Project Name- Encoding many strings stored as an array into Base64 efficiently

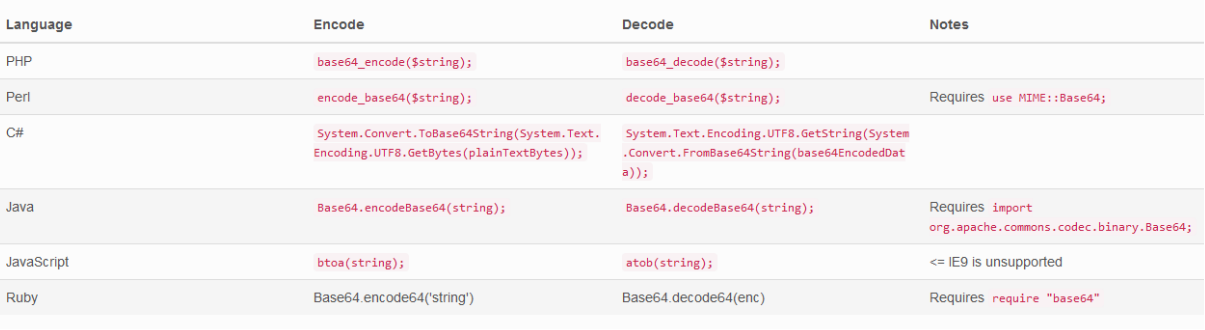
Project Category- Encoding/Decoding, Base64

Pre-requisites- Base64 Encoding, ASCII Values

Motivation-

Since many languages (except C/C++) have inbuilt function to Encode and Decode strings. So I thought to implement my own program to encode a string in C/C++. The "Decoding" part will be done in the next project.

Inbuilt functions in various languages to Encode/Decode strings -



Language	Encode	Decode	Notes
PHP	`base64_encode($string);`	`base64_decode($string);`	
Perl	`encode_base64($string);`	`decode_base64($string);`	Requires `use MIME::Base64;`
C#	`System.Convert.ToBase64String(System.Text.Encoding.UTF8.GetBytes(plainTextBytes));`	`System.Text.Encoding.UTF8.GetString(System.Convert.FromBase64String(base64EncodedData));`	
Java	`Base64.encodeBase64(string);`	`Base64.decodeBase64(string);`	Requires `import org.apache.commons.codec.binary.Base64;`
JavaScript	`btoa(string);`	`atob(string);`	<= IE9 is unsupported
Ruby	Base64.encode64('string')	Base64.decode64(enc)	Requires `require "base64"`

Explanation-

The below text has been taken from - http://www.hcidata.info/base64.htm . I felt that the explanation given in this website is the best I can find on the web in terms of simplicity.

What is Base 64 Encoded Data?

Put simply, base64 encoded data is a string of character that contains only a-z, A-Z, 0-9, + and / characters and is often used in situations when sending non-text information via a text only transmission protocol.

Why use Base 64 Encoding?

The transport system of e-mail protocol is designed for sending plain ASCII text. The plain ASCII text includes a-z, A-Z, 0-9, ", %, &, ', (,), *, +, -, _ and , (the comma). They will be referred to as the universal ASCII character set. There are other ASCII characters, but these are not universal. If an e-mail is sent using non-universal character (for example the £ symbol) the receiver of the e-mail may see a completely different character on their screen or printer. The situation is ever worse when trying to send e-mails in non English languages. Often these languages will have characters that are not part of the ASCII set. What is needed is a method to convert non-universal ASCII characters to a common form. This common form is called Base64 encoding.

What does Base 64 Encoding Do?

Base 64 Encoding takes a stream of characters and converts them to characters that belong to the universal ASCII character set. Once a stream of characters has been converted to characters that belong to the universal ASCII character set (Base 64 encoded) they can be transported with ease over the Internet using the e-mail protocols.

How does Base 64 Encoding Work?

ASCII characters belong to the 256 byte character space and includes the usual alpha, numeric, special characters (e.g. %), control characters (e.g. CR, LF, BELL) and other characters. Each character is 8 bits in size. For some languages the ASCII character set cannot be used (e.g. oriental pictographic languages). These use a UNICODE character set where each characters needs 16 bits to represent it.

