Project 4

Encryption/Decryption Algorithms:
arrays.h
arrays.cpp
test1.cpp

Due: Wednesday, 19-March-2014 Due: 6:00 pm. Accepted until: 11:59:59 pm

Grading: 60 points
10 points: test1.cpp -- testing portion
40 points: arrays.cpp -- runnability portion
10 points: "code quality" for arrays.cpp and test1.cpp

Note: We grade your LAST submit only
Partners are allowed on this one.
If you work with a partner, BOTH must submit all portions
"Partners" is defined as 2 current students. Need NOT be in the same lecture/discussion section
Both partners need to put both names and uniqnames in all cpp files
Note: if you begin this project with a partner, you must finish it with the same partner.

Style grading: If you do this project with a partner, we will randomly choose one of the partner's to grade for style. Both partners will be given the same style score.

Note: Beware the autograder-cheat-detector magic machine
Further details are with this spec.

Learning Objectives

- To have fun
- To gain experience with arrays
- To further understanding with functions using pass by reference

Overview of files

• arrays.h Holds only declarations of functions	• arrays.cpp Holds the definitions of functions listed in arrays.h
• test1.cpp Holds "int main()" for testing for all functions	

Overview

This problem involves several cipher algorithms to enable you to encode and decode messages to make them readable to people who are supposed to see what they contain. You will need to implement several "helper" functions in addition to the ciphers. This will require skills in manipulating characters and strings, the use of functions, and arrays (both 1-dim arrays and 2-dim arrays).

Approach to What should you do first

Testing/Implementation portion. The first part of this project consists of writing several functions according to specifications. All the RME's are within the GIVEN **arrays.h** file. You are to implement these functions as given. Do not alter any of the prototypes; if you do, your code will not compile in the autograder. The implementation code must be in a file named: **arrays.cpp**.

The best approach is to implement one function and test it. You need to do this anyway so why not benefit from your efforts. Make sure your 1st function works perfectly before moving onto the next function. Implement the 2nd function and test it. Implementation of functions is within: **arrays.cpp.** Your code for testing functions should be in **test1.cpp** – this is also where **int main()** goes.

Note: we strongly urge you to test as you go. Otherwise debugging may be very difficult.

arrays.h && arrays.cpp

This is where you will want to begin. Declared constants and function declarations are in arrays.h. All definitions go in arrays.cpp. We strongly suggest you test each function as you write it because with this project functions call many of the earlier functions. Without testing, you will have no idea of where your errors are coming from.

This project revolves around encrypting and decrypting messages via different ciphers.

`void caesarCipher(int arr[], int size, int key);`

Caesar Cipher shifts all values by a fixed amount. This fixed amount is the key. To encode a message just add the key value. To decode the message, subtract the key value.

The only rule is:
The key needs to be within range of -25 to 25 before applying it any value. Hint: there is an operator that does this.

Given:

```
int arr[] = {72, 101, 108, 108, 111, 33};
int size = 6;
caesarCipher(arr, size, 5);
printArray(arr, size);
```

will print:

```
77 106 113 113 116 38
```

now if you want to decrypt it:

```
caesarCipher(arr, size, -5);
```

prints:

```
72 101 108 108 111 33     <--- back to the
original
```

an easier way to test is:

```
string str = "Hello!";
int arr[SIZE];
int size;
charToInt(str, arr, size);
caesarCipher(arr, size, 5);
printArrayAsChar(arr, size);   // prints: Mjqqt&
caesarCipher(arr, size, -5);
printArrayAsChar(arr, size);   // prints:  Hello!  <--- back to the original
```

`void caesarCipher(char arr[], int& size, int key);`

This is the original version of Caesar Cipher. The Roman ruler Julius Caesar (100 B.C. – 44 B.C.) used a very simple cipher for secret communication. He substituted each letter of the alphabet with a letter three positions further along. Later, any cipher that used this "displacement" concept for the creation of a cipher alphabet, was referred to as a Caesar cipher. Of all the substitution type ciphers, this Caesar cipher is the simplest to solve, since there are only 25 possible combinations.

