CSCI 200: Foundational Programming Concepts & Design Lecture 02

Data in Memory

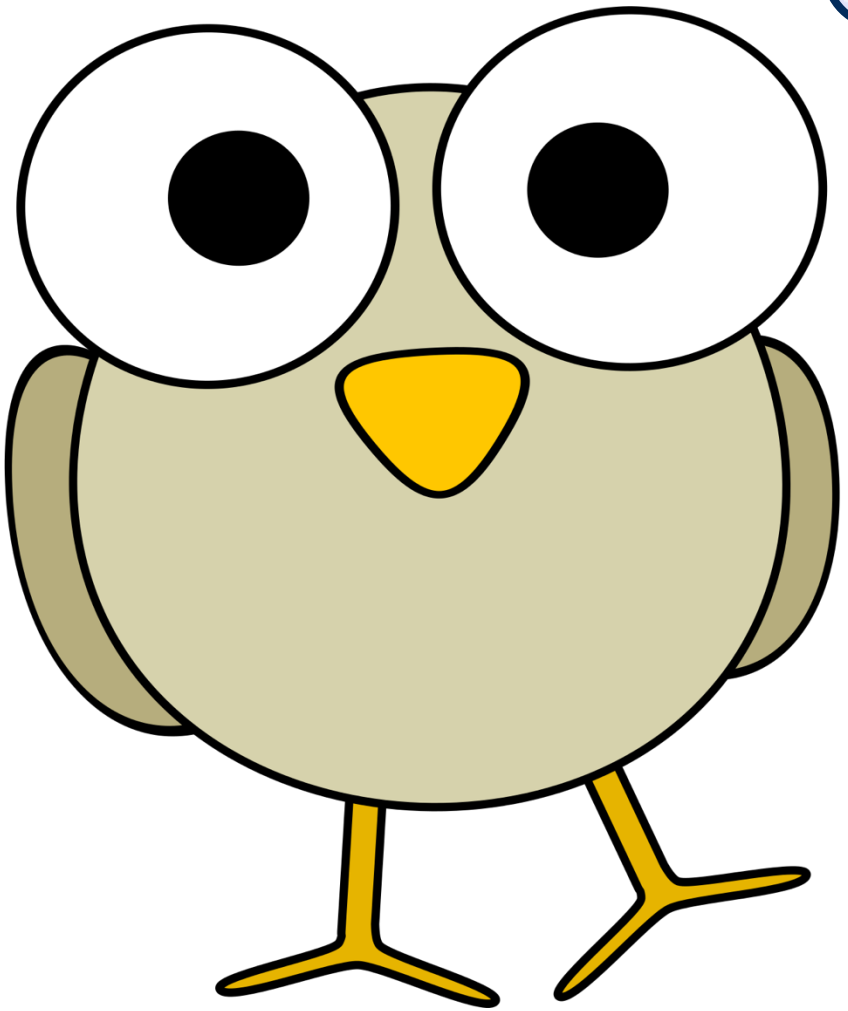
Follow along with handout linked on schedule page

Always have VS Code and iClicker open

Previously in CSCI200

- To display information with a program we use output: This is done wi`cout` keyword
- uild a program from a cpp file, we use the `g++` program.

Questions?



Turing Machine

- Given infinite time and memory, if a machine has the following features:
  1. Sequence
  2.
  3.
  4. Output
  5.
  6.
- It can solve any mathematical problem

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

- Memory & Variables
- Practice

The Computer

Inside the Computer

Inside the Computer

*** Abstraction ***

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

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

Data Types

- int `-7  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

Practice!

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

Precedence Table

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

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

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

Integer Size 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 $+2^{15}-1$	-32,768 to +32,767
`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

Integer Size Modifiers

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

Data Type	Size	Range	
`short`	16 bits / 2 bytes	-2^{15} to $+2^{15}-1$	-32,768 to +32,767
`int`	32 bits / 4 bytes	-2^{31} to $+2^{31}-1$	-2,147,483,648 to +2,147,483,647
`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

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

Practice!

- Using 8 bits, what is the binary representation of the decimal value 7?

