**

- The function gets a separate copy of the passed variable

```c
int func(int a, int b) {
    a = a + 5;
    return a - b;
}

int main() {
    // declare two integers
    int x, y;
    x = 4;
    y = 7;
    y = func(x, y);
    printf(“%d, %d”, x, y);
}
```



No impact on values in main!

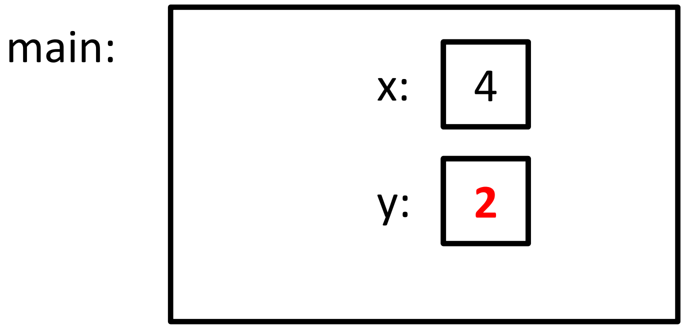
Function Arguments

Arguments are **passed by value**

– The function gets a separate copy of the passed variable

```
int func(int a, int b) {
    a = a + 5;
    return a - b;
}

int main() {
    // declare two integers
    int x, y;
    x = 4;
    y = 7;
    y = func(x, y);
    printf(“%d, %d”, x, y);
}
```



Stack

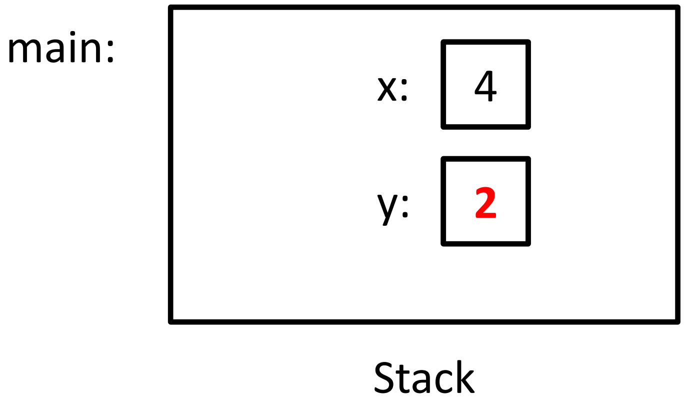
Function Arguments

Arguments are **passed by value**

– The function gets a separate copy of the passed variable

```c
int func(int a, int b) {
    a = a + 5;
    return a - b;
}

int main() {
    // declare two integers
    int x, y;
    x = 4;
    y = 7;
    y = func(x, y);
    printf("%d, %d", x, y);
}
```



Output: 4, 2

What will this print?

```
int func(int a, int y, int my_array[]) {
    y = 1;
    my_array[a] = 0;
    my_array[y] = 8;
    return y;
}

int main() {
    int x;
    int values[2];

    x = 0;
    values[0] = 5;
    values[1] = 10;

    x = func(x, x, values);

    printf("%d, %d, %d", x, values[0], values[1]);
}
```

A. 0, 5, 8
B. 0, 5, 10
C. 1, 0, 8
D. 1, 5, 8
E. 1, 5, 10

Hint: What does the name of an array mean to the compiler?

What will this print?

```
int func(int a, int y, int my_array[]) {
    y = 1;
    my_array[a] = 0;
    my_array[y] = 8;
    return y;
}

int main() {
    int x;
    int values[2];

    x = 0;
    values[0] = 5;
    values[1] = 10;

    x = func(x, x, values);

    printf("%d, %d, %d", x, values[0], values[1]);
}
```

A. 0, 5, 8
B. 0, 5, 10
C. 1, 0, 8
D. 1, 5, 8
E. 1, 5, 10

Hint: Still accessing the same memory location of array in func

Discussion Block 1

structs

- Treat a collection of values as a single type:
  - C is not an object oriented language, no classes
  - A `struct` is similar to the data part of a class

- Rules:
  1. Define a new struct type outside of any function
  2. Declare variables of the new struct type
  3. Use dot notation to access the field values of a struct variable

