Introduction to CS #1

By Sadek Mohammed

1. Lecture 1 "30 minutes"

CS consists of many branches, where the Hack Club discusses:

- Mobile Apps
- Games
- Cybersecurity
- Python

Although CS may look broad, you ought to learn some basics.

This intro gives you the basics, where we will use the c++ programming language.

To start coding c++, use the codeblocks editor

There is a template code you find in the editor once you click **create new project**.

The template code is as follows

```
#include <iostream>

using namespace std;

int main()
{
    // Your Code Goes here
}
```

Neglect those statements, we will explain them later.

To output a statement add the following statement between the curly bracket.

```
cout << "Hello World" << endl; // Output is: Hello World
```

You shall see the words "Hello World" in the console.

It is notable to say that the phrase after the two back slashes are called "comments", and they are not compiled by the compiler.

Try changing the statement to

```
cout << "Hello World" << "\n"; // Output is: Hello World
```

Do you feel any difference? Does that mean they are equivalent?

Yes, they are equal; **however,** the difference is with efficiency. endl is less efficient than \n.

1.1. Basic Data Types

boolean: means a true or false, and it can be represented as 0 or 1. It occupies one byte of memory.

int: means an integer and is any number that lies between -2^{31} and $2^{31} - 1$. It normally occupies about 4 bytes of memory but differs according to the processor and architecture.

unsigned int: means a positive integer and is any number that lies between 0 and $2^{32} - 1$. It normally occupies about 4 bytes of memory but differs according to the processor and architecture.

short: means a short integer and is any number that lies between -2^{16} and $2^{16} - 1$. It normally occupies about 2 bytes of memory but differs according to the processor and architecture.

long long: means a long integer and is any number that lies between -2^{63} and $2^{63} - 1$. It normally occupies about 8 bytes of memory but differs according to the processor and architecture.

float: means a short decimal and stores about 7 decimal digits, occupying about 4 bytes of memory.

double: means a long decimal and stores about 15 decimal digits, occupying about 8 bytes of memory.

char: means a letter, and it occupies a single byte in the memory. See the ASCII chart here.

0	NUL	16	DLE	32	SP	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

```
cout << (char)73 << endl; // The output is the letter "I"
```

Notice the use of the spreading operator to change from integer to char which will be discussed in the next lecture.

1.2. Proof of the limits of the DataTypes

The reason of the 2^{31} limit of the integer is due to the nature of the memory of the computer. Imagine the 4 bytes limit, each Byte contains **8 bits**, and each bit is a transistor. That transistor has an "on" signal and "off" one. The on is represented as 1; on the other hand, off is represented as 0. Since each bit could be any of two values, and according to simple combinatorics, that implies 2^n probabilities. That directly translates to the 2^{32} limit of the unsigned integer.

Since the integer contains negative values, too, we have to store an equal number of positive and negative values, so the number of values should be divided by 2. This is represented by the following formula.

$$\frac{2^{32}}{2^1} = 2^{32-1} = 2^{31} \ (property\ of\ division\ of\ common\ bases)$$

The reason of the recurrent -1 is the consideration of 0 as a positive number. (i.e. If you have space for 16 number, you, basically, have spaces for numbers from 0-15 inclusive)

1.3. Operations on datatypes

1.3.1. Addition

Datatypes could be added like the ordinary mathematics sign ($+$). However, if you added two integers that exceed the limit (i.e. imagine adding $2^{31} + 2^{31}$) that causes overflow in the value of the datatype. That introduces as to the importance of **casting**.

1.3.2. Subtraction

Datatypes could be subtracted, too, using the regular subtraction sign ($-$). Remember that all datatypes (except booleans, chars, and unsigned variables) have negative values.

1.3.3. Multiplication

Datatypes could be multiplied, too, using the aestrix $(*)$. Remember that multiplication reach the maximum value quickly.

1.3.4. Division

Datatypes could be be divided, too, using the slash $(/)$. Remember to take care of decimals through casting.

1.3.5. Modulus

To check whether a number is divisible by another number. Modulus gives the remainder using a . If a number is divided by another number, it gives a remainder (modulus) of 0.