SOLUTION:
Fall 2017 CIS 3362 Week One Assignment

For questions 1 - 3 decode each message. The techniques used to encrypt the messages are given in parentheses right before the cipher text. In your write-up, explain the process you used to decrypt and include any code you might have used as an aid. Please do not use websites that automatically solve ciphers as most of your grade will be based on your description of the decryption process and original code you include in your write up.

Methodology (Questions 1 and 2): The following two questions ask you to decipher text encrypted with a shift cipher. Shift ciphers are a subset of substitution ciphers that rely on a key equal to how many characters each letter in the ciphertext is shifted from that of the plaintext. This means that once you determine the mapping of one letter, you can easily decrypt the rest of the message. A common strategy in cryptography is looking at letter frequency and patterns. By determining which letters are used most often, you can get a good guess at what letters they map to (just look up English letter frequencies). Alternatively, you may realize the brute-force method of trying all 26 possible keys is a much easier option with the help of a computer. This is the route I took, and I have included the relevant code (in C) below.

1) (shift)

ozwfoaddlzwjsafklghakzgmdvygtmqsfmetjwdds

Plaintext:
whenwilltherainstopishouldgobuyanumbrella

2) (shift)

nenahxwnrwlujbbfruuyaxkjkuhknqdwpahbrwlnrcbwnjaudwlqcrvn

Plaintext:
everyoneinclasswillprobablybehungrysinceitsnearlunchtime

Methodology (Question 3): Because the affine cipher is also a type of substitution cipher, you could still use the method of finding letter frequencies / combinations to narrow down key options and go from there. However, with only 312 possible keys, it is still much easier to take the brute-force option and try each key through the use of a program. (If you are curious as to why there are only 312 keys, refer to the chapter 2 course notes for a brief explanation and a list of which key values to use.) However, you still need to solve for the plaintext character in the encryption algorithm to get the formula needed for decryption; to do this, simply break down the example affine encryptions and work backwards. Again, I have included the code used to decrypt the question 3 ciphertext below.

3) (affine)

orhqfqjweuuosrgwrfuomovvdolwnjotwuerlfdwvcaefocruchfdcuwnjot
wumovvpwjwbwevwlmdwricqpjwekfdwaondwjfwxfuosobwicqhcjfdoueuu
osrgwrfohicqacgwpigichhoawpwhcjwaveuucrmwlrwuleierlfwvvgwfdw
uwajwfneuumcjlouugckorzeiaqfvwjerlicqejwfdwhojuffclcucicqmov
vswfenjotw

Plaintext:
infutureassignmentsiwillhideprizesandthelocationsofthoseprizeswillberevealedwhenyoubr
eakthecipher textsigiveyouforthisassignmentifyoucomebymyofficebeforeclassonwednesda
yandtellmethesecretpasswordissmokinjaycutlerandyouarethefirsttodosoyouwillgetaprize

Methodology (Question 4): Because this question requires encryption and not decryption, all that needs to be done is transform the encryption algorithm into code and plug in the given plaintext and keys. The code is below.

4) Using the affine cipher with the encryption keys a = 17 and b = 20, encrypt the following plaintext:

packmyboxwithfivedozenliquorjugs

Ciphertext:
puciqmlyveafjbanktydkhzagwyxrwso

Grading criteria: 5 pts for both question #1 and #2, 3 pts for the decrypted message, 2 pts for the explanation and code. 8 pts for question #3 with 5 pts for the message and 3 for the explanation or code, and 7 pts for question #4, grader decides partial in each case.

Shift Cipher Decryption (in C)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define ALPHABET_SIZE 26

void bruteForceShift(char ciphertext[]);

int main() {
	// Question 1 ciphertext
	char ciphertext[] = "ozwfoaddlzwjsafklghakzgmdvygtmqsfmetjwdds";
	bruteForceShift(ciphertext);

