CS 105 | MP2: J ibwf b tfdsfu *(I have a secret)*

February 12–14, 2014

Jouspevdujpo

Pof pg uif nptu jnqpsubou ufdiopmphjft uibu bmmpx uif joufsofu up gvodujpo bt ju epft upebz jt uif bcjmjuz up dpnnvojdbuf tfdvsfmz. Fwfszuijoh gspn tipqqjoh, cboljoh, boe vojwfstjuz hsbeft bsf qspufdufe gspn bozpof fmtf cvu zpv sfbejoh uif jogpsnbujpo uispvhi b ufdiopmphz lopxo bt IUUQT.

Jotufbe pg xpsljoh xjui IUUQT, b ufdiopmphz uibu jowpmwft fyusfnfmz ifbwz-evuz nbui boe uppl zfbst gps b ufbn up eftjho, xf xjmm fybnjof b tjnjmbs bmhpsjuin uibu xjmm tujmm nblf pvs dpnnvojdbujpot npsf “tfdvsf”.

…oh, whoops! Allow me to start over…

Introduction

One of the most important technologies that allow the internet to function as it does today is the ability to communicate securely. Everything from shopping, banking, and university grades are protected from anyone else but you reading the information through a technology known as HTTPS.

Instead of working with HTTPS, a technology that involves extremely heavy-duty math and took years for a team to design, we will examine a similar algorithm that will still make our communications more “secure”.

…was that better?

The preceding two blocks of text both contain the exact same text, except the first block has been **encrypted** to prevent someone who might have found this paper lying around to easily read it. Specifically, the encryption method we used is called the **Caesar Cipher** and it is done by simply shifting the letters in the alphabet by a certain number of characters. In our case, we only shifted the characters by one.

Knowing how it was encrypted, we can set up a translation table that will allow us to **decrypt** the original message:

Encrypted:	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
Decrypted:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Using the translation table, we can start decrypting. The first word in the encrypted text was “Pof”.

- Looking at “P” in the encrypted row, it translates to “O”.
- Looking at “o” in the encrypted row, it translates to “n”.
- Looking at “f” in the encrypted row, it translates to “e”.
- …therefore, the “Pof” ➔ “One”.

In this MP, you will be writing a **Caesar Cipher** to both encrypt and decrypt text. When you have finished this MP, you will be able to encrypt a string and send that message to a friend who will only be able to read it if they can decrypt your message!

Remember: This MP is a **solo assignment**. You should review the academic integrity policy on the course website if you have any concerns about what is reasonable to do when completing this MP

Getting Started

Similar to MP1 and the labs, we have prepared a base set of files for you work from. You can find these files on the CS 105 website, under the “Assignments” tab, and under the “MP1” link. The file you will download is a zipped file and needs to be **extracted**.

Overview and Review of Lab #4

In order to encrypt or decrypt the String, you must visit each and every character, either shift or un-shift that character, and add the shifted or un-shifted character to a new string.

This process is very similar to your Lab #4: “A MessAge in a ZillIoN Glass bottles” where you looked at each character and checked if it was a capital (uppercase) letter. Instead of checking if each character is an uppercase letter, you should encrypt (or decrypt) the letter and add that to the String you will be returning. You should refer to Lab #4 to review the **`s.charAt()`** and **`s.charCodeAt()`** functions.

Encrypting the String

During Week #2 of lecture and as part of Lab #4, you have seen the **ASCII chart** – the numeric representation that the computer uses for each Character that makes up a String. The ASCII chart for the uppercase and lowercase letters is reprinted here:

60 +					70 +										80 +										90
5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

90 +			100 +										110 +										120 +		
7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2
a	b	c	d	e	f	g	h	I	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

In order to **encrypt** a character, similar to the encryption shown at the beginning of this document, you will need to find the ASCII value of your character and add to it the amount of characters you wish to shift it by. For example: encrypting the character “M” with a **`shift`** of 1 will in the character “N”; encrypting the character “Q” (ASCII value 81) with a **`shift`** of 5 will result in 81 + 5 = 86, which is “V”.

In addition to simply adding or subtracting the shift from the ASCII code, there are two important cases to consider:

- If a character is shifted past “Z”, it should “wrap” around as though “A” follows immediately after “Z”. This is quite easy: for capital letters, simply check if your ASCII value after the shift is greater than the code for “Z” (that is, greater than 90). If so, simply subtract 26 from the value.
- Lowercase letters must always shift to a lower case letter. Likewise, uppercase letters must always shift to an uppercase letter. Non-letters (spaces, punctuation, etc.) should not be shifted.

When you have the final ASCII value of your encrypted code, you only have a number! You need one last function in order to convert it back to a string. To convert an ASCII value back into a character, you will need to call the function **`String.fromCharCode(asciiValue)`**, where **`asciiValue`** is the variable storing your ASCII value. In your code, you might use this line to append the character onto the end of your result:

```
        result += String.fromCharCode(asciiValue);
```

Once you have completed your `encrypt()` function to **encrypt** the String and return it from the function, you should test your function – your encrypt should work just fine even with nothing in `decrypt()`. Here are some sample things to test:

- Using a shift of 5, “Illinois” should encrypt to “Nqqnstnx”.
- Using a shift of 20, “Champaign” should encrypt to “Wbugjucah”
- Using a shift of 5 again, “CZT” should encrypt to “HEY”

Reversing the Process: Decryption

Once you have a working encryption, the rest of the MP should be really easy! In order to write the `decrypt()` function to decrypt `str`, you should be able to use all of your code from the `encrypt()` function with two small changes:

- Instead of adding the `shift` to the ASCII value of the character, decryption requires you to **subtract** that value from the ASCII value (eg: the opposite of addition).
- Since we are subtracting, it is now possible to go below the ASCII value for “A” (65) or “a” (97). Instead of subtracting 26 when this happens, we will again do the opposite and add 26.

With these two changes, your `decrypt()` function will now reverse exactly what your `encrypt()` function had done. It’s time to test it!

Testing and Submitting Your Program

Since the `encrypt()` function and `decrypt()` function reverse each other, you can use any String to test this MP! Enter whatever you want in the text area, click “Encrypt”, and then click “Decrypt”. If the original string changed to garbage when you clicked “Encrypt” and then returned back to the original string after you click “Decrypt”, then your program works!

Scoring and Submission

This MP will be graded for correctness of encrypting and decrypting strings. In testing your MP, we will test several different input strings with different shift values. The point values for the tests are specified in the following table:

Grading Rubric for MP2	
Description	**Pts.**
`encrypt()`: Correctly encrypts simple strings?	**4**
`encrypt()`: Correctly wraps from Z→A?	**3**
`encrypt()`: Correctly ignores non-characters?	**3**
`encrypt()`: Correctly keeps the case of the letters? (upper/lowercase)	**3**
`decrypt()`: Correctly decrypts simple strings?	**4**
`decrypt()`: Correctly wraps from Z←A?	**3**
`decrypt()`: Correctly ignores non-characters?	**3**
`decrypt()`: Correctly keeps the case of the letters? (upper/lowercase)	**3**
MP fully complete?	**4**
Maximum Score:	**30**

To submit MP2, follow the link on the MP2 page on the CS 105 website for the submission page.