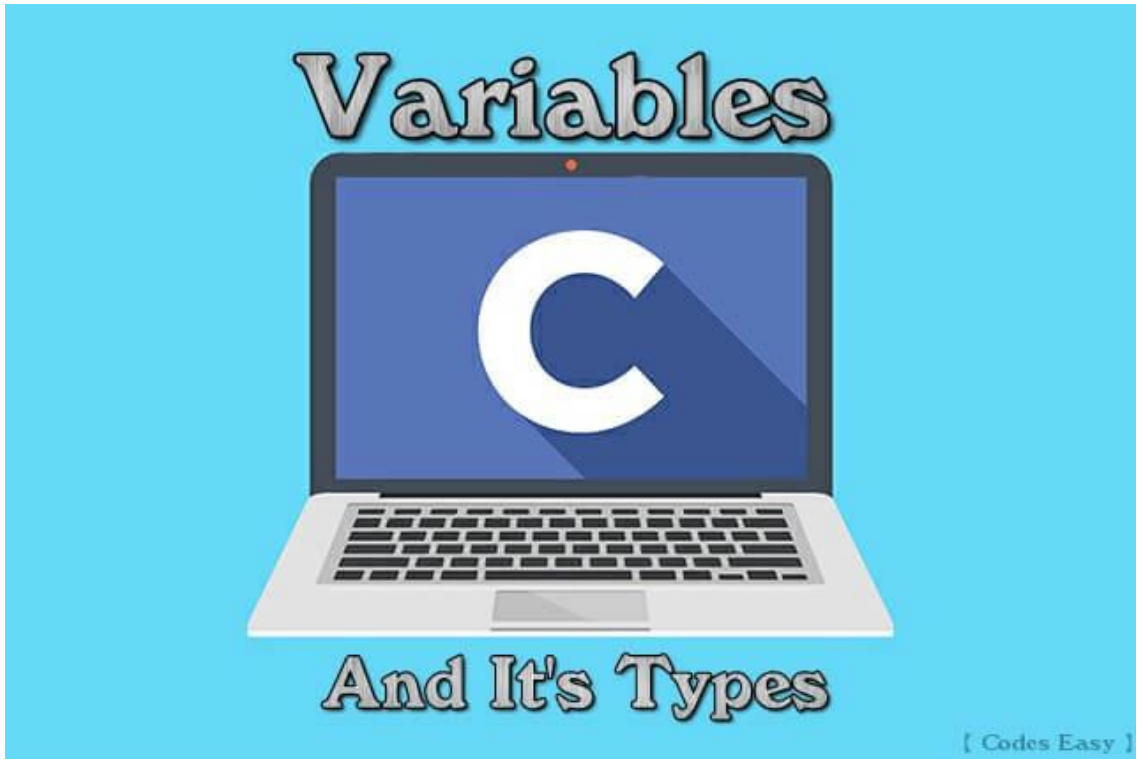
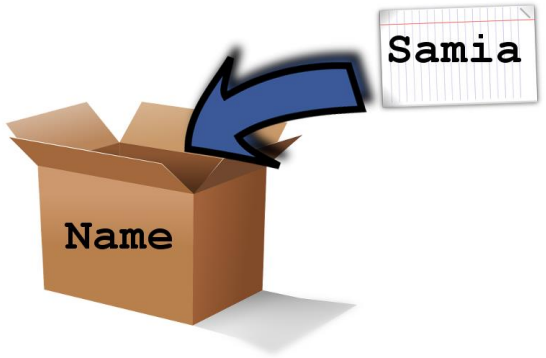


C Variables

Managing data and Values



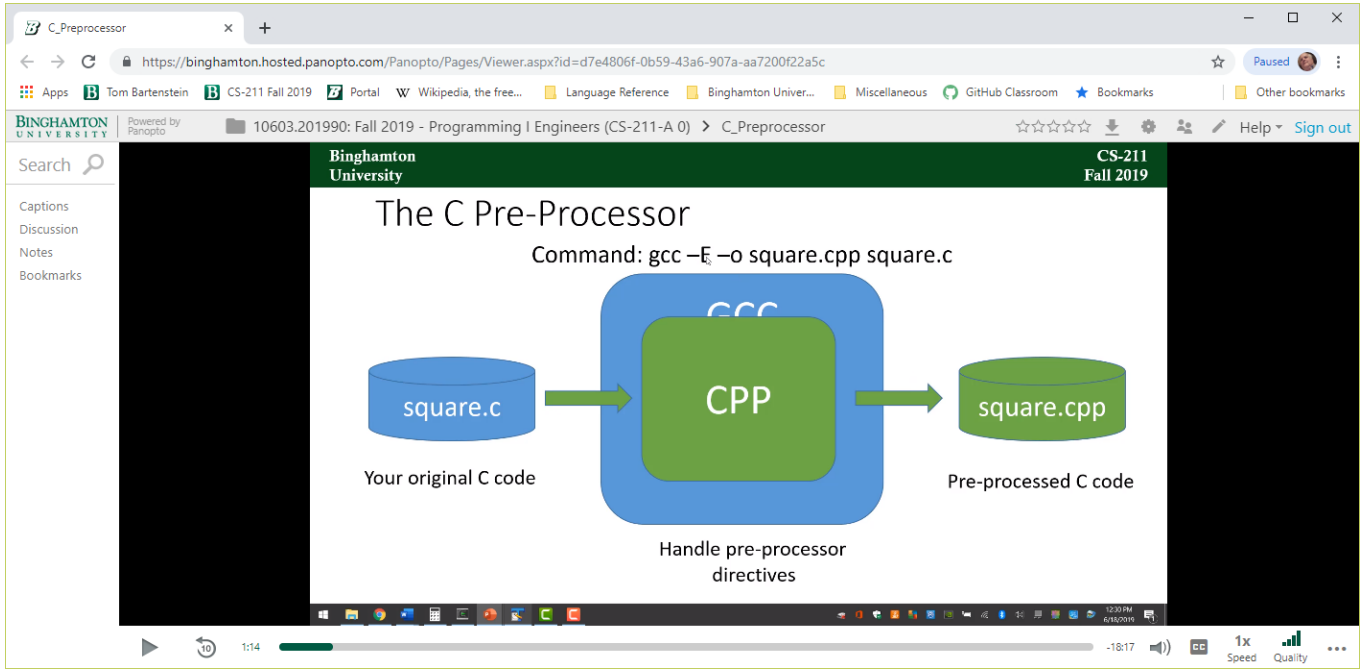
iClicker Attendance

Please click on A if you are here:

A. I am here today.

Video Review: C Preprocessor

- #include
- #define
- #ifdef
- Pre-processed Code
- Questions???



