Indeed some links are a lot larger that others. Larger links equals a much complex and discrete path address. Being discrete is not a big deal, but that makes it harder to share or introduce into a project. Hence we found a way to rectify this problem.

We developed a set of algorithms to solve the ‘Long URL Address’ problem and implemented them on our website that we created via the conventional web development tools, like HTML, CSS, Ruby on Rails and jScript.

{

**HTML**-For structure

**CSS**-For layout and styling

**JS**- For interaction

**Ruby On Rails** - Framework

}

**We called this web based application : “*ShBee.Ly*”.**

The word short means, well, *short.* But when we talk about shortening an url into something much easier to access, we need to see length as no. of characters. The smaller we can reduce the address character size, the better we can access it via a third party gateway.

Hence we had set ‘30’ as our initial, default limit. And we judge how to manipulate the entered url by checking if it’s length is larger than 30.

This is what a normal URL can be pictured as :

**https://<host>/<path\_address>**

Even if the syntax looks innocent, this URl can turn into something wicked in real life. eg.,

[**https://www.google.co.in/search?q=usual&rlz=1C1CHBF\_enIN746IN746&oq=usua&aqs=chrome.1.69i57j0l5.12623j1j7&sourceid=chrome&ie=UTF-8**](https://www.google.co.in/search?q=usual&rlz=1C1CHBF_enIN746IN746&oq=usua&aqs=chrome.1.69i57j0l5.12623j1j7&sourceid=chrome&ie=UTF-8)

This is the code we use to calculate the new shortened address on ***ShBee.Ly*** .

(language used == C++)

CODED DEMO :

**#include<iostream>**

**#include<algorithm>  //for defining character’s range**

**#include<string>**

**using namespace std;**

**string idToShortURL(long int n)**

**{**

**// Character Map to store 62 possible characters**

**char map[] = "abcdefghijklmnopqrstuvwxyzABCDEF"**

**"GHIJKLMNOPQRSTUVWXYZ0123456789";**

**string shorturl;**

**//convert given string id to a base62 number**

**while (n)**

**{**

**// use above map to store actual characters as a new gateway shortcut**

**// in shortened url**

**shorturl.push\_back(map[n%62]);**

**n = n/62;**

**}**

**// Reverse shortURL to complete base conversion**

**reverse(shorturl.begin(), shorturl.end());**

**return shorturl;**

**}**

**// Function to get integer ID back from a short url**

**long int shortURLtoID(string shortURL)**

**{**

**long int id = 0; // initialize result**

**// A simple base conversion logic**

**for (int i=0; i < shortURL.length(); i++)**

**{**

**if ('a' <= shortURL[i] && shortURL[i] <= 'z')**

**id = id\*62 + shortURL[i] - 'a';**

**if ('A' <= shortURL[i] && shortURL[i] <= 'Z')**

**id = id\*62 + shortURL[i] - 'A' + 26;**

**if ('0' <= shortURL[i] && shortURL[i] <= '9')**

**id = id\*62 + shortURL[i] - '0' + 52;**

**}**

**return id;**

**}**

**int main(){**

**//shortURLtoID();**

**string url;**

**cout << "Enter an URL to be shortened : ";**

**cin >> url;**

**int siz = url.size();**

**int id = shortURLtoID(url);**

**//cout << id;**

**string shurl = idToShortURL(id);**

**if(siz<=30)**

**{**

**cout << "Url is already Short enough ...";**

**}**

**else**

**{**

**cout << endl<< "Shortened URL : \n";**

**cout << "**[**www.shbee.ly/**](http://www.shbee.ly/)**"<<shurl; // name of the gateway**

**}**

**}**

Algorithms used for url shortening:

1. Take input as url.
2. Check if length of url <=30
3. If True : no need for shortening the url
4. Else : shorten the urls using specified functions.

*(We convert all the urls to a unique id, which is used to make the conversion much safer. There could be 62 combinations for an path address code in an URL including [a-z A-Z 0-9]. To form an id, we just need to convert a base 62 number(each number denoting a character in character map string) to a decimal no.)*

*(now this unique id is used to formulate an even unique url address, that is now much shorter and versatile than the one used before.)*

The above code though only calculates a much secure and a shorter path address. Yet to protect our shortened URLs from malicious attackers we add another few elements to our URLs:

What Other URL Shortners do-

**https<protocol>/bit.ly<USS-name>/fwefwf<random 5-6 char path generated>**

What we do-

**https<protocol>/bit.ly<USS-name>/www.wikihow.com<Original host name>/ffdsfw<securely generated path>**

<Original Host name> adds a layer of security that if the USS get attacked, their links won’t die off.

<secure key> adds another layer to shortened URL, as it is provided to the original host so that they can only handle their page without any middleman(or attackers).thus, protecting the short URL to be redirected to any other malicious websites.