Bitwise complement or NOT in C++

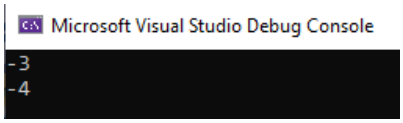
Signed numbers generally follow a 2's complement representation (invert bits and add 1). Thus, when you apply negation, C++ finds 2s complement of a number and return the answer. For now, if you don't know about 2s complement and logic behind it that why we use 2's complement for dealing with signed integers (+ve and -ve) then you can simply ignore. You will hopefully cover that in DLD course. Currently, just note the following rule and don't confuse yourself. I will not give any question in exam/ quiz that asks you to dry run and provide detailed working of bitwise AND,OR,NOR operations and their resultant values.

For any integer n, bitwise complement of n will be -(n+1).

Consider the code:

```
#include <iostream>
using namespace std;
int main()
{
	int a=2,b=3;
	int result_not_a = ~(a);
	int result_not_b = ~(b);
	cout << result_not_a << endl;
	cout << result_not_b << endl;
	return 0;
}
```

Output:



How value of ~a is found?

	…	64	32	16	8	4	2	1
a	0	0	0	0	0	0	1	0

	…	64	32	16	8	4	2	1
b	0	0	0	0	0	0	1	1

After applying NOT

	…	64	32	16	8	4	2	1	In Decimal
a	0	0	0	0	0	0	1	0	2
	1	1	1	1	1	1	0	1	253

After 2s complement of a result (253):

…	64	32	16	8	4	2	1		In decimal
1	1	1	1	1	1	0	1		
0	0	0	0	0	0	1	0	***Invert***	***2s***
						+	1	***Add 1***	***complement***
0	0	0	0	0	0	1	1		3

Thus, we get an answer -3 for ~a.