Class Exercise: Pre-Processor

- Identify all the pre-processor statements in helloWorld.c

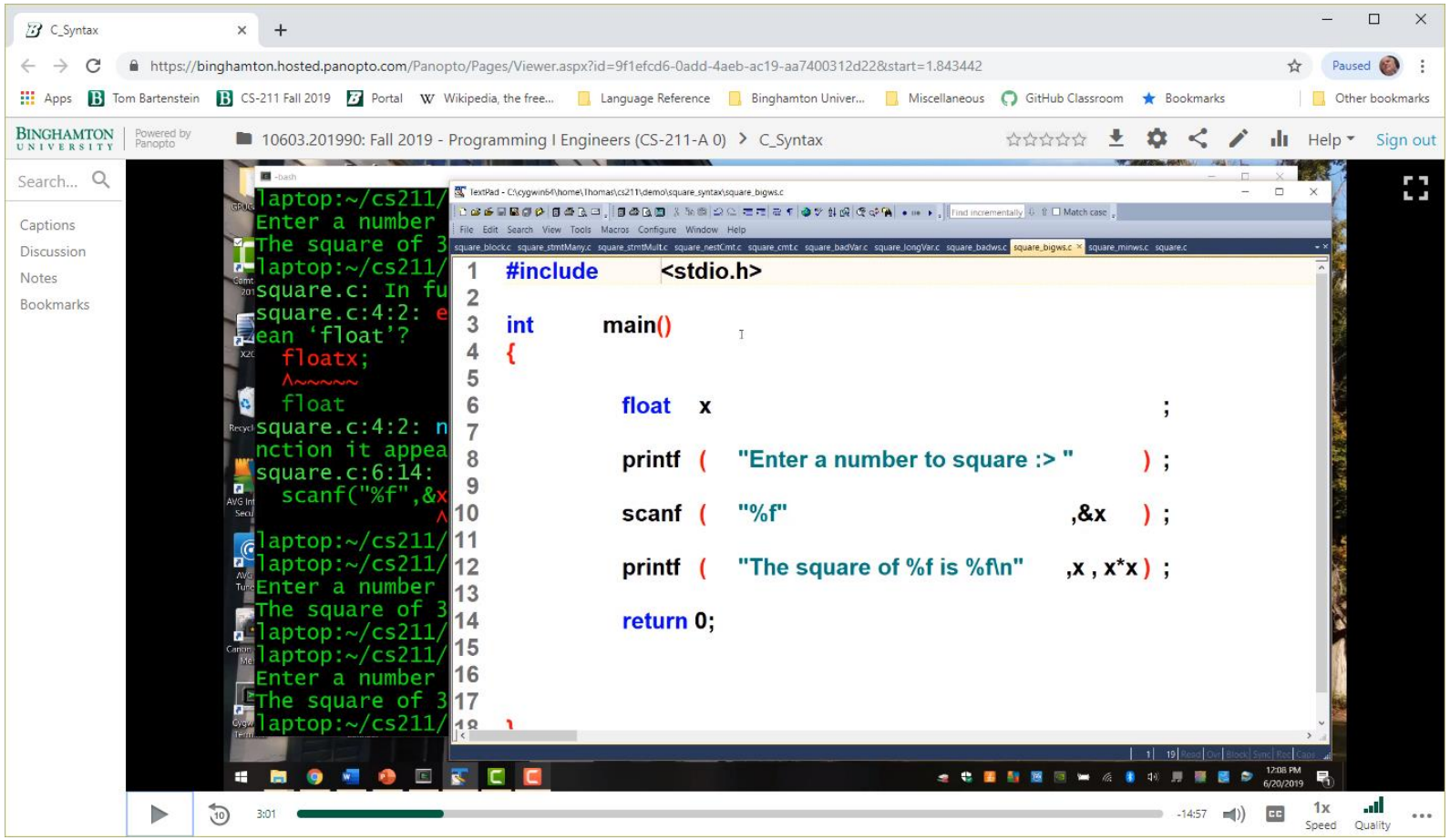
```
#include <stdio.h>

int main() {
    printf("Hello World!\n");
    return 0;
}
```

- What does the pre-processor do to this code?

Video Review: C Syntax

- Whitespace
- Identifiers
- Comments
- Statements
- Blocks
- Questions?



iClicker Question

Please click on the best choice:

A. I like to use indentation to keep track of C blocks of code, and blank lines to separate “paragraphs” of code.
B. I just type in code starting at column zero with no indentation or blank lines.
C. I like to put all my C statements on a single line in the file. If it was hard to write, it should be hard to read!

Class Exercise: Identifiers

- Which of the following are legal identifiers in C?
  - number_of_inputs_supplied_by_the_user
  - float
  - z
  - widgets
  - count@start
  - v354
  - for_while
  - a34QZZwx43_Rrrrrz0591101
  - repeat_cnt

Class Exercise: Identifiers

- Which of the following are useful identifiers in C?
  - number_of_inputs_supplied_by_the_user
  - float
  - z
  - widgets
  - count@start
  - v354
  - for_while
  - a34QZZwx43_Rrrrrz0591101
  - repeat_cnt

