First implement the python examples in C++ (*port: same functionality in C++ program as original python program*), then solve parts a) - e) on the following pages in C++

Part A: Eftirfarandi verkefni eru gefin sem python forrit.
Útfærið sömu virkni í C++

Skrifið forrit sem skrifar út fastan streng ("Hello C++").

Skrifið forrit sem leggur saman tvær tölur og skrifar þær út.
Látið forritið skrifa þær út þannig að það sé texti með (8 plus 3 is 11).
Látið forritið taka tölurnar inn sem inntak.
Prófið hvað gerist ef maður slær ekki inn tölu heldur eitthvað annað.

Skrifið forrit sem tekur inn staf sem inntak.
Ef stafurinn er 'a' skrifa út: **You have entered the first letter of the alphabet.**
Ef stafurinn er 'o' skrifa út: **'o' could refer to "object" or maybe "output".**
Ef stafurinn er 'c' skrifa út: **CONTROL CHARACTER!**
Annars ætti forritið að skrifa út að inntakið sé ekki þekkt.
Jafnvel má forritið skrifa út hvaða inntak er þekkt og hvernig á að nota það.

Skrifið forrit sem tekur inn staf sem inntak.
Ef stafurinn er 'p' látið forritið taka inn tvær tölur, leggja saman og skrifa úttakið
Ef stafurinn er 'm' látið forritið taka inn tvær tölur, taka mismun og skrifa úttakið
Annars ætti forritið að skrifa út að inntakið sé ekki þekkt.
Skrifa jafnvel út hvaða inntak er þekkt og hvernig á að nota það ('p', 'm', 'q')
Látið forritið keyra virknina sjálfkrafa aftur og aftur þangað til inntakið er 'q'

Skrifið forrit sem spyr hversu hátt á að telja.
Látið forritið skrifa út jafnmargar línur, t.d. "The number is <i>"
Látið þessa virkni endurtaka sig alveg þangað til slegið er inn 0

*Útfærið liði **a)** - **e)** á næstu blaðsíðum í C++*.

Part B: Build the following functionality in C++:

info:
The bit shift operators << and >> shift all the bits in a variable a number of spaces left or a number of spaces right. Test them and see what the end bit is set to, specifically in an unsigned variable.

```
unsigned int i = 256;
unsigned char c = 1;

c = c << 1; //one space left
i = i >> 2; //two spaces right
```

To get the numeric (ascii) value of char, rather than the character, you can typecast the value:

```
int i = (int)c;
cout << (int)c << endl;
```

a)
Define an unsigned character variable.
Print the variable's numeric value to the screen with some text to clarify.
Set the variable to 0.
After each step, print text saying what the operation was and the print the variable.
Operation can be "add 1 to value" or "shift bits one place to the left".
Experiment with values here.

b)
Make a program that keeps track of a single unsigned char value.
Give the user a choice; to add 1 to value, shift bits one place left, shift one place right or quit.
After every choice print the new value and allow the choice again.
Experiment with these operations and keep track of the underlying 8 bit string on a piece of paper to confirm the program's results.

info:
The bitwise operators | (or) and & (and) take two operand variables and a third result variable and for each place they set the result bit to the "or" or "and" of the operands.
a = b & c;
a = b | c;
c = c | 0x01;

The number format 0xNM is the hexadecimal format or HEX for short.
0x36 is pronounced "HEX thirty six".
it's value is 3*16 + 6.
The possible values for N and M are 0-9, A, B, C, D, E and F.
The value 0xFF is the highest value for an 8 bit HEX number.
0xFAB3E is also possible. Each letter represents 4 bits. This makes it a valuable method to store bit values in a readable format.

c)
Now, in the previous program, add the possibility of setting the rightmost (least significant) bit to 1 or setting it to 0. Note the difference between setting a bit to 1 and adding 1 to the value.
First, on a piece of paper, decide how you will set the rightmost bit to 1 without changing any of the other bits, also making sure that if it is already 1, it stays 1 with nothing else changing. Then make the same decision for 0.

d)
Make a function called is_bit_set which takes a character and a place as parameters.
Its declaration could look like this:
bool is_bit_set(unsigned char byte, int place)

The operation should return a boolean value, true if bit number place is 1, false if it is 0, regardless of the values of other bits. How can you use the bitwise operators & and | to do this? Should the leftmost or rightmost bit have the lowest place number? Should that bit be number 0 or number 1?

e)
In data structures students have sometimes been asked to make a recursive operation to check if a number is odd or even. That's fun and all but not the most efficient method. How can you use the operators we've learnt in this part to make an efficient method is_odd or is_even that takes an (unsigned?, does it matter?) integer and returns a boolean value stating whether it's even or odd?
Make a program that asks for numbers constantly until you input 0. Tell the user if each of them is even or odd. Try allowing negatives as well.