CSCI 200: Foundational Programming Concepts & Design Lecture 02

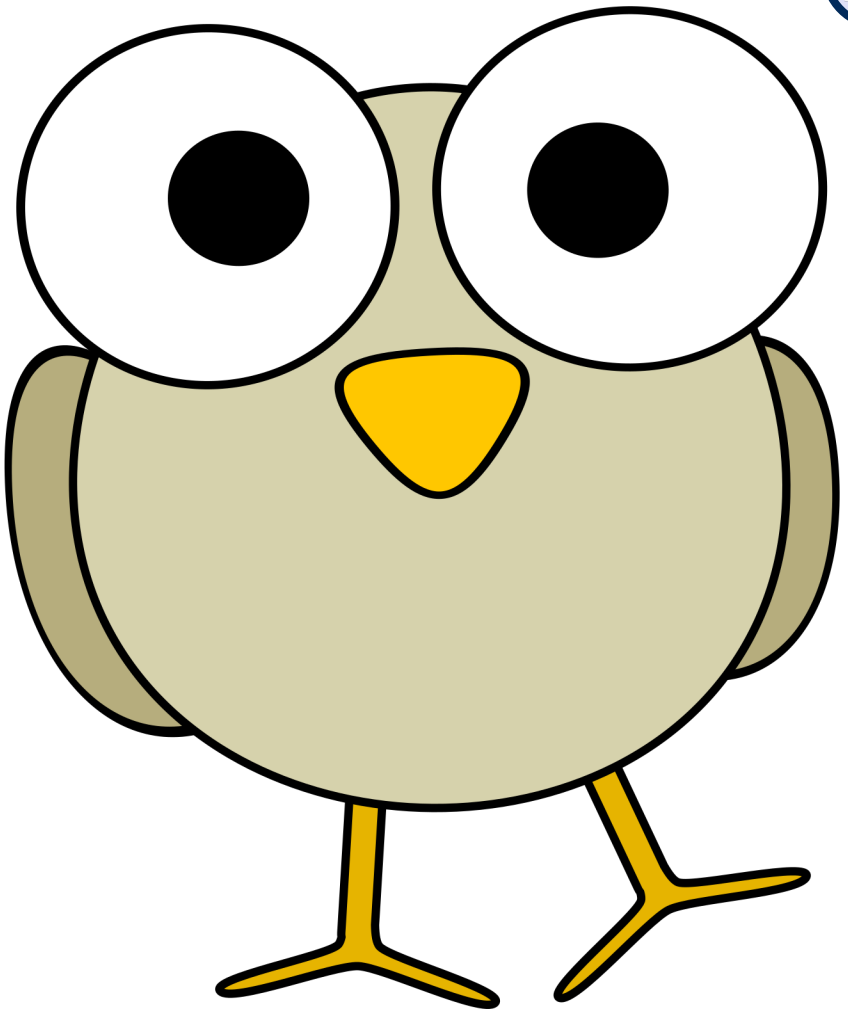
Data in Memory

Follow along with handout
linked on schedule page

Previously in CSCI200

- To display information with a program we use output: This is done with the [ ] keyword
- To build a program from a cpp file, we use the [ ] program.

Questions?



Learning Outcomes For Today

- Create a Hello World program, construct a simple interactive application, and build the program via the terminal.
- List C++ primitive data types and explain the appropriate use of each data type.
- List & identify C++ arithmetic operators, translate math equations to C++, and solve arithmetic expressions.
- Explain how values are stored in memory, how the values are interpreted differently based on data type, and list common errors that can occur with data types.
- Discuss the effects of a statically typed language.
- Diagram how integer and decimal values are represented in binary.
- Convert between binary and decimal formats.
- Convert one data type to another.
- Recite the order of operations and evaluate an expression.