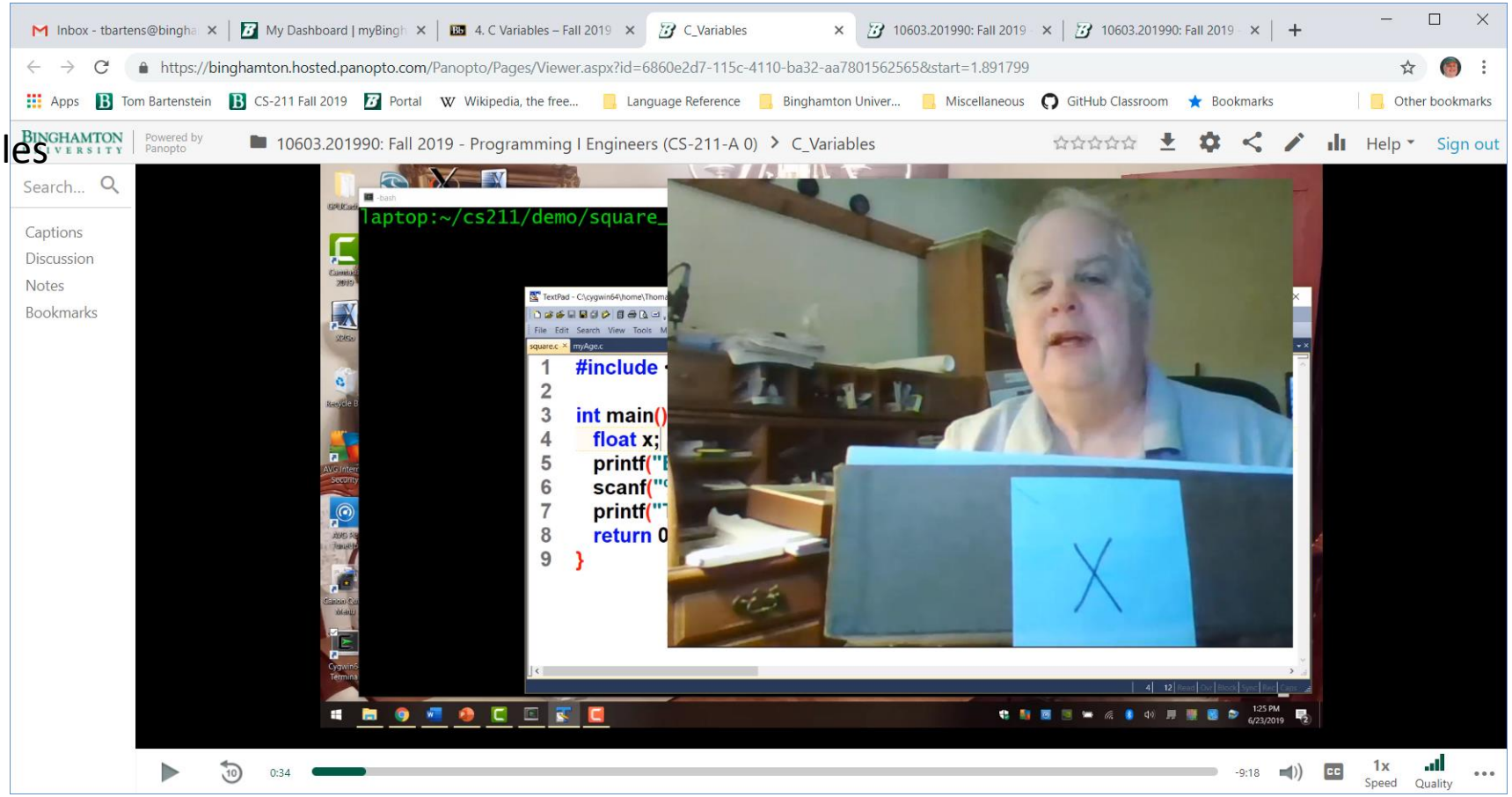
Class Exercise: Comments

- What does the following code print?

```
#include <stdio.h>
int main() {
    printf("Hello World!\n");
    // printf("This is CS-211!\n");
    printf( /* "Watch your comments!\n");
    printf("Are you following this?\n"); /*
    printf("Having fun yet?\n"); */
    "Take me to your leader.\n"); /* that'll show em */
    return 0;
}
```

