1. obstacle:

One obstacle I meet is that after modifying multiple versions of the cpp file, I found that they can run successful with all test cases passed on my computer but always failed test cases on g31 server. I then trace very line of code but still found no errors. Later on, after I changed my eighth version from the file name “decrypt8.cpp” to “decrypt.cpp”, the g31 can run and all the test cases can pass. Now I know the reason I didn’t pass the g31 test was because I didn’t change my file name to the required file name “decrypt.cpp.

Another obstacle was that I’m very overwhelmed at first about how to deal with the program at the beginning. Later, after viewing the tips on the FAQ section on project 5, I started to cope with the simplest version of the question, like decrypt only one word at first. Later, I found that it is very useful to begin with the simplest version in order to get familiar with the thinking process.

1. Pseudocode:

bool decrypt(const char ciphertext[], const char crib[])

{

Check if crib or ciphertext is null

If the crib or ciphertext is null, return false

Create a key that contains 26 empty spaces

Create a Rciphertext that is exactly the same with ciphertext

Create a returnasnwer that is exactly the same with ciphertext

Create a Rcrib that is exactly the same with crib

Clean the Rciphertext

Clean the Rcrib

If Rcrib (after cleaned crib) is longer than 90 chars:

Check if there are spaces or newlines at the beginging or end of the Rcrib

If yes, check if it is longer than 90 chars without the spaces and newlines

If yes, return false

if there is spaces at the beginning or at the end of the Rcrib:

delete the number of spaces in the Rcrib and calculate the true length of Rcrib

if the i th item of Rciphertext is not alpha, move to the next letter

checking by moving along the Rcrib:

if the item in the Rcrib is space or newline,

move on checking the next letter in Rcrib

else

calculate the index of the letter in Rcrib in alphabet

if at the same index the Rciphertext is not a letter

break;

else

check whether the letter in Rciphertext has already appeared

if the letter has already appeared,

check if it is corresponding to the Rcrib letter at this moment

if not, break

if the corresponding key is a letter and the letter is not corresponding to the rciphertext letter at this moment,

move one;

else

input that new letter into the key

if at that moment, we are out of bound of the Rciphertext,

break;

if the total length that we matches is the same with the true length of Rcrib,

call answer(returnasnwer, key);

return true,

else,

clear the key to empty

move on to checking

}

void clean(char Rtext[])

{

Create a new rough array;

For every char in Rtext[]:

If the char itself is a newline,

Put a ‘@’ into the rough array;

Else if the char itself is not a letter,

Put a space into rough array;

Else

Put whatever the letter is into the rough array;

For every char in rough array:

If this char is empty and the previous char is also empty,

Keep moving forward;

Else

Put the char into the renew array

If there are space and ‘@’ encountered,

Cleared the space and only leave ‘@’ in the renew array

For every char in renew array:

Lower every chars in the renew array, and put every char in the Rtext[];

Cleared the renew array and the rough array

}

void clear(char clear[])

{

For every item in the clear[]:

Make every item in the array an empty char;

}

void answer(char want[], char key[])

{

Create a truth char array that put corresponding 26 letters in it with correct sequence;

For every char in the want array:

If the char is a alpha,

Lower it

If the alpha is not exist in the key array

Lower the corresponding char in the want array and leave it untouched;

Else

Put the corresponding upper deciphered letter into the want array

}

1. Test:

runtest("Hirdd ejsy zu drvtry od.\nO'z fodvtrry.\n", "my secret"); //ciphertext with \n; result is true

runtest("Hirdd ejsy zu drvtry od.\nO'z fodvtrry.\n", "shadow"); //ciphertext iwht \n; result is false

runtest("Hirdd ejsy zu drvtry od.\n", "12@(><?"); //crib is with non-letter; result is false

runtest("kvbz pqzzyfq bz zqxjqk", "secret"); //crib has repetition letters; result is true

runtest("kvbz pqzzyfq bz gqbjw", "secret"); //crib has repetition letters; result is false

runtest("kvbz pqzzyfq bz zq jqk", "secret"); //the corresponding answer is separated by space;result is false

runtest("kvbz pqzzyfq bz zq\njqk", "secret"); //the corresponding answer is separated by \n;result is false

runtest(" sdf rnto GX NMKLFWN", "bank"); // bank of america; result is true

runtest("Rswjo qgx Tmeuo sgjsy jds vqgf vo jds vqzf xbby.\nUdbyjo iqcju cg wybgj cg jds esjqiqo zqy\nXbg'j rsj jds jsrrsy jycn jds ucrsgj qrqyt.\nZU 31 cu zdqrrsgecge!", "silent alarm."); //silent alarm; result is true