- Using 8 bits, what is the binary representation of the decimal value 6?

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)

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 $+2^{15}-1$	-32,768 to +32,767
`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?

```
01 signed int x = -1;
02 unsigned int y = -1;
03 cout << x << endl;
04 cout << y << endl;
```

Math Concern

- What is the output?

```
01 short everest = 29032;
02 short elbert = 14439;
03 short total = everest + elbert;
04 cout << total << endl;
```

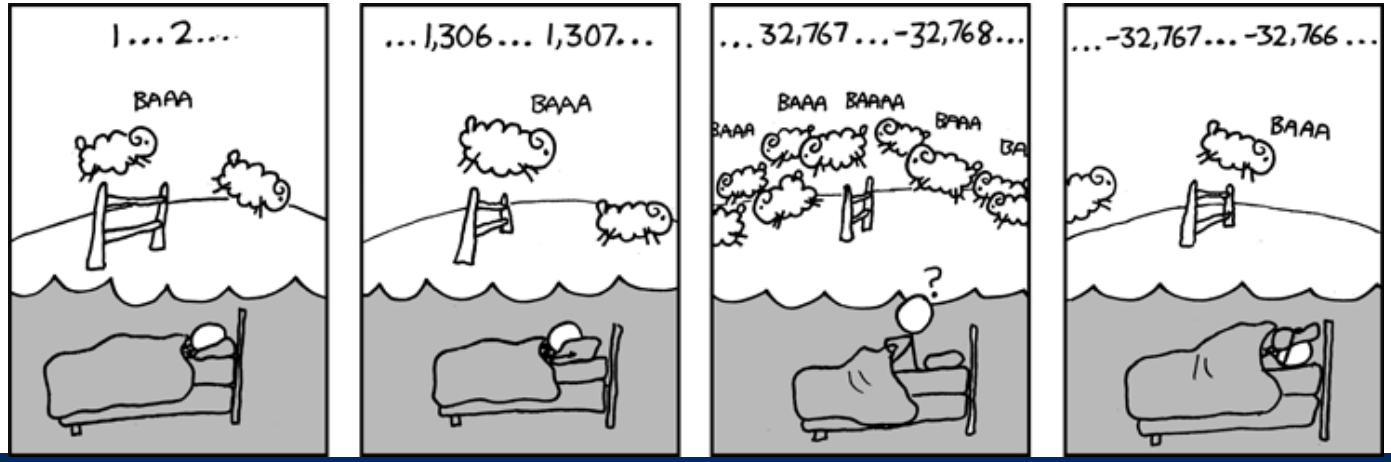
Data Type	Size	Range	
short	16 bits / 2 bytes	-2^{15} to $+2^{15}-1$	-32,768 to +32,767

Math Concern

- What is the output?

```
01 short everest = 29032;
02 short elbert = 14439;
03 short total = everest + elbert;   // == 43471
04 cout << total << endl;            // prints -22065 !
```

- 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} =

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

Memory Example

```
01 int numCars = 5;
02 double temp = 37.1;
03 char mcAns;
04 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

```
01 int numCars = 5;
02 double temp = 37.1;
03 char mcAns;
04 mcAns = 'd';
05 mcAns += 1;
06 // mcAns is now 'e'
```

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

Why Types Matter II

- 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`)

Why need to type cast?

- Consider

```
01 int numberSections = 3, totalStudents = 220;
02 double studentsPerSection = totalStudents / numberSections;
03 double sectionAverage = totalStudents / (double)numberSections;
04 cout << studentsPerSection << " " << sectionAverage;
```

- What is the output?

```
73 73.3333
```

Precedence Table

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

Turing Machine

- Given infinite time and memory, if a machine has the following features:
  1. Sequence
  2.
  3.
  4. Output
  5. Input
  6. Variables
- It can solve any mathematical problem

On Tap For Today

- Primitive Data Types
- Memory & Variables
- Practice

To Do For Next Time

- Read & watch Random Number Generation
- Lab1A write up is on the course website
  - Get started!
- Next time:
  - Random Numbers
  - Command Line Interface
  - Makefiles