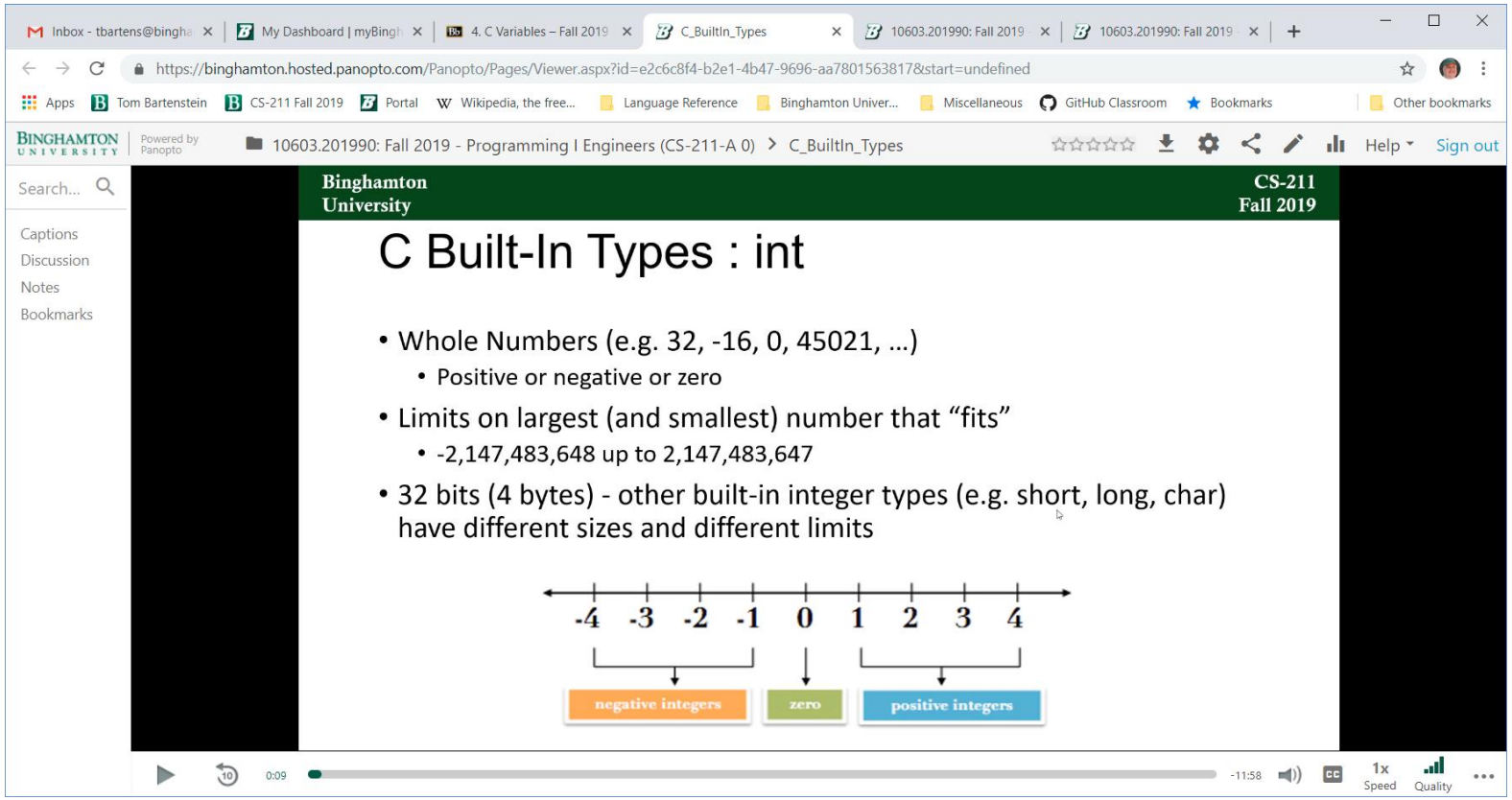
C Variables

- Watch the video, available on myCourses
  - Content
    - Videos
      - 4. C Variables



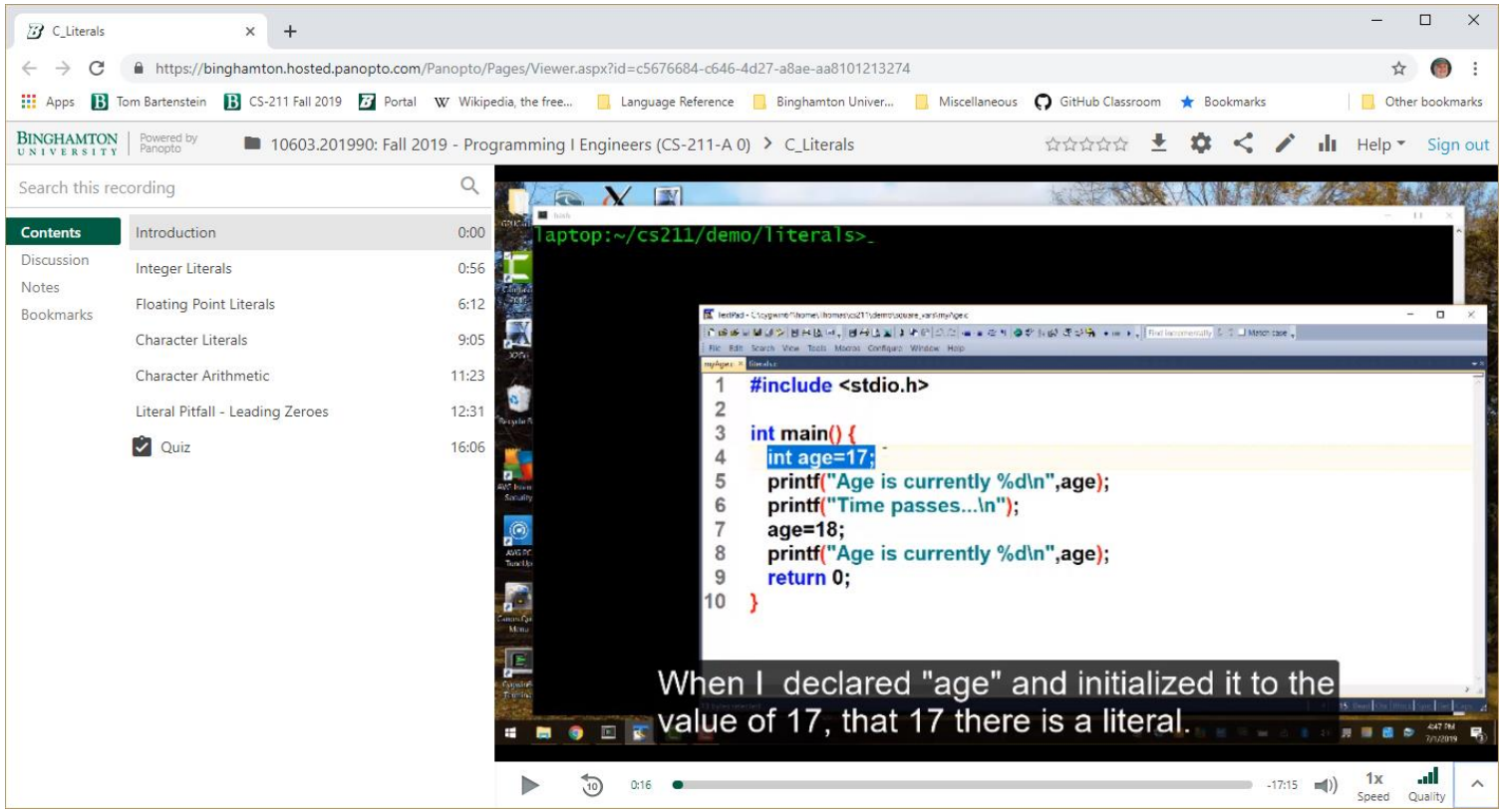
C Built-In Types

- Watch the video, available on myCourses
  - Content
    - Videos
      - 4. C Variables



C Literals

- Watch the video, available on myCourses
  - Content
    - Videos
      - 4. C Variables



Class Exercise: Mass of Atoms

A hydrogen atom consists of one proton and one electron. If the mass of an electron is approximately 9.1×10^{-31} kilograms, and the mass of a proton or a neutron is approximately 1.7×10^{-27} kilograms, what is the mass of a hydrogen atom?

A typical carbon atom contains 6 electrons, 6 protons, and 6 neutrons. What is the mass of a carbon atom?

Write a generalized program that asks for the element name, number of protons, neutrons, and electrons, and prints it's mass.