On Tap For Today

- Primitive Data Types
- Memory
  - Variables & Input
  - Constants & Modifiers
- Random Values
- Practice

On Tap For Today

- Primitive Data Types
- Memory
  - Variables & Input
  - Constants & Modifiers
- Random Values
- Practice

Data Types

- **int** `-3  0  1`
  - integers aka whole number
- **float** / **double** `-3.92f  0.44f / 2.718  3.141`
  - floating point numbers aka decimal numbers
- **char** `'a'  't'  '6'`
  - a single character: any letter or number
- **bool** **true** **1** **false** **0**
  - true or false

Arithmetic Operators

- Just like algebra

  +, -, *, /, %

  - 16 % 3 → 
  - 16 / 3 → 

- Follows precedence rules

Precedence Table

Precedence	Operator	Associativity
1	Parenthesis: ()	Innermost First
2	Binary Operators: a*b a/b a%b	Left to Right
3	Binary Operators: a+b a-b	Left to Right

Practice!

- 7 + 3 * 5 – 2 **20**
- 4 + 11 / 3 **7**
- 8 % 3 * 6 **12**
- (7 + 3) * 5 - 2 **48**

More Powerful Math

- Include the standard math library

```
#include <cmath>
```

- Gives you access to math functions
  - sqrt(x)
  - pow(x, y)
  - log(x)
  - sin(x)
  - And many more!

Longer (Not Complete) List of Math Functions

Function	Description		Function	Description
pow	Raise to power		***cos***	Cosine
sqrt	Square root		***sin***	Sine
			tan	Tangent
exp	Exponential function		***acos***	Arc cosine
log	Natural logarithm		***asin***	Arc sine
log10	Common logarithm		***atan***	Arc tangent
			atan2	Arc tangent with two parameters
ceil	Round up value		***cosh***	Hyperbolic cosine
fabs	Compute absolute value		***sinh***	Hyperbolic sine
floor	Round down value		***tanh***	Hyperbolic tangent
fmod	Remainder of division			
abs	Compute absolute value		***frexp***	Get significand and exponent
			ldexp	Generate number from significand and exponent

On Tap For Today

- Primitive Data Types
- Memory
  - Variables & Input
  - Constants & Modifiers
- Random Values
- Practice

The Computer

Inside the Computer

Inside the Computer

Abstraction

- "Don't care what's inside as long as it works"
  - The computer
  - Math functions

- But somewhere in there is a big block of memory to store values

On Tap For Today

- Primitive Data Types
- Memory
  - Variables & Input
  - Constants & Modifiers
- Random Values
- Practice

Variables

- Need to declare facts about the world to the computer

- Facts that **change** over time are called **variables**

Variables & Memory

- Identifier points to memory address where value is stored

- When you reference a variable, computer looks up in memory the value at the corresponding address

Declaring a Variable

- Recall, computers are dumb
- Variables need a **data type**
  - Why? Computer needs to know how much memory it needs ahead of time
    - Why is memory needed? To store a value
- Variables need an **identifier**
  - Why? We need to know how to access the value in memory

Identifiers

- Names pointing to a value (like algebra)

- Should describe the value
  - **temperature** instead of t
  - Why?
    - Everyone knows what that variable is storing
      - Time? Temperature? Tornados?
    - Try searching for the letter t to find something

C++ is Case-Sensitive

- `int Porsche;`
- `int porsche;`
- `int pOrScHe;`

- These are THREE different variables
- Need a convention, or **style**, to be consistent

Identifier Style: lower camelCase

- `double avgWindSpeed;`
- `bool isAfricanSwallow;`

- Starts w/ lower case
- Every new "word" starts with upper case

- **REQUIRED** for this course
  - Will add to our style list as course goes on

Notes on Identifiers

- Cannot
  - Contain special characters ($%^&@#! and similar)
  - Begin with a number
  - Be a reserved word

Table 1.10.1: C++ reserved words / keywords.

alignas (since C++11)	enum	return
alignof (since C++11)	explicit	short
and	export	signed
and_eq	extern	sizeof
asm	false	static
auto(changed C++11)	float	static_assert(since C++11)
bitand	for	static_cast
bitor	friend	struct
bool	goto	switch
break	if	template
case	inline	this
catch	int	thread_local(since C++11)
char	long	throw
char16_t(since C++11)	mutable	true
char32_t(since C++11)	namespace	try
class	new	typedef
compl	noexcept(since C++11)	typeid
const	not	typename
constexpr(since C++11)	not_eq	union
const_cast	nullptr (since C++11)	unsigned
continue	operator	using(changed C++11)
decltype(since C++11)	or	virtual
default(changed C++11)	or_eq	void
delete(changed C++11)	private	volatile
do	protected	wchar_t
double	public	while
dynamic_cast	register	xor
else	reinterpret_cast	xor_eq