Struct Example

Suppose we want to represent a *student* type.

```
struct student {
    char name[20];
    int grad_year;
    float gpa;
};
// Variable bob is of type struct student
struct student bob;
// Set name (string) with strcpy()
strcpy(bob.name, “Robert Paulson”);
bob.grad_year = 2019;
bob.gpa = 3.1;

printf(“Name: %s, year: %d, GPA: %f”, bob.name, bob.grad_year, bob.gpa);
```

Arrays of Structs

```
struct student {
    char name[20];
    int grad_year;
    float gpa;
};
//create an array of struct students!
struct student classroom[50];

strcpy(classroom[0].name, “Alice”);
classroom[0].grad_year = 2023
classroom[0].gpa = 4.0;
```

```
// With a loop, create an army of Alice clones!
int i;
for (i = 0; i < 50; i++) {
        strcpy(classroom[i].name, “Alice”);
        classroom[i].grad_year = 2023;
        classroom[i].gpa = 4.0;
}
```

Arrays of Structs

```
struct student classroom[3];

strcpy(classroom[0].name, "Alice");
classroom[0].grad_year = 2021;
classroom[0].gpa = 4.0;

strcpy(classroom[1].name, "Bob");
classroom[1].grad_year = 2022;
classroom[1].gpa = 3.1

strcpy(classroom[2].name, "Cat");
classroom[2].grad_year = 2023;
classroom[2].gpa = 3.4
```

