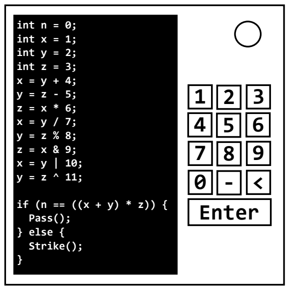
On the Subject of Lines of Code



```
int n = 0;
int x = 1;
int y = 2;
int z = 3;
x = y + 4;
y = z - 5;
z = x * 6;
x = y / 7;
y = z % 8;
z = x & 9;
x = y | 10;
y = z ^ 11;

if (n == ((x + y) * z)) {
  Pass();
} else {
  Strike();
}
```

"Okay, we've got Lines of Code." "Uh... how'd you hack into the mainframe?"

This module consists of a display with some code on it, and a keypad which consists of 10 number buttons, a negative sign button, a backspace button, and a enter button. To solve the module, enter a number that will modify the code in such a way that the module solves. However, if you enter a number that will modify the code in such a way that a strike will occur, the module will strike.

Use the table below to determine what each line of code means. Note that when a variable gets a new value, it **overwrites** the old value. Lines of code are run from **top to bottom.** So all lines above a particular line must run first, and all lines below that line must run after.

Code:	Meaning:
`int n = 0;`	The integer variable 'n' gets assigned the value of 0. Integers cannot have any decimal points, **so if a result has any decimals, you simply remove them.** If the name on the left side is 'n', the number on the right side of the equation is the number which can be modified and entered using the keypad, all other value names cannot be modified.
`x = y;`	The variable 'x' gets assigned the value of 'y'.
`x = y + z;`	The variable 'x' gets assigned the value of the sum of 'y' and 'z'.
`x = y - z;`	The variable 'x' gets assigned the value of the difference of 'y' and 'z' (specifically, 'y' minus 'z').
`x = y * z;`	The variable 'x' gets assigned the value of the product of 'y' and 'z'.
`x = y / z;`	The variable 'x' gets assigned the value of the quotient of 'y' and 'z' (specifically, 'y' divided by 'z').
`x = y % z;`	The variable 'x' gets assigned the value of the remainder after you divide 'y' by 'z' (Also known as modulo 'y' by 'z').
`x = y & z;`	The variable 'x' gets assigned the value when you use an AND gate for inputs 'y' and 'z'.
`x = y | z;`	The variable 'x' gets assigned the value when you use an OR gate for inputs 'y' and 'z'.

`x = y ^ z;`	The variable 'x' gets assigned the value when you use an XOR gate for inputs 'y' and 'z'.
`x = y << z;`	The variable 'x' gets assigned the value when you bitshift the value of 'y' to the left a 'z' number of times.
`x = y >> z;`	The variable 'x' gets assigned the value when you bitshift the value of 'y' to the right a 'z' number of times.
`x += y;`	Increments the value of 'x' by 'y', then assigns 'x' to that value.
`x -= y;`	Decrements the value of 'x' by 'y', then assigns 'x' to that value.
`x *= y;`	Multiplies the value of 'x' by 'y', then assigns 'x' to that value.
`x /= y;`	Divides the value of 'x' by 'y', then assigns 'x' to that value.
`x %= y;`	Moduloes the value of 'x' by 'y', then assigns 'x' to that value.
`x &= y;`	Uses an AND gate on 'x' and 'y', then assigns 'x' to that value.
`x |= y;`	Uses an OR gate on 'x' and 'y', then assigns 'x' to that value.
`x ^= y;`	Uses an XOR gate on 'x' and 'y', then assigns 'x' to that value.
`x <<= y;`	Bitshifts the value of 'x' to the left a 'y' number of times, then assigns 'x' to that value.
`x >>= y;`	Bitshifts the value of 'x' to the right a 'y' number of times, then assigns 'x' to that value.
`x++;`	Increments the value of 'x' by 1.
`x--;`	Decrements the value of 'x' by 1.

`if (n == x) {`
`Pass();`
`} else {`
`Strike();`
`}`
	If the value of 'n' is equal to that of 'x', the module will solve. Otherwise, the module will strike.

For accuracy of the code, `Pass();` should be changed to `GetComponent().HandlePass();`, and `Strike();` should be changed to `GetComponent().HandleStrike();`.

Appendix LogicBits: Logic Gates, Bit Shifting, and Two's Complement

All numbers used in this module are 32-bit integers, meaning that all numbers are made up of 32 bits which can either be a 1 or a 0. Values of 32-bit integers can go from 2147483647 to -2147483648. If a value goes over the maximum value or under the minimum value, wrap around to the other side.

Two's Complement (https://en.wikipedia.org/wiki/Two%27s_complement):

If a negative number is used, convert to binary, invert the bits (change all 0s to 1s and vice versa), then add 1 before doing any operations.

Logic Gates:

To use logic gates on values, convert the number to binary, then use the gates below on each place value, put them into a binary string using the order of the place values, then convert back into decimal.

- 0 & 0 = 0, 0 & 1 = 0, 1 & 0 = 0, 1 & 1 = 1
- 0 | 0 = 0, 0 | 1 = 1, 1 | 0 = 1, 1 | 1 = 1
- 0 ^ 0 = 0, 0 ^ 1 = 1, 1 ^ 0 = 1, 1 ^ 1 = 0

Bit Shifting:

Bit shifting is simply moving the bits of a binary number left a number of bits or right by a number of bits. Shifting left by one place value has the effect of doubling the number, and shifting right by one place value has the effect of halving the number.

When left shifting, any newly created bits will be 0. When right shifting, any newly created bits will have the same value as the original MSB. In both cases, any bits which no longer fit in a 32-bit value will be "deleted". Note that the right argument to a shift operation is limited to be between 0 to 31, so take argument modulo 32 so it fits within the range.