This original form only dealt with letters and therefore several more rules apply.

1) All characters other than letters and spaces are stripped from the message
2) Before you add the key, the message must be all UPPERCASE
3) The key is not applied to spaces
4) Following the shift, the characters must be put back within the 'A' - 'Z' range

You can add the key in the same manner

```
int val  = int('A') + 2;
cout << char(val) << endl;
```

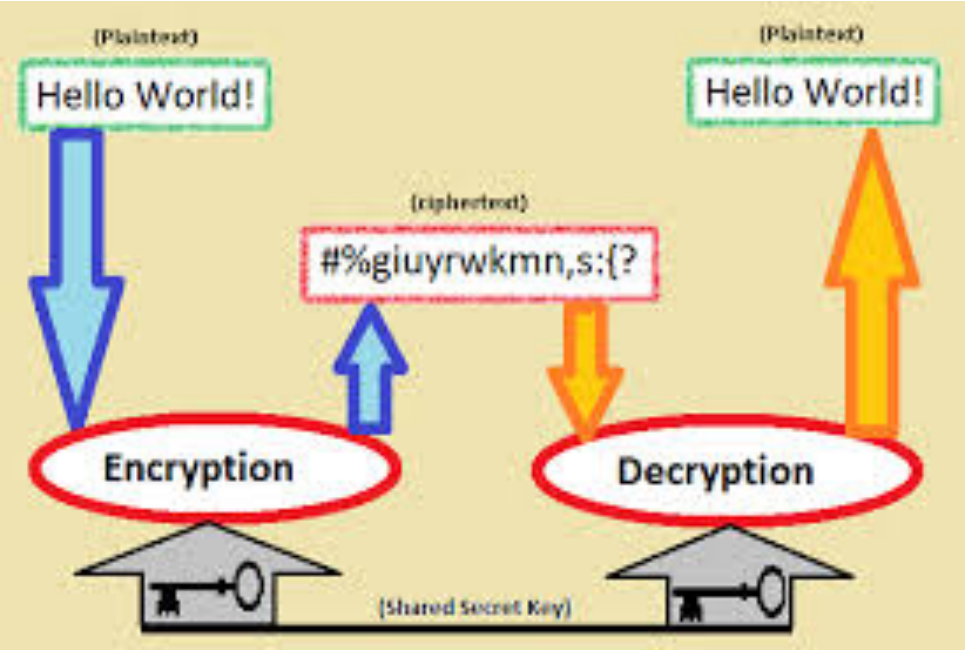
is not only legal but is very useful and gives the expected result, i.e., 'C'

Given:

```
str = "Hello from the 183 staff!";
char arr2[SIZE];
stringToChar(str, arr2, size);
printArray(arr2, size);
caesarCipher(arr2, size, 5);
printArray(arr2, size);
caesarCipher(arr2, size, -5);
printArray(arr2, size);
```

Prints:

```
Hello from the 183 staff!
MJQQT KWTR YMJ  XYFKK
HELLO FROM THE  STAFF
```



```
void vigenereCipher(char arr[], int& size, string key,
                    char code);
```

Vigenere Cipher is an encryption method which dates to the fifteenth century and is one of the truly great breakthroughs in the development of cryptography. The Vigenere Cipher was the most significant breakthrough in encryption for over one thousand years.

This cipher relies on a codeword to encrypt and decrypt the text letter by letter. The codeword is repeated as often as necessary and each corresponding letter is added to the text to form the encryption. For example, using the keyword "ACORN", the text "HELLO FROM EECS STAFF" is encrypted as:

```
  HELLO FROM EECS STAFF
+ ACORN ACOR NACO RNACO
======================
  HGZCB FTCD REEG JGAHT
```

Notice that when an A is used in the key, the encrypted letter is the same as the original letter- a B would make it go to the next letter (for example, an M would be encrypted as N), a C would go to two letters after, etc. The alphabet wraps using this code- the next letter after Z is an A (for example, a Y encoded with a D would give a B as a result). The amount that gets added (the offset or shift) is letter - 'A'.