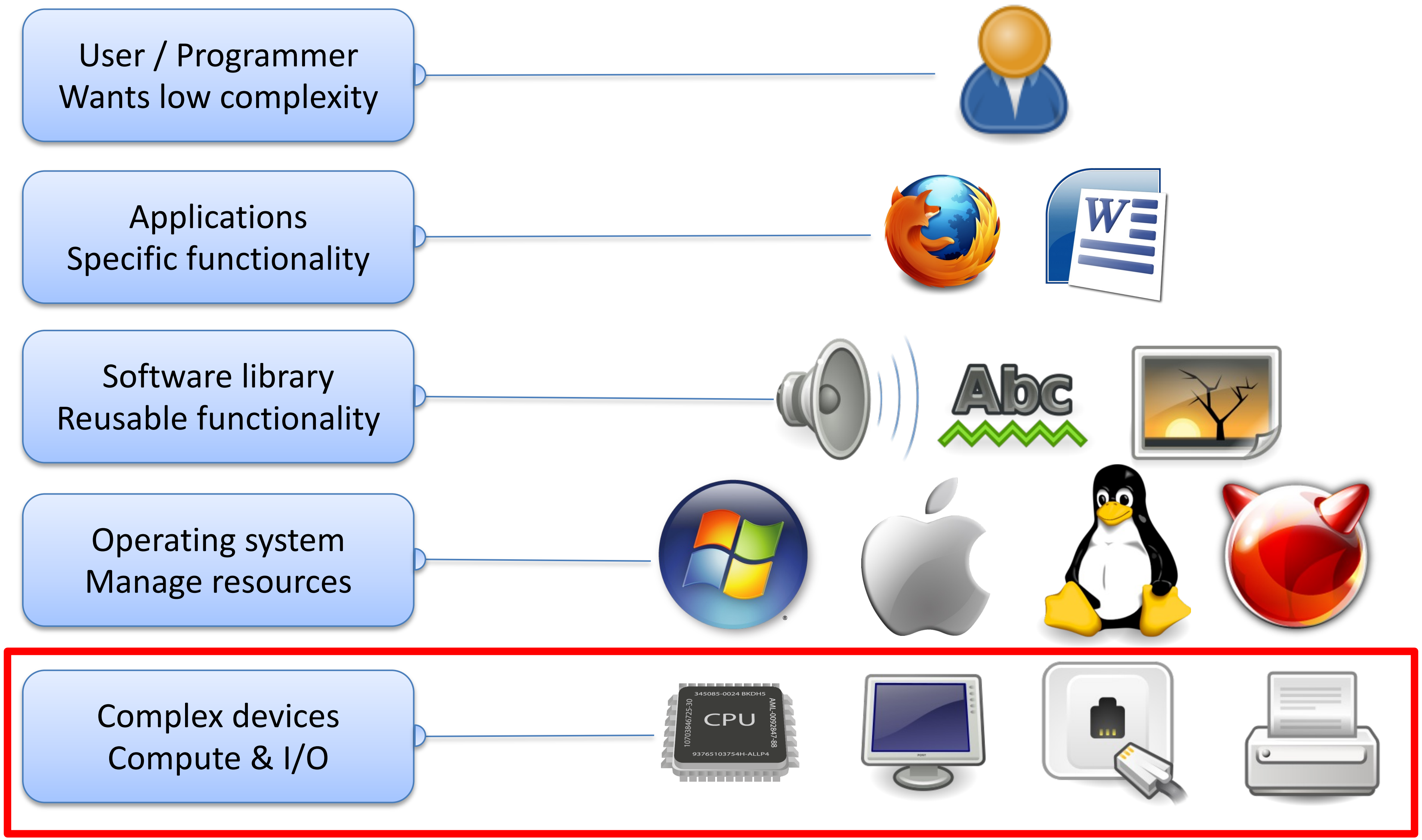
Array of Structs: Layout in Memory

`classroom: array of structs`

‘A’	‘l’	‘i’	‘c’	‘e’	‘\0’	…	‘B’	‘o’	‘b’	‘\0’	…	‘C’	‘a’	‘t’	‘\0’	…
2021							2022					2023				
4.0							3.1					3.4				

[0] [1] [2]

Abstraction



Data Storage

- Lots of technologies out there:
  - Magnetic (hard drive, floppy disk)
  - Optical (CD / DVD / Blu-Ray)
  - Electronic (RAM, registers, …)
- Focus on electronic for now
  - We’ll see (and build) digital circuits soon
- Relatively easy to differentiate two states
  - Voltage present
  - Voltage absent

Bits and Bytes

- Bit: a 0 or 1 value (binary)
  - HW represents as two different voltages
    - 1: the presence of voltage (high voltage)
    - 0: the absence of voltage (low voltage)

- Byte: 8 bits, the smallest addressable unit

Memory:	01010101	10101010	00001111	…
(address)	[0]	[1]	[2]	…

- Other names:
  - 4 bits: Nibble
  - "Word": Depends on system, often 4 bytes

Files

Sequence of bytes… nothing more, nothing less

Binary Digits (BITs)

- One bit: two values (0 or 1)
- Two bits: four values (00, 01, 10, or 11)
- Three bits: eight values (000, 001, …, 110, 111)

How many unique values can we represent with 9 bits? Why?

- One bit: two values (0 or 1)
- Two bits: four values (00, 01, 10, or 11)
- Three bits: eight values (000, 001, …, 110, 111)

A. 18
B. 81
C. 256
D. 512
E. Some other number of values.

How many unique values can we represent with 9 bits? Why?

- One bit: two values (0 or 1)
- Two bits: four values (00, 01, 10, or 11)
- Three bits: eight values (000, 001, …, 110, 111)

A. 18
B. 81
C. 256
D. 512
E. Some other number of values.

How many values?

