ENGR101 Assignment 3

Name: Niels Clayton Student ID:300437590

Core 1:
1011
8+0+2+1
11

Core 2:
13
8+4+0+1
1101

Core 3:
1011 > Bit-shifted right > 0101

0101
0+4+0+1
5

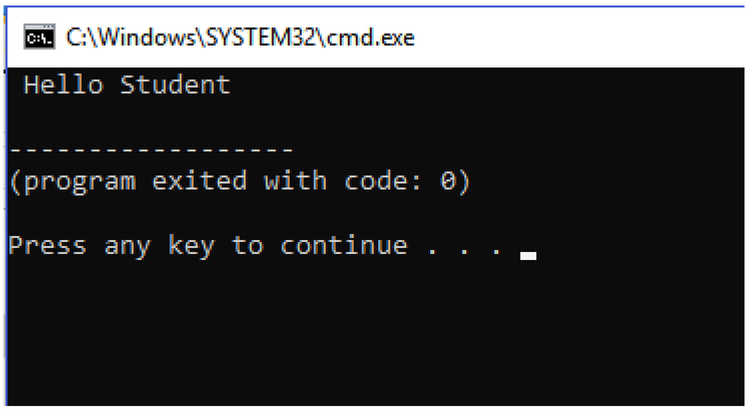
Core 4:
1101 > Bit-shifted left > 1010

1010
8+0+2+0
10

Core 5:

Compiling converts program text e.g. (UI.println ("Hello world!"); into object code that the computer can execute like binary.

Core 6:



```
C:\Windows\SYSTEM32\cmd.exe
 Hello Student

-------------------
(program exited with code: 0)

Press any key to continue . . .
```

Core 7:

There is no file extension equivalent of an executable (.exe) in linux

Completion 1:
Prediction:
This code will not compile, this is because you have not specified what type of variable ‘number’ is. In order to correct this, you put with in the brackets that you are printing an integer “%i”, it will then take the variable called ‘number’, call it and print the decimal 120.

Result:
error compiling

Explanation:
This code did not compile because we have not specified what type of variable ‘number’ is.

Corrected Code:
```
# include <stdio.h>
int main (){
int number = 120;
printf ("%d", number );
return 0;
}
```

Core 8:
```
# include <stdio.h>

int main (){
	int num = 1000;
	char character = 2;
	char unassignedchar;
	double doublenu = 99;
	long longnu = 999;

	int intsize = sizeof num;
	int charsize = sizeof character;
	int unassignedsize = sizeof unassignedchar;
	int doublenusize = sizeof doublenu;
	int longnusize = sizeof longnu;

	printf("%d\n", intsize);
	printf("%d\n", charsize);
	printf("%d\n", unassignedsize);
	printf("%d\n", doublenusize);
	printf("%d\n", longnusize);
}
```

Variable Type	Number of bytes used
int	4
char	1
long	4
double	8

<u>Core 9:</u>
Unsigned char is the same size in bytes as char, and since one byte has 8 bits, its maximum value will be 2^8-1=255

<u>Challenge 1:</u>

Unsigned char uses 1 byte or 8 bits. Because of this the largest possible number that can be stored is 255, as this written in binary is ’11111111’. If we were to add 1 to 255 in binary this would be written;

```
  11111111
+        1
100000000
```

But since only 8 bits can be stored by the type unsigned char, this number will be written down as ‘00000000’ which equals 0.
The program then prints both of these numbers onto the same line as there is no ‘\n’ which leads to the number 2550 being printed.

<u>Completion 2:</u>

Type char is 1 bit, and it is unassigned meaning that it is half positive and half negative (-128-0-127), because of this as soon as you add 1 onto 127 it jumps down to the negative value 128.

<u>Core 10:</u>
A= 00111100
B= 00001101

Core 11:

A	B	A AND B
0	0	0
0	0	0
1	0	0
1	0	0
1	1	1
1	1	1
0	0	0
0	1	0

A AND B = 00001100 = 8+4 = 12

Core 12:

A	B	A OR B
0	0	0
0	0	0
1	0	1
1	0	1
1	1	1
1	1	1
0	0	0
0	1	1

A OR B = 00111101 = 32+16+8+4+1 = 61

Core 13:

A	B	A XOR B
0	0	0
0	0	0
1	0	1
1	0	1
1	1	0
1	1	0
0	0	0
0	1	1

A XOR B = 00111101 = 32+16+1 = 49

Completion 3:

```
# include <stdio.h>
int main (){
	unsigned int A =60;
	unsigned int B =13;

	printf("%d\n", A & B);
	printf("%d\n", A | B);
	printf("%d\n", A ^ B);

}
```

Challenge 2:

<your answer here>