To decrypt coded text (what you'll be doing in this project), the shift for each letter is done in the opposite direction:

```
  HGZCB FTCD REEG JGAHT
- ACORN ACOR NACO RNACO
 ======================
  HELLO FROM EECS STAFF
```

All letters should be uppercase before any shifting is done. Shifting is done on letters only. There are minimal rules that apply to the Vigenere Cipher:
1) All non-letters are stripped out of the message except for spaces
2) The key must be only letters
3) All letters in the message and key are uppercased.
4) Spaces are not shifted
5) Once a letter is shifted, make sure the resulting letter is within 'A' - 'Z' range

```
void monoAlphabetCipher(char arr[], int& arr_size,
                        char key[], int key_size,  char code);
```

The **Mono-Alphabet Cipher** is one step up from the simple Caesar Cipher. Ciphers in which the cipher alphabet remains unchanged throughout the message are called Mono-alphabet Substitution Ciphers.

If we permit the cipher alphabet to be any rearrangement of the plain alphabet, then we generate an enormous number of distinct modes of encryption. There are over 400 ,000,000,000,000,000,000,000,000 such rearrangements, which gives rise to an equivalent number of distinct cipher alphabets. If our message is intercepted by the enemy, who currently assumes that we have used a mono-alphabetic substitution cipher, they are still faced with the difficult task of checking all possible keys. If an enemy agent could check one of these possible keys every second, it would take roughly one billion times the lifetime of the universe to check all of them and find the correct one. (Resource: http://www.simonsingh.net/The_Black_Chamber/monoalphabetic.html)

For our cipher, the key doesn't use numbers but words. Suppose the key word is FEATHER. First remove duplicate letters yielding FEATHR. Next append the other letters in reverse order and add a space if the key did not already have one.

Your key will be:

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

Now encrypt the letters as follows:

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

Decryption is just the reverse of the above.

<u>Now that happens if the key has spaces within it</u>. Example if the key is FEATHER WITH SPACES. First you remove all duplicates yielding: FEATHR WISPC. Next append the other letters in reverse order and add a space if the key did not already have one -- it does in this example.

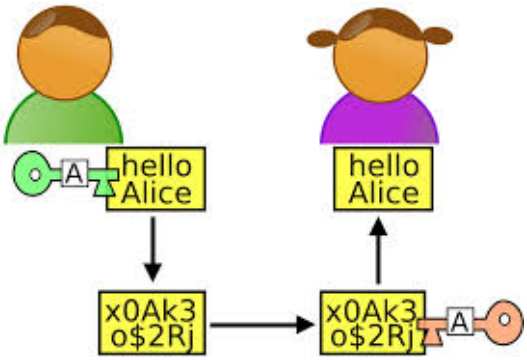
Your key will be:

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

Now encrypt the letters as follows:

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

`Hello from EECS 183 Staff` will be encrypted as:
`WHCCXBRQXZBHHAOBBONFRR`



Rules for the Mono-Alphabet Cipher
1) All non-letters are stripped out of the message and key except for spaces
2) All letters in the message are uppercased
3) All letters in the key are uppercased
4) The remaining letters in the alphabet are appended to the key in reverse order. Append a space if one is not already in key
5) All duplicates are stripped out of the key
6) If code is 'E' or 'e', then message is encoded. Otherwise decode the message

```
void EECS183Cipher(char message[], int& size,
                   const char key[], int key_size);
```

The **EECS 183 Cipher** is a cipher that replaces each pair of letters in the plaintext message with another pair of letters. Let's assume we want to encrypt the message:

MEET ME AT THE BIG HOUSE TONIGHT

Firstly, the sender and receiver must agree on a keyword. We will use CHARLES.
First we will append all letters of the alphabet to the end of CHARLES

CHARLESABCDEFGHIJKLMNOPQRSTUVWXYZ

Now pull out all duplicates including spaces:
CHARLESBDFGIJKMNOPQTUVWXYZ

Now eliminate either I or J -- whichever comes last in the list.
CHARLESBDFGIKMNOPQTUVWXYZ

You now have 25 letters left. Make a square out of them

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

You may assume that I/J both occupy the same square.

To encode: I'm going to break the plaintext message into pairs to make this more understandable.
ME ET ME AT TH EB IG HO US ET ON IG HT

Now look up the pairs in the square. Encryption falls into two categories
1) If both letters are in the same row or in the same column, then swap the letters
2) Otherwise, find the letters in the square. Now find the two letters that form the other corners of the rectangle. The first letter -- 'M' will be encoded as 'G' -- in the same row and one of the other corners of the rectangle. The second letter -- 'E' will be encoded as 'D'

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

Taking the 2nd pair: ET. The 'E' will be encoded as a 'D'. The 'T' will be encoded as an 'O'

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

The final Encryption will be:
GD DO GD RQ PR BE GI CP PF DO UG GI RP

If the number of letters in the message is odd, don't encrypt the last one.

Rules for the EECS 183 Cipher
1) All letters in the message are uppercased
2) All letters in the key are uppercased
3) All duplicates are stripped out of the key
4) All non-letters are stripped out of the message and key
5) The remaining letters in the alphabet are appended to the key 'A' - 'Z'
6) Either I or J is removed from the key -- whichever occurs last
7) A 2-dim array is made with the key that has 25 characters in it
8) Look-up a pair in the 2-dim array,
    a) If both letters are in the same row or in the same column, then swap the letters
    b) Otherwise, look along its row until you reach the column containing the second letter; the letter at this intersection replaces the first letter. To encrypt the second letter, look along its row until you reach the column containing the first letter; the letter at the intersection replaces the second letter.
9) If there is an odd number of letters in the message, don't encrypt the last one

The following code will test the EECS183 Cipher for the example above

```
string str = "Meet me at the Big House Tonight";
stringToChar(str, message, size);
str = "CHARLES";
stringToChar(str, key, key_size);
EECS183Cipher(message, size, key, key_size);
cout << "Encode: ";
printArray(message, size);
```

Testing: test1.cpp

Since testing is critical to the success of writing good code, your testing for all functions must go into file test1.cpp. **Note: it is considered good practice to write the test suite BEFORE you implement a function.** This way, once you have written code for a function, you can test it immediately. It is always good to know if you are correct or not -- without using a submit to the autograder. It is also a very good way to cut coding time by a significant amount.

Pros for testing as you go

- save significant development time
- pinpoint errors
- know immediately where the problem resides -- if there is a problem
- easy to do
- an extremely useful skill
- did we mention **save time**

There are more deals and hints on testing throughout this document and in the RME's of functions. Testing This MUST be done with NO input from the keyboard.

NOTE: Make sure you have that

```
#include "arrays.h"
```

at the top of your test1.cpp. It is needed or your code will not compile. This line is inserting all those prototypes that are in arrays.h into your program. It will make the compiler very happy.☺

Note: The "Requires" clause of printArray states:

size is the number of elements within the array arr. size > 0 && size <= SIZE

Therefore, the call

```
printArray(arr, -5)    // not a valid call
```

is NOT valid since it violates the "Requires" clause. The “Requires” clause means that this MUST be true when the function is called. It is not the responsibility of the function to check. It is the responsibility of the programming calling the function to guarantee that the "Requires" clause is adhered to. And if you do this and submit this code to the autograder, you will get a 0.

The feedback will state you violated the “assertion”.