runtest("@@@@UcRSgJ", "sIlent"); //different capitals, result is true

runtest("Rzy pkr", "dog"); //dog test, result is true

runtest("cdc ef", "aba"); //matches part of the phrases, result is true

runtest("kvbz pqzzyfq bz zqxjqk", "secret"); // matches the last word of the test

runtest("kvbz pqzzyfq bz gqbjw", "secret"); // the last word of the test is the same length as the crib but the letter is not corresponding

runtest("F gspt fe! zyxZYXzyx--Abca abCa bdefg## $$dsptrqtj6437 wvuWVUwvu\n\n8 9\n", " hush???hUSh--- --- until NovemBER !! "); // the brib contains multiple spaces, sybmbols, and newlines

runtest("23563eff@!\*@#", "\*(!abb"); // multiple numbers, symbols in both crib and ciphertext

runtest("\*@&#%&@ad ddde", ")(&(# cf\n fffl"); // \n in crib

runtest("", "asdlkfj"); //the ciphertext is empty

runtest("asdf", ""); // the crib is empty

runtest("\*&%(\*&%(\*&", "eblaksj"); // all symbols and numbers in ciphertext

runtest("alskddd", "\*&%(&^$&(^%"); // all symbols and numbers in crib

runtest("qwer tyu", "qw ertyu"); //letters are corresponding but spaces in the wrong place

runtest("qwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyuiopp", "qwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyuiopp"); // crib is 91 characters long

runtest("qwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyui op", "pb"); // ciphertext contains a line more than 90 letters

runtest("wrAv9ZehQxUBWMGhBG8M6ssyCYhxgOkc4WujI5KBqzhJq1JUgpFHiaT8wCRUnjfShogCwPfNjmitkD7u0DyJYwrHyx\n"

"QGREv9eGnsTTqIScbvAXZXcG1PDkL6fcW0ku1Wt4UAslSpjElOUBy84w95PEPMomC9HChnb1VS692nOD1kyI4RSHvn\n"

"tmOZYCcPqtUNZmhk7UTDse5VTRY9E2FYUQL5CPRL9Qng6i5xxrgYHTZaywyT1VQjR2u0n5oZz7t2Gbp4SnFb1yvPNs\n"

"u1BvWDrJqq9MIKSVw23zj1YTA7LP4bEXnRJY8EGP2IMvzfqTCBP6BMqHa0MrWCGbzj1iqIaVQ2b42G6oDxKobBZhtA\n"

"QcuPYIL2g0LbGgotixlQErpVgf0pk4LiTlPA6up6bV4V2yuBKedMBEHWVRRlVnaAzEivUJ7eKPPBiFYzFz4H7VDdfu\n"

"JfjTQ7N1I1IdWl9KkwLTikrdN4o3SifoaHk0C36apcjx5kPOQrWKqWWKnoC09ivNX6Y8Dqr55P9VVoU5lAoaQxL8Y4\n"

"PvZ4k9SL0CvEHvxlduV8Mpmy1p6PTsTp1Hgw2n9unkrAJiIXdzyRVsVGm4twT9XUddZRT0X46PR4id1KbGYdIc4EY6\n"

"llofBNLrf0SIqA24nbSOhmCucbzIHCL2TfzcFzRXx19RM3dXGJGpTCd46M7EZQ3RCh435tJEZMsj2WAK3kwkzVhxos\n"

"gDtVJwZH5aPpZT7Cdkt80BN9lglZukdiKoh7ZHnZD93hxM0YZbA29aI4dI5zMmRYymJU5sxeee790Ns8Js8MdVBnaF\n"

"5W0eB9m5ATkBKTYQ40V5Bx3BRQ6J3O1WDxjV6etGv2KXGE3WAKFDao4cDjdpqYGo7LkBHjAEzANWOtEYT3ibnqaPvG\n"

"ha5fe5KtrYikJi6qcJrQ1wVz6Lox3XqnL5jaFML4JE98pYeYU9A7IExSV7TUC01v70Itnsr9mqymaHTJ3rKmjhVCuH\n"

"EjQmeCSVgnlsOMoFvp8PedbjHOS3r1ohWJAqPNdKn7jSJKvgIgiQm3jA4XCmjq5lgCrHBgWRZGaHwubIMM3jDBqH8f\n"