The number of bits per character is not a problem for Base 64 Encoding. Base 64 Encoding takes a stream of bits and converts them to 8 bit characters that belong to the universal ASCII character set. Base 64 Encoding does not care about how many bits (8 or 16) are necessary to make a character as it works at the bit level. Once a stream of bits have been converted to characters that belong to the universal ASCII character set (Base 64 encoded) they can be transported with ease over the Internet using the e-mail protocols.

Base64 Encoding takes three bytes of character data (3 ASCII characters or 1½ UNICODE character) and produces four bytes of data from the universal character set. That is, Base64 Encoding takes 24 bits of input data and converts it to 32 bits of encoded data.

Base64 Encoding does not care what the data is. If the data contains line feeds, nulls or special characters, Base64 Encoding does not care - it simply sees it as a sting of bytes.

Encoding "Mary had" to Base 64

In this example we are using a simple text string ("Mary had") but the principle holds no matter what the data is (e.g. graphics file). To convert each 24 bits of input data to 32 bits of output, Base 64 encoding splits the 24 bits into 4 chunks of 6 bits. The first problem we notice is that "Mary had" is not a multiple of 3 bytes - it is 8 bytes long. Because of this, the last group of bits is only 4 bits long. To remedy this we add two extra bits of '0' and remember this fact by putting a '=' at the end. If the text string to be converted to Base 64 was 7 bytes long, the last group would have had 2 bits. In this case we would have added four extra bits of '0' and remember this fact by putting '==' at the end.

Each of these groups is then converted to a decimal number from 0 to 63. Base 64 gets its name from the fact that there are 64 states for each element. This is the same as decimal being called Base 10 are there are 10 states (0-9) for each element.

And finally, the decimal number is used as an index into the universal ASCII character set of ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/ to give the Base 64 Encoded string.

In our example:

ASCII text	Mary had
Hex representation of text	4D 61 72 79 20 68 61 64
Bit representation of text grouped by bytes	01001101 01100001 01110010 01111001 00100000 01101000 01100001 01100100
Bit representation of text in groups of 6 bits	010011 010110 000101 110010 011110 010010 000001 101000 011000 010110 010000=
Decimal representation of text in groups of 6 bits	19 22 05 50 30 18 01 40 24 22 16=
Base 64 encoded string	TWFyeSBoYWQ=

Decoding "TWFyeSBoYWQ=" from Base 64 to ASCII

The first thing to note is the '=' at the end of the Base 64 encoded string. A Base 64 encoded string will have zero, one or two '='s at the end. As '=' is not part of the Base 64 encoding, it can only ever appear at the end and has a special meaning.

The stages to convert the Base 64 encoded data to ASCII is:

1. Take each letter and find its position offset (0-63) within ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/ to give us 11 decimal numbers
2. These decimal numbers are converted to bit strings, each with 6 bits
3. As there was a '=' at the end of the Base 64 encoded data, remove 2 '0's from the end of the bit string. Had there been two '='s at the end of the Base 64 encoded data, 4 '0's would have been removed from the end of the bit string.
4. Split the bits into groups of 8
5. Use each group of 8 bits to be the ASCII code for an ASCII character or ASCII control code

Implementation-

Since the main objective of this program is to quickly/efficiently encode the strings stored in an array, so we are using two hash maps one to "*ASCII Character to corresponding 8-Bit Representation*" and another one to map from "*6-Bit Representation to Base64 Character*"

We will hence use two **unordered_map** to quickly insert and retrieve the values.

Time Complexity-

We insert and find the elements in the **unordered_map** in O(1) Amortized time.
Hence the overall time complexity of this program is **O(No_of_Strings * Avg_length_of_each_String)**

Auxiliary Space / Space Complexity-

We are using two **unordered_map** containing 128 and 64 elements . So considering them as constant space, we can say that we have used **O(len)** auxiliary space as we are storing the cumulative bits in the string-'*grouped*'.

References-

http://www.hcidata.info/base64.htm