Prompting for values

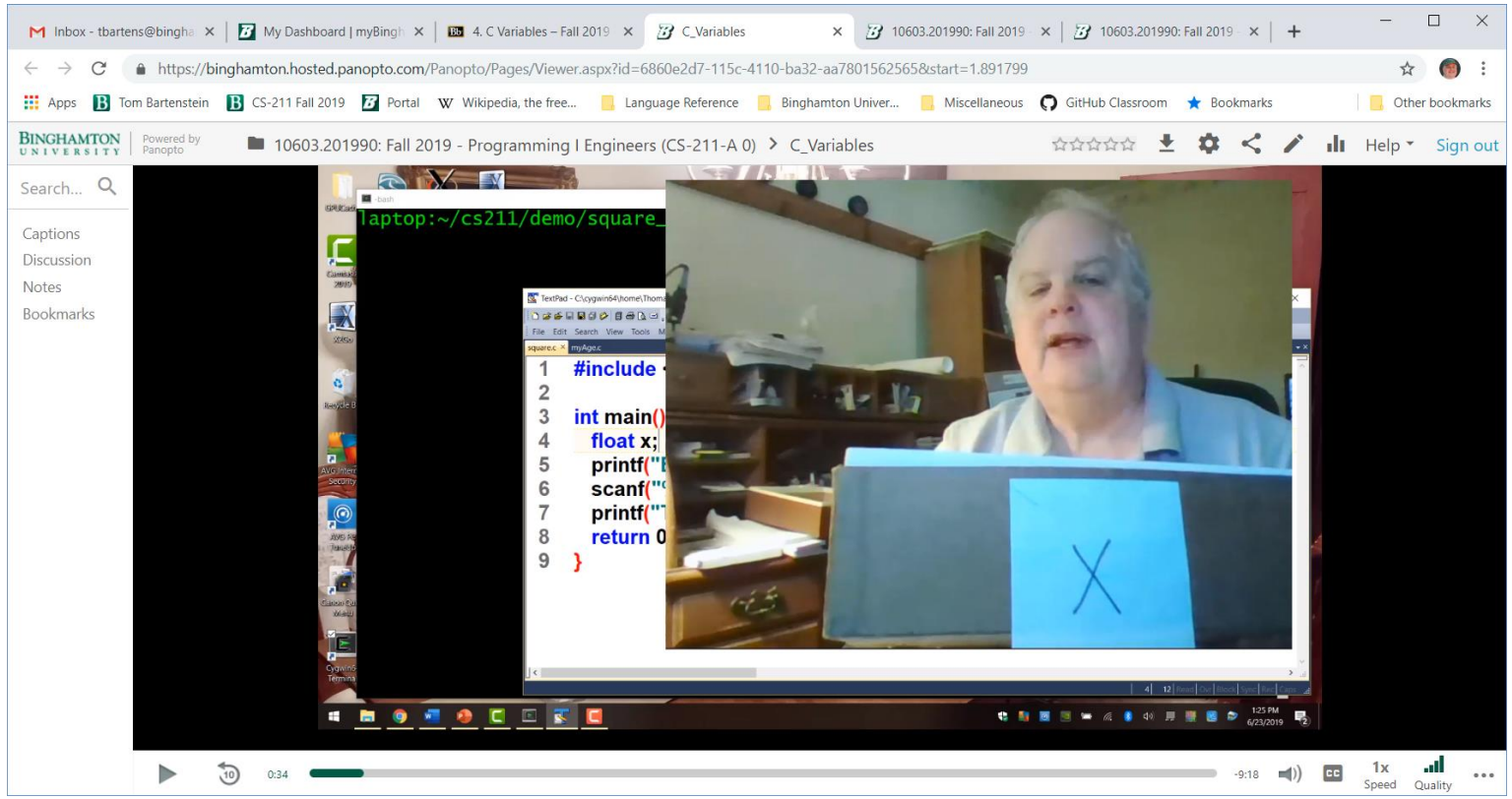
```
printf(“Enter element name, protons, neutrons, and electrons :>”);
char name[100]; // allow name to be up to 100 characters long
int protons=0;
int neutrons=0;
int electrons=0;
scanf(“ %s %d %d %d”,name,&protons,&neutrons,&electrons);
assert(protons>0);
assert(electrons>0);
```

Resources

- Programming in C, Chapter 3
- WikiPedia: C Syntax (https://en.wikipedia.org/wiki/C_syntax)
- WikiPedia: C Data Types (https://en.wikipedia.org/wiki/C_data_types)
- WikiPedia: Type Theory (https://en.wikipedia.org/wiki/Type_theory)

C Variables

Summary Notes



C Variables

- A variable is a named piece of data
- Variables in C have…
  - A name (specified by the programmer)
  - A value (may be unassigned/unknown)
  - A location in memory (determined by the compiler)
  - A type (size and interpretation)
  - … (more to come… scope/ storage class/ etc.)
- Variables must be declared before they are used!

Variable Declaration Statement

type name;

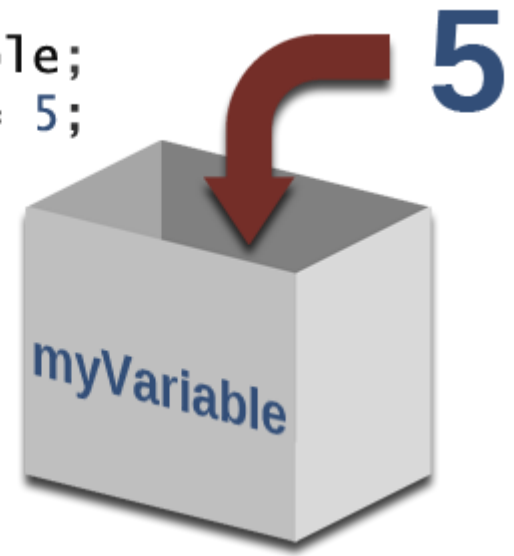
type : One of the built-in types or a derived type

name : Any valid identifier

Examples:

```
int age;
char First_Initial;
float gpa;
```

```
int myVariable;
myVariable = 5;
```



Warning – Variables may only be declared once!