Static Declarations

- Need to declare data type up front so computer can allocate enough memory
- Data types take different amount of memory

Data Type	Size	Range	
`bool`	8 bits / 1 byte*	0 to 1	0 to 1
`char`	8 bits / 1 byte	-2^7 to $+2^7-1$	-128 to +127
`int`	32 bits / 4 bytes	-2^{31} to $+2^{31}-1$	-2,147,483,648 to +2,147,483,647
`float`	32 bits / 4 bytes	±1.18e-38 to ±3.4e38	~7 digits precision
`double`	64 bits / 8 bytes	±2.23e-308 to ±1.80e308	~16 digits precision

- *theoretically 1 bit, but in practice memory access is done by byte

integers

- N-bits can store

  -2^{N-1} to $+2^{N-1}-1$

- Why?

  1 bit for sign (positive/negative)

  $N-1$ bits for value

- How to store value?

  Positive: as normal

  Negative: Two's Complement

Convert to Two's Complement

- Convert absolute value decimal to binary
- Invert bits
- Add one

- 7 =
- -6 =
- Why?
  - Math!

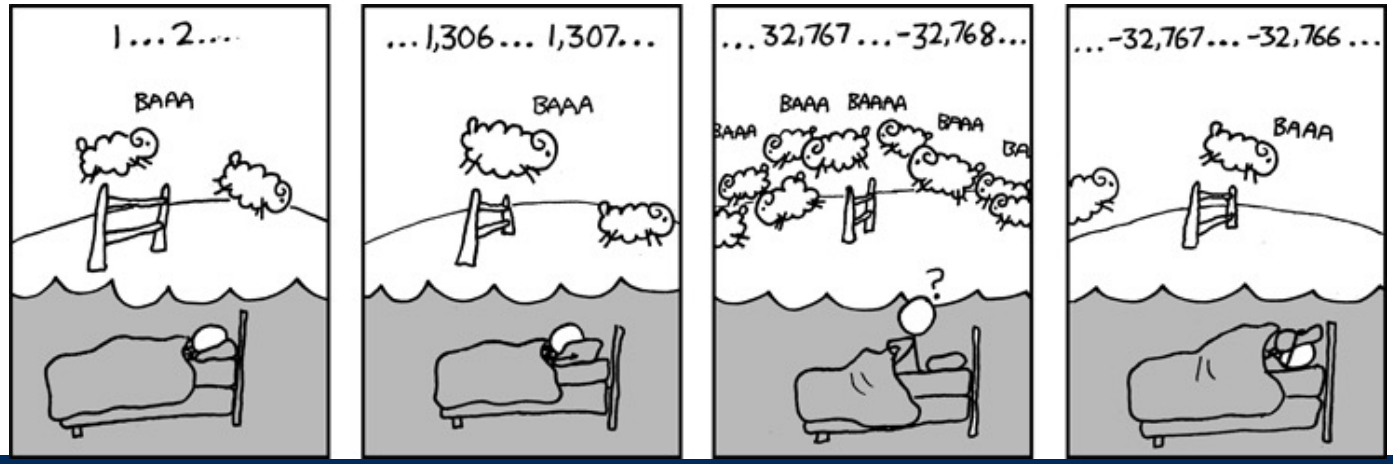
Two's Complement Math

- Addition / Subtraction are the same

  7 – 6 = 1 === 7 + (-6) = 1

  1 – 6 = -5 === 1 + (-6) = (-5)

- Concern
  - Overflow / Underflow



float / double

- Single-precision / Double-precision
  - 1 bit for sign
  - *e* bits for exponent (8 / 10)
  - *m* bits for mantissa (23 / 53)

- 12.375_{10} =

Assigning A Value To A Variable

- Use = to assign a value to a variable
  - Assignment (=) is not equality (==)
- General form
  - `identifier = expression;`
- Examples
  - `double sum = 0.0;`
  - `int length;`
    `length = 12;`
  - `int x = 5;`
    `x = x + 1;`
  - `int x, y = 4;`
    `x = y;`