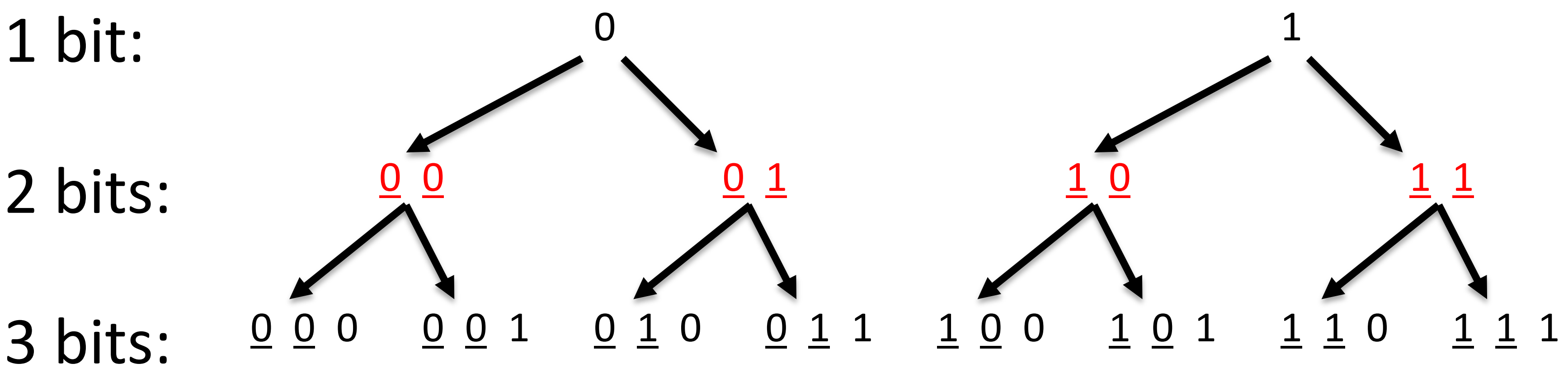
1 bit: 0 1

How many values?

1 bit: 0 1

2 bits: 0 0 0 1 1 0 1 1

How many values?



How many values?

1 bit: 0 1

2 bits: 0 0 0 1 1 0 1 1

3 bits: 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1

4 bits:

0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	16 values
0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	
1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	
1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1	

N bits: 2^N values

C types and their (typical!) sizes

- `1 byte:  char, unsigned char`
- `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 v1;
short s1;
long long ll;
```

WARNING: These sizes are **NOT** a guarantee. Don't always assume that every system will use these values!

```
// prints out number of bytes
printf("%lu %lu %lu\n", sizeof(v1), sizeof(s1), sizeof(ll));
```

How do we use this storage space (bits) to represent a value?

Let's start with what we know…

- Digits 0-9
- <u>Positional numbering</u>
- Digits are composed to make larger numbers
- Known as Base 10 representation

Decimal number system (Base 10)

- Sequence of digits in range [0, 9]

64025

Digit #0: 1’s place, “least significant digit”

Digit #1: 10’s place

Digit #4: “most significant digit”

Decimal: Base 10

A number, written as the sequence of N digits,

$$d_{n-1} \ldots d_2\, d_1\, d_0$$

where d is in {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, represents the value:

$$[d_{n-1} * 10^{n-1}] + [d_{n-2} * 10^{n-2}] + \ldots + [d_1 * 10^1] + [d_0 * 10^0]$$

64025 =

$6 * 10^4 \quad + \quad 4 * 10^3 \quad + \quad 0 * 10^2 \quad + \quad 2 * 10^1 \quad + \quad 5 * 10^0$

60000 + 4000 + 0 + 20 + 5

Binary: Base 2

- Used by computers to store digital values.

- Indicated by prefixing number with **0b**

- A number, written as the sequence of N digits, $d_{n-1}...d_2d_1d_0$, where d is in {0,1}, represents the value:

$$[d_{n-1} * 2^{n-1}] + [d_{n-2} * 2^{n-2}] + ... + [d_2 * 2^2] + [d_1 * 2^1] + [d_0 * 2^0]$$

Converting Binary to Decimal

Most significant bit → $\underline{1}000111\underline{1}$ ← Least significant bit

7 6 5 4 3 2 1 0

Representation: $1 \times 2^7 + 0 \times 2^6 \ldots + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$128 + \quad + 8 + 4 + 2 + 1$

10001111 = 143

Hexadecimal: Base 16

- <u>Indicated by prefixing number with **0x**</u>

A number, written as the sequence of N digits,

$$d_{n-1}\ldots d_2 d_1 d_0,$$

where d is in {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, <u>A</u>, <u>B</u>, <u>C</u>, <u>D</u>, <u>E</u>, <u>F</u>}, represents:

$$[d_{n-1} * 16^{n-1}] + [d_{n-2} * 16^{n-2}] + \ldots + [d_2 * 16^2] + [d_1 * 16^1] + [d_0 * 16^0]$$

Generalizing: Base b

- The meaning of a digit depends on its position in a number.

A number, written as the sequence of N digits,

$$d_{n-1} \ldots d_2\, d_1\, d_0$$

in base b represents the value:

$$[d_{n-1} * b^{n-1}] + [d_{n-2} * b^{n-2}] + \ldots + [d_2 * b^2] + [d_1 * b^1] + [d_0 * b^0]$$

Base 10: $[d_{n-1} * 10^{n-1}] + [d_{n-2} * 10^{n-2}] + \ldots + [d_1 * 10^1] + [d_0 * 10^0]$

Other (common) number systems.

- Base 2: How data is stored in hardware.
- Base 8: Used to represent file permissions.
- Base 10: Preferred by people.
- Base 16: Convenient for representing memory addresses.
- Base 64: Commonly used on the Internet, (e.g. email attachments).

It's **all** stored as binary in the computer.

Different representations (or visualizations) of the same information!

Discussion block 2

Important Point…

- You can represent the same value in a variety of number systems or bases.

- It's **all** stored as binary in the computer.
  - Presence/absence of voltage.

Hexadecimal: Base 16

- Fewer digits to represent same value
  - Same amount of information!

- Like binary, the base is power of 2

- Each digit is a “nibble”, or half a byte.

Each hex digit is a “nibble”

- One hex digit: 16 possible values (0-9, A-F)
- $16 = 2^4$, so each hex digit has exactly four bits worth of information.
- We can map each hex digit to a four-bit binary value.
(helps for converting between bases)

Each hex digit is a "nibble"

Example value: 0x1B7

Four-bit value: 1

Four-bit value: B (decimal 11)

Four-bit value: 7

In binary: 0001 1011 0111

1 B 7

Converting Decimal -> Binary

- Two methods:
  - division by two remainder
  - powers of two and subtraction

Method 1: decimal value D, binary result b (b_i is ith digit):