	return 0;
}

// This function takes a string of ciphertext and prints each of the 26 possible
// plaintexts with corresponding keys 0 to 25. (For this ciphertext, the only
// legible plaintext corresponds to key value 18.)
void bruteForceShift(char ciphertext[]) {
	int key, c;
	// Loop thorugh every possible key
	for(key = 0; key < ALPHABET_SIZE; key++) {
		printf("Key = %2d: ", key);
		// For each key, shift every character in the ciphertext left by the key value
		for(c = 0; c < strlen(ciphertext); c++) {
			// Adjust the current character's ASCII value to its position in the alphabet
			int adjustedCharVal = ciphertext[c] - 'a';
			// Use the shift cipher decryption formula on the adjusted character value
			int decryptedCharVal = (ALPHABET_SIZE + (adjustedCharVal - key)) % ALPHABET_SIZE;
			// Readjust the decrypted character value to its ASCII value for printability
			char decryptedChar = decryptedCharVal + 'a';
			printf("%c", decryptedChar);
		}
		printf("\n");
	}
}
```

Affine Cipher Decryption (in C)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Alphabet Size
#define A_S 26

// All possible values of key A
int aVals[] = {1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25};

int modInverse26(int keyA);
void bruteForceAffine(FILE* fp, char ciphertext[]);

int main() {
	// For this program, there will be a lot of plaintext output, and being able to search
	// for common words (e.g. "the") will allow you to quickly find the correct answer.
	FILE* fp = fopen("Plaintext.txt", "w");
	// Question 3 ciphertext
	char ciphertext[] = "orhqfqjweuuosrgwrfuomovvdolwnjotwuerlfdwvcaefocruchfdcuwnjot"
				"wumovvpwjwbwevwlmdwricqpjwekfdwaondwjfwxfuosobwicqhcjfdoueuu"
				"osrgwrfohicqacgwpigichhoawpwhcjwaveuucrmwlrwuleierlfwvvgwfdw"
				"uwajwfneuumcjlouugckorzeiaqfvwjerlicqejwfdwhojuffclcucicqmov"
				"vswfenjotw";
	bruteForceAffine(fp, ciphertext);

	return 0;
}

// Pre-condition: gcd(keyA, 26) = 1, Post-condtion: returns a inverse mod 26.
// This is lazy, it just tries each possible inverse.
int modInverse26(int keyA) {
	for (int i=1; i<26; i++)
		if ((keyA*i)%26 == 1)
			return i;
	return -1; // indicates that the inverse of keyA mod 26 doesn't exist.
}
```

```
// Given a file pointer and the string of ciphertext, this function will try all 316
// combinations of the 7 possible key A values and 26 possible key B values to print
// a list of each possible plaintext and its corresponding key values to a file. (In
// this example, the correct key values are Key A = 11 and Key B = 4.)
void bruteForceAffine(FILE* fp, char ciphertext[]) {
	int i, j;
	int keyA, keyB, keyAInv;
	// Loop though each of the possible values of key A
	for(i = 0; i < (sizeof(aVals) / sizeof(aVals[0])); i++) {
		keyA = aVals[i];
		keyAInv = modInverse26(keyA);
		// Loop through each of the possible values of key B
		for(keyB = 0; keyB < A_S; keyB++) {
			fprintf(fp, "Key A: %d, Key B: %2d\n", keyA, keyB);
			// Loop through each character in the ciphertext
			for(j = 0; j < strlen(ciphertext); j++) {
				// Adjust the current character's ASCII value to its position in the alphabet
				int adjCharVal = ciphertext[j] - 'a';
				// Plug the adjusted character value into the equation below. (This is the
				// equation found when solving for the plaintext variable in the original
				// affine cipher encryption equation : c = ap + b(mod26)).
				int decCharVal = (keyAInv * (A_S + (adjCharVal - keyB))) % A_S;
				// Readjust the decrypted character value to its ASCII value for printabilty
				char decChar = decCharVal + 'a';
				fprintf(fp, "%c", decChar);
			}
			fprintf(fp, "\n\n");
		}
	}
}
```

Affine Cipher Encryption (in C)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define ALPHABET_SIZE 26

void encryptAffine(char plaintext[], int keyA, int keyB);

int main() {
	// Question 4 key values and plaintext
	int keyA = 17;
	int keyB = 20;
	char plaintext[] = "packmyboxwithfivedozenliquorjugs";
	encryptAffine(plaintext, keyA, keyB);

	return 0;
}

// Given a plaintext string and keys A and B, this function prints the correspoding
// ciphertext as encrypted with the affine cipher.
void encryptAffine(char plaintext[], int keyA, int keyB) {
	int i;
	// Loop through each character of the plaintext
	for(i = 0; i < strlen(plaintext); i++) {
		// Adjust the current character's ASCII value to its position in the alphabet
		int adjustedCharVal = plaintext[i] - 'a';
		// Using the affine cipher encryption equation, plug in the plaintext character
		// and the keys passed to the function.
		int encryptedCharVal = (keyA * adjustedCharVal + keyB) % ALPHABET_SIZE;
		// Readjust the decrypted character value to its ASCII value for printabilty
		char encryptedChar = encryptedCharVal + 'a';
		printf("%c", encryptedChar);
	}
	printf("\n");
}
```