Precedence Table

Precedence	Operator	Associativity
1	Parenthesis: ()	Innermost First
2	Binary Operators: a*b a/b a%b	Left to Right
3	Binary Operators: a+b a-b	Left to Right
4	Assignment Operators: a=b	Right to Left

Memory Example

```
int numCars = 5;
double temp = 37.1;
char mcAns;
mcAns = 'd';
```

Address	Identifier	Value
0xf3da8000		
0xf3da8004		
0xf3da8008		
0xf3da800c		
0xf3da8010		
0xf3da8014		

ASCII Table

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

Memory Example

```
int numCars = 5;
double temp = 37.1;
char mcAns;
mcAns = 'd';
mcAns += 1;
//mcAns now is 'e'
```

Address	Identifier	Value
0xf3da8000	mcAns	**0x0065**
0xf3da8004	numCars	0x00000005
0xf3da8008		
0xf3da800c	temp	0x40428cc cccccccd
0xf3da8010		
0xf3da8014		

Why Types Matter I

- How are values stored in memory?
  - In binary!
- Data type merely states how to interpret binary value
- Statically typed – variables will always refer to the same data type for the life of the program
  - Will always interpret memory in the same manner
  - Values & operations checked at compile time to prevent *some* runtime errors

Why Types Matter II

```
int x;


x = 2;                  // x is
x = 3.0;                // x is
x = 4.8;                // x is
x = 9 / 5;              // x is
x = 9.0 / 5.0;          // x is
```

Why Types Matter II

```
double x;


x = 2;                  // x is
x = 3.0;                // x is
x = 4.8;                // x is
x = 9 / 5;              // x is
x = 9.0 / 5.0;          // x is
```

What's the difference?

```
float x;

x = 9 / 5;
```

x = ☐

```
float x;

x = 9.0f / 5.0f;
```

x = ☐

Order of Operations:

Calculations happen before assignment

Types Matter

- 9 / 5 = 1
  - int / int = int
- 9.0 / 5.0 = 1.8
  - double / double = double
- 9.0 / 5 = 1.8
  - double / int = double
  - (int gets temporarily promoted to a double)

Type Casting

```
int x = 9, y = 5;
double z;

// style 1
z = (double) x / y;                    // z = 1.8

// style 2
z = static_cast<double>(x)/ y;   // z = 1.8
```

Why need to type cast?

- Consider

```
int numberSections = 6, totalStudents = 363;
double studentsPerSection = totalStudents / numberSections;
double sectionAverage = totalStudents / (double)numberSections;

cout << studentsPerSection << endl;
cout << sectionAverage << endl;
```

- What is the output?

```
60
60.5
```

Precedence Table

Precedence	Operator	Associativity
1	Parenthesis: ()	Innermost First
2	Unary Operators: (type)a	Right to Left
3	Binary Operators: a*b a/b a%b	Left to Right
4	Binary Operators: a+b a-b	Left to Right
5	Assignment Operators: a=b	Right to Left

More Assignment Operators

- Compound Operators

Operator	Example	Equivalent Statement
+=	x += y;	x = x + y;
-=	x -= y;	x = x – y;
*=	x *= y;	x = x * y;
/=	x /= y;	x = x / y;
%=	x %= y;	x = x % y;

Negation

- Can either multiply by -1

```
int x, y(3);

x = -1 * y;
```

- Or simply negate variable

```
int x, y(3);

x = -y;
```

Precedence Table

Precedence	Operator	Associativity
1	Parenthesis: ()	Innermost First
2	Unary Operators: +a -a (type)a	Right to Left
3	Binary Operators: a*b a/b a%b	Left to Right
4	Binary Operators: a+b a-b	Left to Right
5	Assignment Operators: a=b a+=b a-=b a*=b a/=b a%=b	Right to Left

Other “tricks”

- Multi-assignment

```
int x, y;

x = y = 5;
```

```
y = 5; // y is assigned 5
```

- Which means, this is valid

```
a = b += c - d;
```

cin

- Character INput
  - or standard input from the screen

```
int myAge;


cin >> myAge;
```

  - "Computer, take whatever the user types when they hit [enter] and assign it to the variable following the >> symbol."
  - What should precede cin?