Uninitialized Variables

- What is in a variable *before* you assign a value to it?

```
int x;
printf("The value of x is %d\n",x);
```

- Under the covers
  - The compiler chooses a place in memory for the x value
  - The starting value of x is whatever happened to be in memory at the location the compiler chose.
  - The variable x has some value, but you have no idea what that value is!
- In fact, the compiler warns about reading uninitialized variables.
- Moral: Always specify an initial value

Variable Declaration Statement (w/ init.)

type name = const;

type : One of the built-in types or a derived type

name : Any valid identifier

const : Constant specification (should be of type *type*)

Examples:

```
int age = 17;
char First_Initial = 'a';
float gpa = 3.985;
```

Pitfall : Initialization vs. Assignment

Initialization

```
int age=17;
…

int age=18; // got older
```

Initialization and Assignment

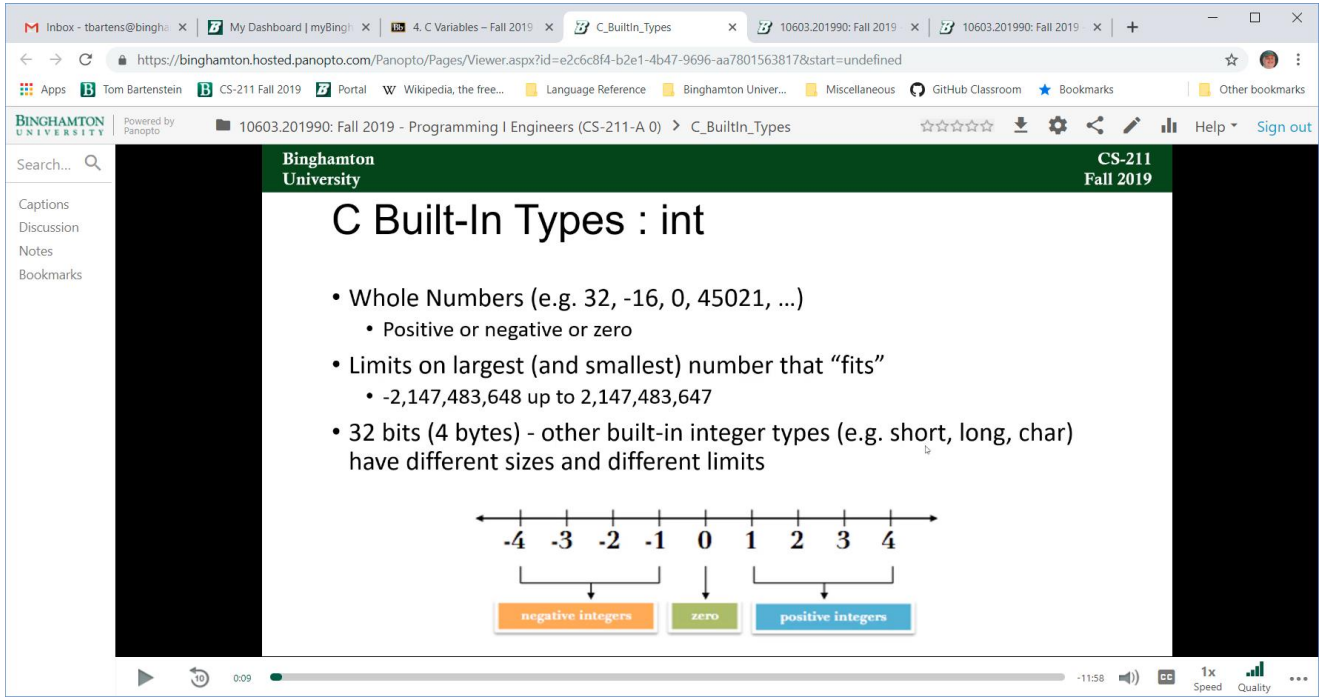
```
int age=17;
…

age=18; //  got older
```

Age is declared above, and cannot be declared again!

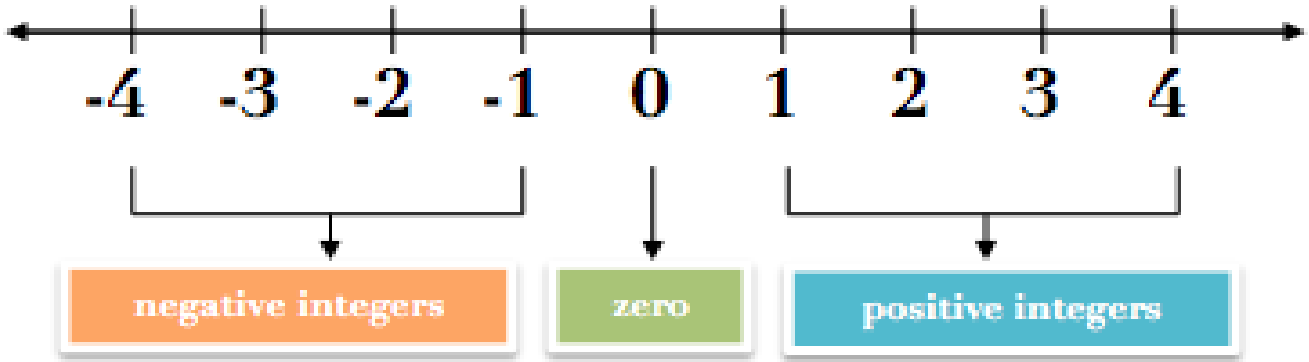
C Built-In Types

Summary Notes



C Built-In Types : int

- Whole Numbers (e.g. 32, -16, 0, 45021, …)
  - Positive or negative or zero
- Limits on largest (and smallest) number that “fits”
  - -2,147,483,648 up to 2,147,483,647
- 32 bits (4 bytes) - other built-in integer types (e.g. short, long, char) have different sizes and different limits