The autograder is written to make absolutely sure you adhere to those “Requires” clauses.

Submission Requirements: arrays.cpp
test1.cpp

test1.cpp
The testing file gets submitted to

Project 4-Test Suite

due at 6:00 but accepted until 11:59:59 pm on the due date. You have 2 submits per day with feedback plus one "wildcard" submit for this portion of Project 4.

arrys.cpp
The main file: arrays.cpp should be submitted to

Project 4-Ciphers
due at 6:00 but accepted until 11:59:59 pm on the due date. You have 2 submits with feedback per day plus one "wildcard" submit for this portion of Project 4.

Partners - BOTH submit - plus names in program

Although you are welcome to work alone if you wish, we strongly urge you to consider partnering up for P4. If you would like a partner but don't know anyone in the class, we encourage you to post on Piazza's study group forums if you want to find someone.

As a further reminder, a partnership is defined as two people. You are encouraged to help each other and discuss the project, but don't share project code with anyone but your partner.

Both partner's names and uniqnames go in both students' submits, and BOTH must do the final submit. Your code may be identical to your partner's or be different, whatever you want. We will grade them all individually for runnability; partnering means we will ignore any similarities between the two partners. This also means that if only one partner submits only that one will get credit.

We STRONGLY suggest you start your project with the names/uniqnames entered such that you don't forget to do this.

Note: if you begin this project with a partner, you must finish it with the same partner.

Bonus Point Policy

Bonus Point Policy applies to runnability and test suite portions of this project. Bonus Points do NOT apply to the style portion.

Late Day Policy

You are allowed 3 late days for the entire semester with no penalty. The autograder is only a guideline but not exact.

If a 4th late day is used, you will receive 50% of your project score.
If a 5th late day is used, you will receive a 0 on the project. This is non-negotiable.

Keep in mind we take your last submit to determine bonus points and late days. We do not take the highest score. We do not go back and take an earlier submit. We do not excuse the one you didn't mean to submit. We do not excuse a submit because you didn't read carefully enough.

Check ctools/resources/Administrivia/policies for the full statement of "late" policies.

Code Quality:

We will review your functions.cpp file for code quality. This is done by extremely knowledgeable people. For details on what we will be looking for refer to:

ctools/resources/Style Guidelines

Read the style guideline so you will know: How to ACE "style" points

Partner Style grading: If you do this project with a partner, we will randomly choose one of the partner's to grade for style. Both partners will be given the same style score.

Array Bounds:

Note: even though C++ does not do bounds checking on array bounds, we set the autograder to check bounds. Therefore, your code could run in xCode or Visual and have a run-time error in the autograder. If you receive an error such as:

```
==14151== ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7fffdc05f1b0 at pc
0x403c62 bp 0x7fffdc05efc0 sp 0x7fffdc05efb8
WRITE of size 1 at 0x7fffdc05f1b0 thread T0
    #0 0x403c61 (/tmp/ag_grader/tmpBIZ6Em/stubbed+0x403c61)
    #1 0x40b631 (/tmp/ag_grader/tmpBIZ6Em/stubbed+0x40b631)
    #2 0x405586 (/tmp/ag_grader/tmpBIZ6Em/stubbed+0x405586)
    #3 0x319461ed1c (/lib64/libc-2.12.so+0x1ed1c)
    #4 0x401388 (/tmp/ag_grader/tmpBIZ6Em/stubbed+0x401388)
Address 0x7fffdc05f1b0 is located at offset 432 in frame <EECS183Cipher> of T0's stack:
  This frame has 7 object(s):
    [32, 36) 'ax'
    [96, 100) 'ay'
    [160, 164) 'bx'
    [224, 228) 'by'
    [288, 313) 'key_arr'
    [352, 432) 'temp'
    [480, 680) 'key'
```

and this goes on for many more lines --

This indicates that you have indeed exceeded an array bound. Go back to your code and find the issue.