"xvhte3nnJnOX6OIC5sBcBMjCNENEmNw9Z9XOKjjf8bJP2di78VV4dHXjYNvu0vNxUyFIvditg9G3o3siXxScvZJkvl\n"

"oO4lbkffSKkUIwMnEec4hbTpcnQp2CpPvIRfCAfzsDGBrLuD1ztF19IEJHkxiuuX5h8zdxUrXSlSdV3jeNYQXHlSOn\n"

"s2Aqlv7jlqPHcpCkqQloqRRg9O7pG49GyaYMY63lSIAX7VI7uiUFUJS0ePbdW8ik2EXQ89MKaePI98GpGR6XewJIiU\n"

"tq3Z0Hp1D0UPLgbVuRxPEfQhsHx5X63wsBny9I7siz3RPtJSn9YK7WJHt5QitR2lZwNhuh12WoDhWOqYEgGIX9Zxu2\n"

"GfS6LrnHCPCt8voKTTy1vfdu9sOjq0BNTxpvKdhkb779vVjJJXqSKFgwWVQ2ubJVGPjmQaYPN5vXkeChBQj0K9J25x\n"

"KL3qyDBe3nKTMqmx6cSe4Sue6TFFd6TNbJgDmKIJ8uAhIADEg2wLUkVS3Sdgdy9xnlPuCkb0HiHiBrTJDNrZE1NjKv\n"

"PGq3ZZgkL6QYzKiN6sktcZy8is1SQf02D7cwGmzBJ42WztJnmhbMomlijCGAzIPLmX8KcPj2mLvbg39kHOh1xBxMnh\n"

"gGer7arIBt6IrUXBPQiV588Q4N0rxhZIfiIKaM65CxQbgepgxDcwsLy1d0lKe4NgsL0ooPXpDk7ebRI3sjkoYYrXWD\n"

"yis9evhesWo6s9ULpq8GcCBR1lAzAOAHTy97jhcFIJvK854cVdMmGj1WR9ipBEGAHk2OGBYjoDi5bDN6LlFutCgsNY\n"

"aPSbKSsA660XeGh3DjPgWoVT4p5ANp0jlhHqAISERElF6w1JkFWvG0plaZGzvroDjNOBKYcH0Y5W8xbJwWfUOUEIAM\n"

"anHTG7oeFNcXtcgxPKL7oKKKRneBHmOFtEuKotAu7xXKjJget70H6MepKZZdQHS8IJ3FQzcIqSej0dqzZPmnqXKulu\n"

"aXoySnqRzPrEiKz39REPMe7HM3WqQdPQcrzg64UkJmKqDPj7P6ilnuP5AcEWf2xxoqq0ul4mLVwu5wedjLQR3UxxRg\n"

"CQPox2KabvWw8dJ4NFr6bGcWFJRKEck27mvT5PykMJEOV4bDkQ46j28GzrJxF6BGquXOX6qdy1DIocVKMWIx9JBWWZ\n"

"BnM9jOLCipq2eJajrSB0Xh5z06xbiALq5ftx3tbzuhmDhcgjUv0RWbpbJF3lF2w1Rt2f9hZuE1Dl4qtBWtsMSaR9ef\n"

"7Mu1XaBerXJ92fjnQ0aP6AHYKo5gI6Z31j2B4eOb4Avue54RObN7QcA8l2AYTgnGHghEM4lVcMUYsCTfQQqK20Lqml\n"

"vL5Z2y4pYcZ4eFCnAM3o6YnCJ2xawrI75g0qRKWQv3M9Hfx2LlHzL4vxyXax8JjrXnvDDSDJtMXfU49fxtYtb1Clez\n"

"jNgf9czoZNP79EM9pgpvukwSHxGzlCRYZDhgR2jRIAgKPaor8ZnIT0w3FP5ZcP3WFKj6y0ntECYXlDmGPQVdScD4VD\n"

"zmzg3VhdAaamCCUMudH2SBxlkMWO8oTdfg4nPJ1mXRQgT37LYz7IpXl2hqzDTaOpHx6jxoXgnjAYPtaga10LqjMEJ8\n"

"EKp2CTBLqFonhUWXnRar88R3CsWxqJ6ThoaCWDzG6NffHI3vh34GsoOD2663SShVi9UtjshvdxxxwLy1oG5NTktoLZ\n"