Integers – Internal Representation

- Number represented by a vector of *n* binary digits (1's or 0's)
- If leftmost bit is a zero, number is positive, simple binary

$$value = \sum_{i=0}^{n-1} b_i \times 2^i$$

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	1	0	1	0

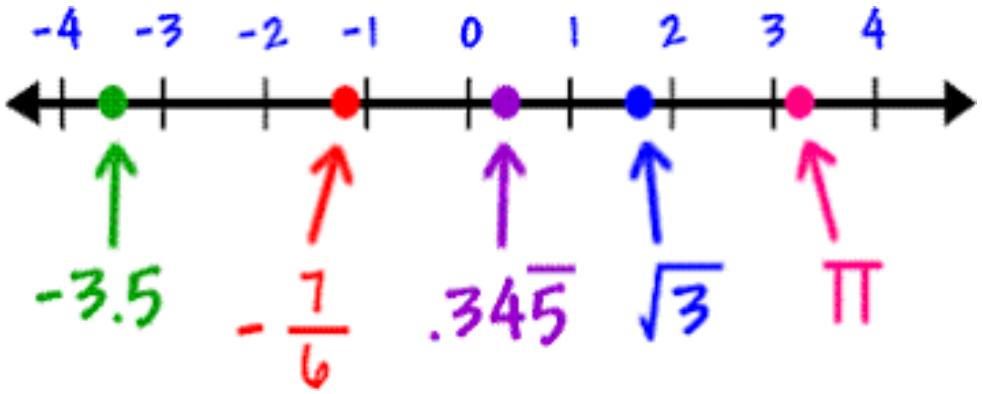
- If leftmost bit is a one, number is negative

$$value = \left(\sum_{i=0}^{n-1} b_i \times 2^i \right) - 2^n$$

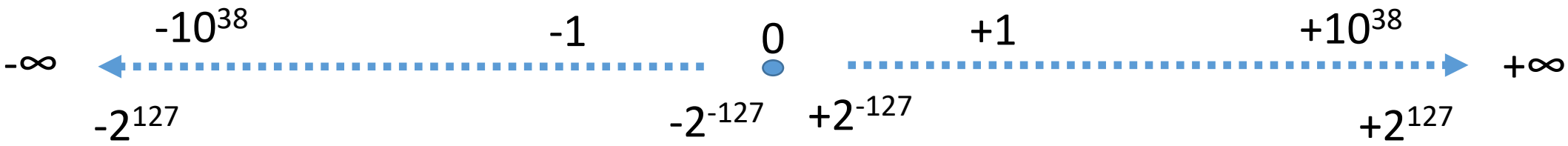
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	1	0	1	0

C Built-In Types: float

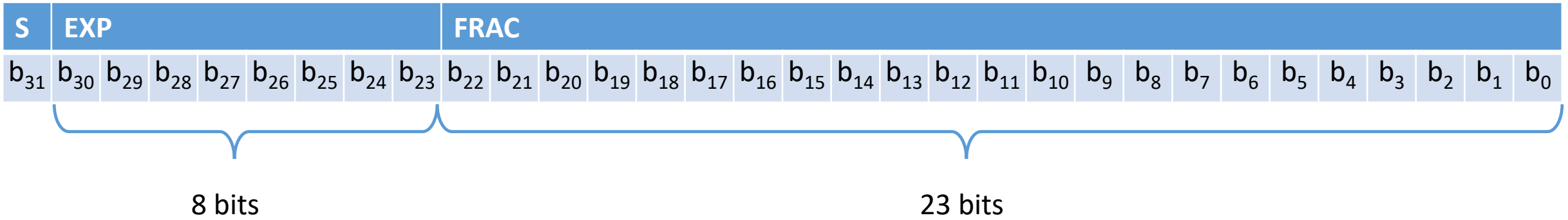
- Decimal numbers (e.g. 1.0, 3.1415, 6.22×10^{23} etc.)
- Limits on largest number that fits: ~ +/- 3.4×10^{38}
- Almost always approximate values! e.g. 1.0 != 3 x (0.333333333333)
- Special values for +/- ∞ +/-0, and +/- NaN (Not a Number)
- Other floating point types (e.g. double) have different sizes, different precision, and different limits



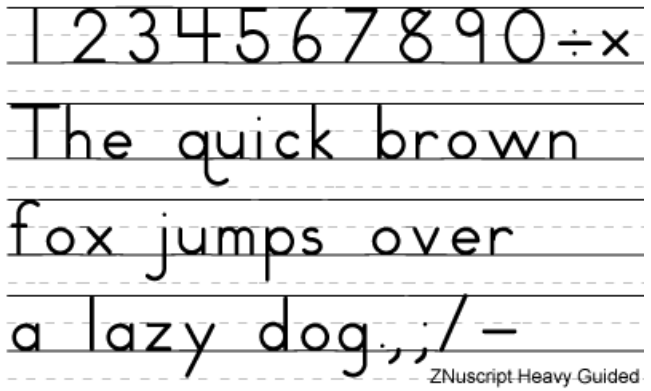
Float Internals: IEEE floating point std.



- Represent the number as: $-1^{S} \times 1.FRAC \times 2^{exp}$
- S = 0 (positive) or 1 (negative)
- $1 \leq FRAC < 2$ (except for special cases like 0 or +/- ∞)
- $-126 \leq \exp \leq 126$



C Built-In Types : char



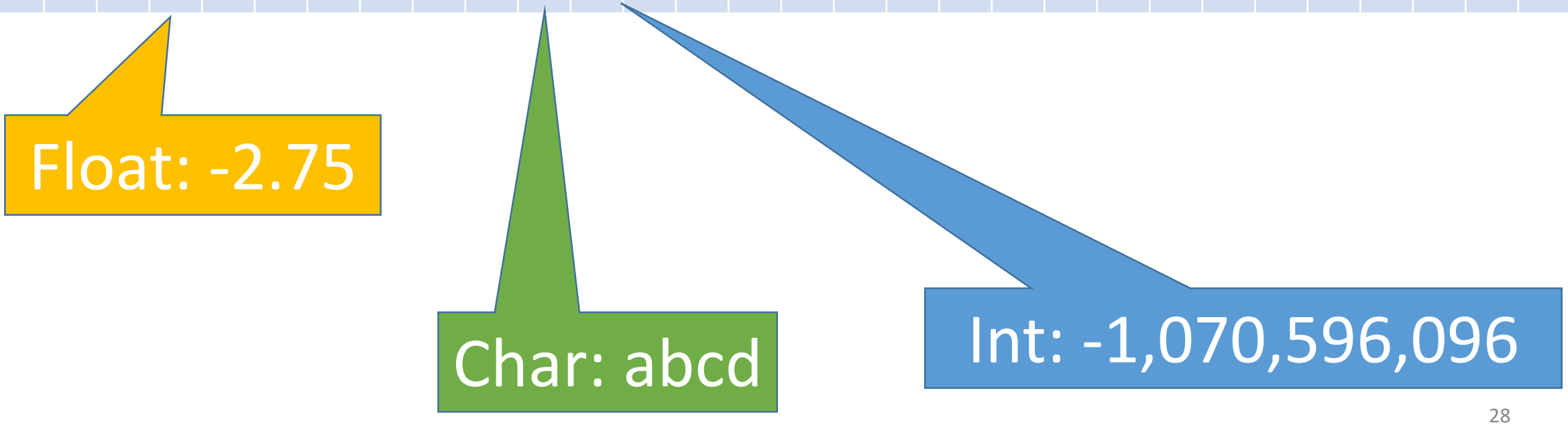
- Space to hold a single letter (e.g. ‘a’, ‘Q’, ‘$’, ‘_’, ‘ ‘)
- Internally 8 bit number between 0 and 256
  - ASCII mapping from letter to number and back
  - See http://www.asciitable.com/
  - Some numbers do not map to a letter – “unprintable”
- In numeric contexts (e.g. arithmetic operations) C treats char as a number

