Compression is used to reduce data size. We use this to zip up data, transmit data over the internet or even to obfuscate data to make it less readable to anyone but us. If we are transmitting data over the internet we want to send as little data as possible. The more data that is sent, the more time it takes to receive it. If we have data for our game, we want to obfuscate it so a hacker cannot change the data and cheat.

The compression algorithm we will be using is a modified version of Huffman encoding. The idea is to use as few bits as possible to represent our text. People have done some analysis of the English language and determined the frequency of usage of each character. We are going to use that information to encode our data. Space is the most common used character, therefor we want it to use as little data as possible. e is the second most common character followed by t, a, o, I, n, s, h, r, d, l, c, u, m, w, f, g, y, p, b, v, k ,j ,x, q, z. We are going to leave out punctuation and capital letters.

We will place our letters into groups using a vector of strings (a dynamic array of strings, more on vector in a few weeks).

const int NUMGROUPS = 4;

std::string groups[NUMGROUPS] = { " e", "taoi", "nshrdlcu", "mwfgypbvkjxqz" };

Group 0 has ' ' & e  
Group 1 has t, a, o, i  
Group 2 has n, s, h, r, d, l, c, u  
Group 3 has m, w, f, g, y, p, b, v, k ,j ,x, q, z

Since we have 4 groups we can tell what group we are by using 2 bits (from 0b00 – 0b11). So a space would be represented by 2 group bits (0b00) and 1 index in the group bit (0b0) for a total of 3 bits (0b000). An 'o' would be represented by 2 group bits (0b01) and 2 index in the group bits (0b10) for a total of 4 bits(0b0110). 'c' is in group 3 (0b10) and index in the group 0b110, so 0b10110.

We want our bits to be packed as tightly together as possible.   
"c" = 0b10110,   
"cc" would be 0b10110 + 0b10110 = 0b1011010110   
"ccc" would be 0b10110 + 0b10110 + 0b10110 = 0b101101011010110

**Encoding**

When writing out the bits we will be writing from left to right (MSB to LSB). We will be writing things out using a char (8bits) because that is the smallest size we use in C++. If we are to encode "ccc" we first initialize a character to 0.   
7 6 5 4 3 2 1 0 (bit positions)  
0 0 0 0 0 0 0 0

Then we write the group bits (0b10) for our first character (the first 'c')  
7 6 5 4 3 2 1 0 (bit positions)  
1 0 0 0 0 0 0 0

Next write the index in the group (0b110)  
7 6 5 4 3 2 1 0 (bit positions)  
1 0 1 1 0 0 0 0

Now we want to write the next character in our string which is another 'c'.   
First write the group bits  
7 6 5 4 3 2 1 0 (bit positions)  
1 0 1 1 0 1 0 0

Next write the index in the group (0b110), but we don't have enough space.   
Add another 8 bits to our string  
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 (bit positions)  
1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0   
Now write the index in the group value (0b110)   
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 (bit positions)  
1 0 1 1 0 1 0 1 1 0 0 0 0 0 0 0   
Notice here that the second 'c' will span 2 bytes of data

Write the final character of our string (again a 'c').   
First write the group bits (0b10)  
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 (bit positions)  
1 0 1 1 0 1 0 1 1 0 1 0 0 0 0 0

Now write the index in the group value (0b110)  
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 (bit positions)  
1 0 1 1 0 1 0 1 1 0 1 0 1 1 0 0

We now have a compressed string of 0xB5AC which represents the string "ccc", so 16 bits to represent 3 characters (that is compression)

**Decoding**

Giving the string 0x91AD58 let's decrypt it.   
The first 2 bits will be the group number.   
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 (bit positions)  
**1 0** 0 1 0 0 0 1 1 0 1 0 1 1 0 1 0 1 0 1 1 0 0 0

The group number is 0b10 which is group 2.   
Group 2 uses 3 bits for the index in the group  
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 (bit positions)  
1 0 **0 1 0** 0 0 1 1 0 1 0 1 1 0 1 0 1 0 1 1 0 0 0

Group 2 index 2 is 'h'

The next 2 bits will be the next group number  
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 (bit positions)  
1 0 0 1 0 **0 0** 1 1 0 1 0 1 1 0 1 0 1 0 1 1 0 0 0

The group number is 0b00 which is group 0.   
Group 0 uses 1 bit for the index in the group  
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 (bit positions)  
1 0 0 1 0 0 0 **1** 1 0 1 0 1 1 0 1 0 1 0 1 1 0 0 0

Group 0 index 1 is 'e'

The next 2 bits will be the next group number   
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 (bit positions)  
1 0 0 1 0 0 0 1 **1 0** 1 0 1 1 0 1 0 1 0 1 1 0 0 0

The group number is 0b10 which is group 2.   
Group 2 uses 3 bit for the index in the group  
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 (bit positions)  
1 0 0 1 0 0 0 1 1 0 **1 0 1** 1 0 1 0 1 0 1 1 0 0 0

Group 2 index 5 is 'l'

The next 2 bits will be the next group number   
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 (bit positions)  
1 0 0 1 0 0 0 1 1 0 1 0 1 **1 0** 1 0 1 0 1 1 0 0 0

The group number is 0b10 which is group 2.   
Group 2 uses 3 bit for the index in the group  
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 (bit positions)  
1 0 0 1 0 0 0 1 1 0 1 0 1 1 0 **1** **0 1** 0 1 1 0 0 0

Group 2 index 5 is 'l'

The next 2 bits will be the next group number   
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 (bit positions)  
1 0 0 1 0 0 0 1 1 0 1 0 1 1 0 1 0 1 **0 1** 1 0 0 0

The group number is 0b01 which is group 1.   
Group 1 uses 2 bit for the index in the group  
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 (bit positions)  
1 0 0 1 0 0 0 1 1 0 1 0 1 1 0 1 0 1 0 1 **1 0** 0 0

Group 1 index 2 is 'o'

The next 2 bits will be the next group number   
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 (bit positions)  
1 0 0 1 0 0 0 1 1 0 1 0 1 1 0 1 0 1 0 1 1 0 **0 0**

We have no more bits, so we are done. Our string is 'h'+'e'+'l'+'l'+'o' = "hello"

Again, if we decrypt additional data it will map to a space. We can easily remove the white space (the driver file already does this).

**Additional**

To make this as robust as possible don't hardcode the number of bits to be read. If we wanted to add punctuation and capital letters it should be as easy as adding another string to our groups and changing the const for NUMGROUPS.

To find the number of bits required to store a value use the following function:

int numBits(int value) {

int requiredBits = 1;

while (value >>= 1) {

requiredBits++;

}

return requiredBits;

}

I would suggest that you make some helper functions in your cipher.cpp file for read & write bit.

We will discuss this more when we discuss STL but string has a find function where you can give it a character and it will return the index of the character or it will return std::string::npos if the character was not found. This return value is a size\_t but that can be converted to an int.