Program Input

```
#include <iostream>

using namespace std;

int main() {

    int myAge;

    cout << "What's my age again? ";

    cin >> myAge;

    cout << "That's right, my age is "
         << myAge << endl;

    return 0;

}
```

On Tap For Today

- Primitive Data Types
- Memory
  - Variables & Input
  - Constants & Modifiers
- Random Values
- Practice

Constants

- What is a fact that never changes?
  - A constant!

```
const int DAYS_IN_WEEK = 7;
```

- A variable whose value cannot change
- Style: ALL_CAPS_WITH_UNDERSCORES
  - REQUIRED for this course

CSCI 200 Style Guidelines

- variables follow lowerCamelCase
- CONSTANTS follow UPPER_SNAKE_CASE

Additional Modifiers

- `short int`
- `long int`
- `long long int`
  - Uses less or more memory

Data Type	Size	Range	
`short int`	16 bits / 2 bytes	-2^{15} to $+2^{15}-1$	-32,678 to +32,677
`int`	32 bits / 4 bytes	-2^{31} to $+2^{31}-1$	-2,147,483,648 to +2,147,483,647
`long int`	32 bits / 4 bytes	-2^{31} to $+2^{31}-1$	-2,147,483,648 to +2,147,483,647
`long long int`	64 bits / 8 bytes	-2^{63} to $+2^{63}-1$	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807

Additional Modifiers

- **unsigned**
  - Most significant bit part of value, not the sign

Data Type	Size	Range	
`signed short`	16 bits / 2 bytes	-2^{15} to $+2^{15}-1$	-32,678 to +32,677
`unsigned short`	16 bits / 2 bytes	0 to $2^{16}-1$	0 to 65,535
`signed int`	32 bits / 4 bytes	-2^{31} to $+2^{31}-1$	-2,147,483,648 to +2,147,483,647
`unsigned int`	32 bits / 4 bytes	0 to $+2^{32}-1$	0 to +4,294,967,295
`signed long long`	64 bits / 8 bytes	-2^{63} to $+2^{63}-1$	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
`unsigned long long`	64 bits / 8 bytes	0 to $2^{64}-1$	0 to +18,446,744,073,709,551,615

Signed / Unsigned

- What prints?

```
signed int x = -1;
unsigned int y = -1;
cout << x << endl;
cout << y << endl;
```

Declaring Variables

- [anything in here is optional]

```
[modifier]      dataType      identifier      [= initialValue]          ;
[modifier]      dataType      identifier      [(initialValue)]          ;
```

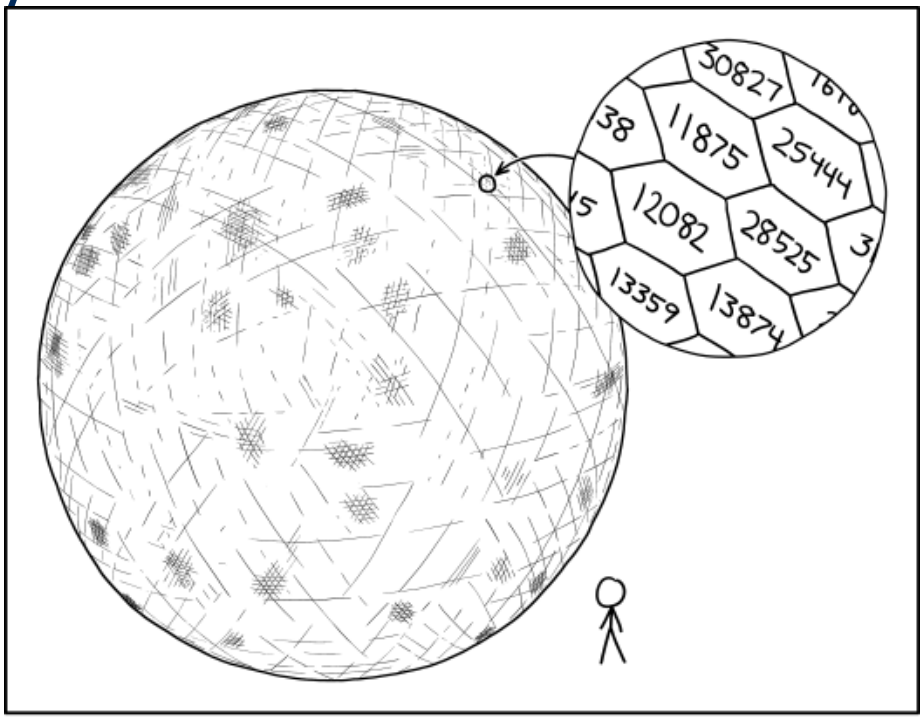
```
                int           myAge                                     ;
unsigned        int           currMonth       = 7                       ;
const           double        PI              = 3.14159                 ;
                char          firstInitial = 'J', secondInitial('P')    ;
```