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Type: How to interpret bits in memory

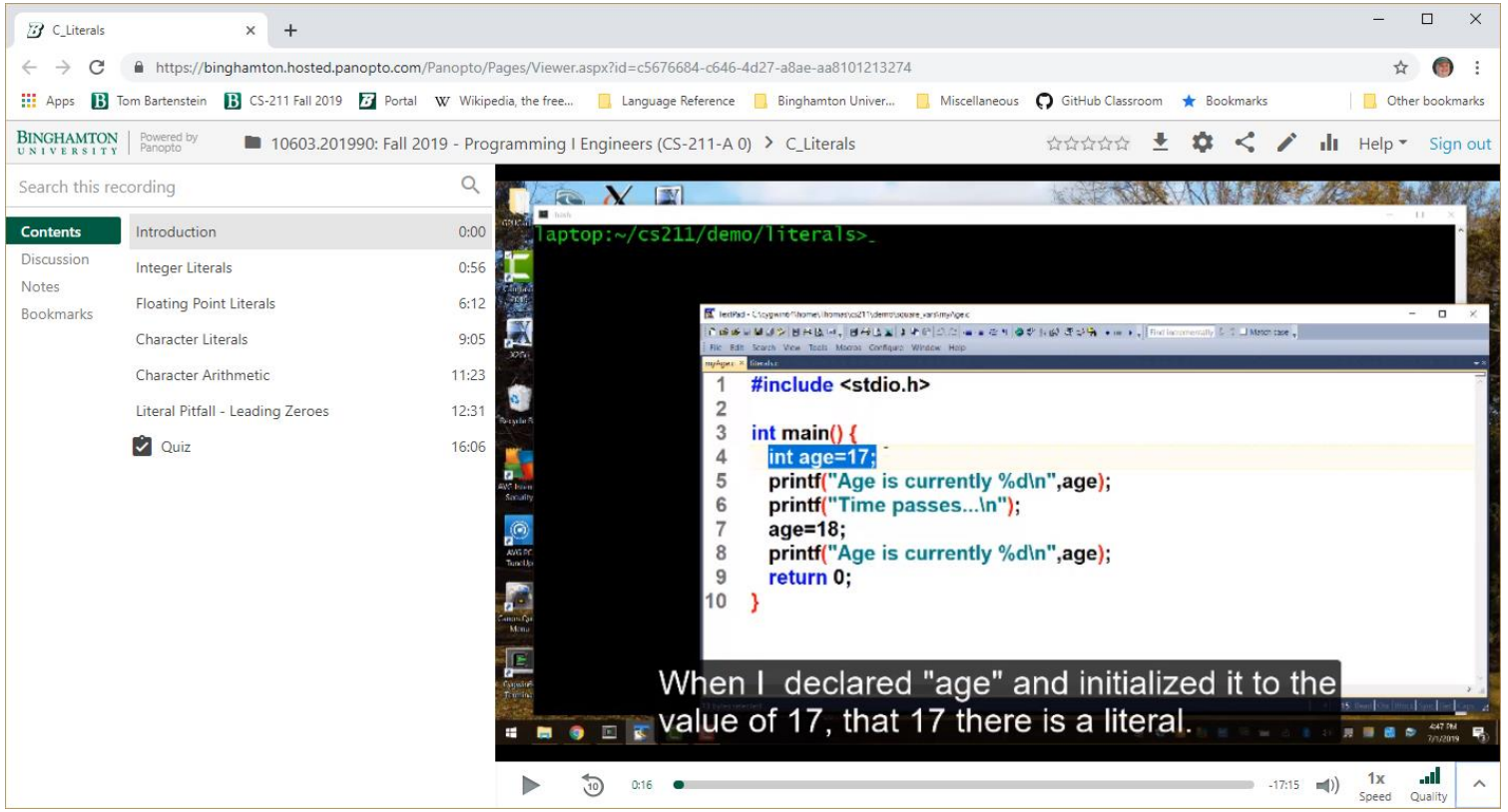
- All computer data consists of strings of 0's and 1's
- How does the computer know what the string means?

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



C Literals

Summary Notes



Compiler Literals

Compiler interprets some program text as literal values…

- Numbers without decimal points are integers: 10, 38221, 41_385
- Integers with leading zeroes are base 8: 0723, -0435, 0943
- Integers can be specified base 2 or base 16: 0b0011_1010, 0x5eC0
- Numbers with decimal points are floats: 1.0, 3.1415
- You may use scientific notation: 6.22e23, 1.1413e-12
- Single letters in single quotes are ASCII char : ‘a’, ‘Q’, ‘$’, ‘ ‘
- Letters in double quotes are ARRAYS of chars: “Hello World!”

Pitfall: Leading Zeroes

```
month[00] = “Jan”;
month[01] = “Feb”;
month[03] = “Mar”;
…
month[07] = “Aug”;
month[08] = “Sep”;
month[09] = “Oct”;
month[10] = “Nov”;
month[11] = “Dec”;
```

Compiler error: Invalid number