"VifmRhBxJXRXALYMbmFFaPgihHzBtwKYm7NvYdYMTy35BJPe79lZWJowm9a6RRforLOb1Ls0tnba9BYchB7NIxgklE\n"

"UelMlUYmBVkFV1N7uHEegEULvpl3K1FANv5o5H2wh92NrS657wh03vG1AR4axjuu0n9rWIuXLwfYB2xtevpBZ3n9Hd\n"

"QDCNMlBhOxbrWRwGniS6JUQvpSg0ouDrr2XfLQjypDxk13ZefaTFfiD8vSAcRPEsrXiVOjTVVw5ZagpVG1i8i52GXK\n"

"pZHbWeQbYfTLr9gK1Wazaaz8HSJarHhO3G5vtmpbk7KtCK6EUipg3R0nMHr4jYtaUO6WpSKaGG9CvxVsBqitsJl7ye\n"

"HLamZEs6zeOB2vxR37MDJHboYqdhzWeCCsQJlKvYnetl2ejdljGABWZnqgE3xdZxneylk45rBqOdb6Kt5DuthtPKT0\n"

"JrBvqfb0QrbC4NS1zQIAR1h8u7fiWqNccJrm7XREkBeYBRrlFidjcHvV0rK6bnJYhulK0eavhMNxNiTzTu18HS5upe\n"

"iLyoJUGMFDXvNzo2WXCrVuPzwFZJCCsNidDk7McivSMWHkN0ll5CwxbPsRqz3VOcx7eH8oWc2mIsqrCQGIHSJVT1eF\n"

"NafIg49tEdZJKumlI9edvRYgP8sGPcr8wAgrVjM54kRxdluLkpgv8ZBqSPDWHrBmy2leavIB8qpA1lmiHLLll7OYya\n"

"y83UPERcGWnZsnG92GRWu9DncN2dS2dN3kSEwDYME6RhxF2j5GvTIVTnMyyhhSJ7Yg2YE3yQyDUbdbyR0DExZzq90L\n"

"EDgVAXhgtq5MW2iXAmdyfcNplLAeZF3pXhQ69DPRDUgqXpBlDdbp34RRhAAdxoJcumbst65yk3k8caoiB2wf50RPQ6\n"

"76LUH1lyp8XXLiIPAZy4Um9rVGDxVJpn9e5626ow4VxKsbroTl9WWD9RHOB7vN4zzGleRKDHewqJgC94U7BY7rQ56K\n"

"TaRcaEBiESet06QNjxuVuJHINaV3xcUF7OfK3Ndimha3azQpExLmZMhG2RGY70X1uWu63CUJ5XEmfr4dnTYUlExmwH\n"

"mjIJK2JpQYSo7Sb8dAunLgi7UkYjhiYqkOHZ6S9095BEPsD2SniBUMmQwc0mYPkOOoPAgmzRX2YPAH5ElRGc7IDE6G\n"

"1l5nwlAsw48M7kqK8KWu1oVepL3Qf9cvHDXsw4Gjcri6y1VX1ivaBqcdzN8sXe1nGV2AhYJQDMZOGrc6zNz5HhHNbm\n"

"FRps1cMzQROHSAMRPG1aYqc5IMRCXnc98TTDD4CCa58EJfpXUjfbmx56qP73NBxIV0jf3MjdOubJQsio2arOjv4XSk\n"

"sLoBg00VAR1wYrWIM8KboBcsg1b9a9i9S1YLkE1QgsiSGCF04ZCikp8CyNPKWICR8IsSwK0zybJ9Xvi4C1XOme0d9q\n"

"Gfv7OXYGC7FDvQyHePgfy1oMJ5fBXEyIcLFbbu8ykziapxTx9ZVF6QO8PXeQhRYLFPOb2dow6wrY7FdeLqU7dAYMlA\n"

"TS6gUPqFTq7tNfGrwxuU8rOLDGaXFelev5Tx5aYUESG2aHb0CCtVWKmegmfky2wwxXV0lRKZcpbw42YkI6gZs8BdsQ\n"

"ESNMdWQK7busUGQacwgB2m3Z9xyLa5RaVQ42C40Jt5MquRk7pW9Hz5nZLcb1Lp6uLPcVNOyFlF4pmzVWsXuHgIu7BE\n"

"uUdJXaWJCfnfIbSj1hoMVXGgU4DYgdcfi6uHf7FETKHgdvBqzw8yJ9lpLaffOgxMVpiSPc09NjcGgdLAFdlLuAr9cR\n"