On Tap For Today

- Primitive Data Types
- Memory
  - Variables & Input
  - Constants & Modifiers
- Random Values
- Practice

Random Numbers

- For the computer
  - Cannot generate purely random numbers
- Pseudo-Random
  - Kinda random
  - Good enough for us!



THE HARDEST PART OF SECURELY GENERATING RANDOM 16-BIT NUMBERS IS ROLLING THE D65536.

https://xkcd.com/2626/

Pseuo-Random Numbers

- Need to include the standard C library

```
#include <cstdlib>
```

- Simply call rand() and get a random number!
  - But…

Seed the RNG

- Random Number Generator (RNG)

- Initialize the sequence of generated random numbers using srand()
  - Use the same seed?
  - Get the same random sequence
  - Default seed: 1

Pseuo-Random Process

1. Include **`cstdlib`**
2. Set the random seed with **`srand()`**
3. Call **`rand()`** as needed

Example

```
#include <cstdlib>
#include <iostream>
using namespace std;


int main() {
    int seed;
    cout << "Enter the seed: ";
    cin >> seed;
    srand( seed );                  // rand is now seeded
    cout << rand() << endl;         // print a random number
    cout << rand() << endl;         // print a random number
    cout << rand() << endl;         // print a random number
    return 0;
}
```

Better Example

```
#include <cstdlib>
#include <ctime>
#include <iostream>
using namespace std;

int main() {
    srand( time(0) );                    // rand is now seeded
                                         // with the current time
    cout << rand() << endl;              // print a random number
    cout << rand() << endl;              // print a random number
    cout << rand() << endl;              // print a random number
    return 0;
}
```

Best Example

```
#include <cstdlib>
#include <ctime>
#include <iostream>
using namespace std;

int main() {
    srand( time(0) );                    // rand is now seeded
                                         // with the current time
    rand();                              // throw away first value

    cout << rand() << endl;              // print a random number
    cout << rand() << endl;              // print a random number
    cout << rand() << endl;              // print a random number
    return 0;
}
```

Bestest Example

```
#include <cstdlib>
#include <ctime>
#include <iostream>
using namespace std;

int main() {
    srand( time(0) );                       // rand is now seeded
                                            // with the current time
    rand();                                 // throw away first value

    cout << rand() % 50 + 5 << endl;        // print a random number between ?
    cout << rand() % 50 + 5 << endl;        // print a random number between ?
    cout << rand() % 50 + 5 << endl;        // print a random number between ?
    return 0;
}
```

Practice

- What is min value rand() will generate?

  0

- What is max value rand() will generate?

  `RAND_MAX`

Practice

- How to generate a random integer between 2 & 10 inclusive?

```
rand() % (10 - 2 + 1) + 2
rand() % (max - min + 1) + min
```

- How to generate a random float between 2 & 10 inclusive?

```
rand() / (double)RAND_MAX * (10.0 - 2.0) + 2.0
rand() / (double)RAND_MAX * (max - min) + min
```

On Tap For Today

- Primitive Data Types
- Memory
  - Variables & Input
  - Constants & Modifiers
- Random Values
- Practice

To Do For Next Time

- Answer review questions in 8/25 Post Class Survey
- Lab1A write up is on the course website
  - Choose two equations
  - Define constants
  - User inputs unknown variables
  - Solve for result
- Next time:
  - Command Line Interface
  - Makefiles: compiling our programs