```
i = 0
while (D > 0)
   if D is odd
         set b_i to 1
   if D is even
         set b_i to 0
   i++
   D = D/2
```

Example: Converting 105

idea:

example: D = 105 b_0 = 1

Method 1: decimal value D, binary result b (b_i is ith digit):

```
i = 0
while (D > 0)
   if D is odd
          set bi to 1
   if D is even
          set bi to 0
   i++
   D = D/2
```

Example: Converting 105

```
idea:     D             example: D = 105        b0 = 1
          D = D/2                D =  52        b1 = 0
```

Method 1: decimal value D, binary result b (b_i is ith digit):

```
i = 0
while (D > 0)
   if D is odd
          set b_i to 1
   if D is even
          set b_i to 0
   i++
   D = D/2
```

Example: Converting 105

```
idea:    D              example: D = 105        b0 = 1
         D = D/2                 D =  52        b1 = 0
         D = D/2                 D =  26        b2 = 0
         D = D/2                 D =  13        b3 = 1
         D = D/2                 D =   6        b4 = 0
         D = D/2                 D =   3        b5 = 1
         D = D/2                 D =   1        b6 = 1
         D = 0 (done)            D =   0        b7 = 0

                                     105   =   01101001
```

Method 2

- $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, $2^5 = 32$, $2^6 = 64$, $2^7 = 128$
-

  To convert $\underline{105}$:
  - Find largest power of two that's less than 105 (64)
  - Subtract 64 ($105 - 64 = \underline{41}$), put a 1 in d_6
  - Subtract 32 ($41 - 32 = \underline{9}$), put a 1 in d_5
  - Skip 16, it's larger than 9, put a 0 in d_4
  - Subtract 8 ($9 - 8 = \underline{1}$), put a 1 in d_3
  - Skip 4 and 2, put a 0 in d_2 and d_1
  - Subtract 1 ($1 - 1 = \underline{0}$), put a 1 in d_0 (Done)

$\underline{1}$	$\underline{1}$	$\underline{0}$	$\underline{1}$	$\underline{0}$	$\underline{0}$	$\underline{1}$
d_6	d_5	d_4	d_3	d_2	d_1	d_0

What is the value of 357 in binary?

A. 101100011
B. 101100101
C. 101101001
D. 101110101
E. 110100101

$2^0 = 1, \quad 2^1 = 2, \quad 2^2 = 4, \quad 2^3 = 8, \quad 2^4 = 16,$

$2^5 = 32, \quad 2^6 = 64, \quad 2^7 = 128, \quad 2^8 = 256$