"NB2LL24Uxui3nxDxl8vRY2BThqBCXBIxpLvYzKm21PiThxgSR0NXuDQXv4cTZPl56UOkGeGJCm7RQPrMfrKldI9uAy\n"

"ahXE74xNGuR6tAOsZbs1hxxB4W9WMYASkktzJy3MqbDUM4jnJvVRk4U0q2pfZv6LWghzBY5lZOmQ5fnWrVZ3pwwF1A\n"

"Xxgpzw0oetg7Wumwa3gHobqNTXbeWj8Padzv2Vu6NUsxl1rNWl31Jip3PTUQeTyDtgL5c4fMyuKTbwGt140Tjk5dtH\n"

"WbnlDCSeoc30SShRbwSEDfqcx2ji3Kl7C1ONqFedI1jVg2XHcwGwoLBYKFUNxjKCgKO76wfdgxYnE9B631XjQvV43m\n"

"YhAOzED7YYoPYOKxAbrSxkLe10qmAcPX8UfiCltPzRav4gV5k23dfGpNuI7lzTizpHBJVgOfAyAjXSNoYaTXFmoJNa\n"

"HbPDbbACS4PMnZEvTdtXPpeGwRtTk14uW3of5EjgfmN0EIzCL9jTMt2N3ZJJSSjclMbzJwJ2AXE6qLIWJWUncaLV8G\n"

"du041dCRPagLwl6cu0KdYA0UW6ilkTp1JD5dIHD7FvUjmyKgf5k1Zxotq9e6ggjRKHTPoylCxL44HNt0obfRVK5rzr\n"

"rXEWfpxs8Ln3OgMpody4BnO57hRhq4TG2hUaL9FNgQe64E3nbQbTN0K2moA2l4U1s0SgFsZm7p7p3w7l5EnkPWDIs1\n"

"j4qBi9lzC9CIRy9nHNt7SkbiUBMyWwXeSvblcccuOoht5SoQPAYbHWf9kIUtThVlReTDSJ7KVQ55Ys1PnK0amFEsE6\n"

"1FsjyBS1z3BnYnxcRySApMmODrUDsrPy3MUxKJVjqyBScPTkwbvQVNiP9Sp1RBbMkyRFQ6lW5r8pCf37KtNcGmRmI5\n"

"m4LsS2FIFzVi70bm5f2SHLC3AOIHdOFtAK63jm9Ten2oeFKgmeHNT7eONBIYLrK8MsfCu2XwFQbsq4XsuN8ywzUb5P\n"

"tsRYIF9iedELMFOCd7yLQjyjKBztoA76Ufl8m4J8lY6nHDygEwYGHBAovZLWjiapWWCzc3nlvvOsB08mqgTEWOUheV\n"

"r1MqJ2pUwkQeHg3372eHj9QfPb8tjGnEMgzVlolFESEATmT9WhnZcsFVqGqbgltzy2znpOW17c4Cskj8Qa2m0KnyeB\n"

"yfIfDyE63VXyH5qBnNzi1NbRQkiVamM2uk3PnJdLdJLAQUE7MD0vp4WnvX99Mc9ytR5d0UweQj1q1RpaaP3cyjq4Z9\n"

"2QmaYE0xvrMRd4VdZIhd2J0kNQmLNpMy6qi0OeNGjaEsNHFHAfH2IEaRCtrNJhrKHX2HNFRU4mEFUzveI1bT0qBg2v\n"

"frTHpV5oxqOQqFoDf4mJ8F3TdNsLd8xbSmZExKRNoc6MmHgbTYPo2SXPLNYYlO7fGmuR3lekivPQBd43fsnPFRV0yh\n"

"DrWcEhs2X70WUgRDxSrFflhGavZRSPF6PS1MiXbD2AM9WwFWZFv50GJcJDx2ehX3bAbs12WEr0QOBjgTgLll9x1W2T\n"

"NpetFM4oJn4kb187wScue8Xb9Zbs39PUutlckAVxVkCheYBI2mJpO5ugjSj5cypNKJQF9K6zP4jaTmUmngaDXOOYOV\n"

"mPx10Y72IsFV5DrWrEurtTe2NZA3qdKQn5hoy72OoABbKCL8ROSxV4PIbAPY3jf2Ti02zLsMJQ7svTOUDjIb abbbd\n", "zyyyl"); // 70 lines, 90 characters each line