// David Nakasone lab12 v2 .... had to improve the encryption on this

#include <iostream>

#include <string>

#include <cstring>

#include <iomanip>

#include <fstream>

#include <cassert>

#include <cmath>

#include <cstdlib>

#include <cctype>

#include <sstream>

using namespace std;

// clang++ -std=c++14 -Weverything main.cpp

// ./a.out < originalASCII.txt

const int ROWS = 200; // can be up to characters (rows) in message ...same for expansion and original

const int RAWcols = 7; // each column is in basic ASCII, so it will be an 7 didgit binary number

const int EXPcols = 16; // the columns will widen from 7 to 16, maybe more depending on how this goes

const int KEY = 1957 + pow(2,15); // use this to unlock the message = 34725

void initializeRaw(int matrix[][RAWcols]); // fill array with 0's

void initializeExp(int matrix[][EXPcols]); // fill in expanded array with 0's

void inputRaw(int matrix[][RAWcols]); // user input goes here, fills an array[200][7]

void passASCII(char character, int ascii[RAWcols]); // used to turn an arry[7] to decimal ASCII integer

void inputExp(int matrix[][RAWcols], int table[][EXPcols]); // transfers input to expanded array[200][16]

int findChar(const int list[EXPcols]); // converts binary to decimal, representing character

void row16decTObinary(int toReturn[EXPcols], int decimal); // decimal returned in binary array[16]

void encrypt(int toEncrypt[][EXPcols], const int original[][EXPcols]); // encrypt expanded matrix, up to user for method

void decrypt(const int encrypted[][EXPcols], int toDecrypt[][EXPcols]); // uses key/algo/LUT to unlock encrypted array

void printRaw(int matrix[][RAWcols]); // test print array [200][7] if needed

void printExp(int matrix[][EXPcols]); // test print array [200][16] if needed

//-------------------------------------------------------------------------------------------------------------------

int main () {

int raw[ROWS][RAWcols];

int expanded[ROWS][EXPcols];

int expandedEncrpyt[ROWS][EXPcols];

int decrpyt[ROWS][EXPcols];

initializeRaw(raw); // fill in with 0's

initializeExp(expanded); // fill in with 0's

initializeExp(expandedEncrpyt); // fill in with 0's

initializeExp(decrpyt); // fill in with 0's

inputRaw(raw); // get user input, using a original.txt in this case

inputExp(raw, expanded); // transfer raw input to larger matrix

encrypt(expandedEncrpyt, expanded); // encrypt expanded array in a new array

decrypt(expandedEncrpyt, decrpyt); // uses the key/algorithm/LUT to unlock

//printRaw(raw); // test print

//printExp(expanded); // test print

// see "TOdecrypt.txt" it should match "original.txt"

cout << "looks good, check the .txt files" << endl;

return 0;

} // end main()

//-------------------------------------------------------------------------------------------------------------------

void initializeRaw(int matrix[][RAWcols]){ // fills all elements of a [200][7] array with 0

for (int i = 0; i < ROWS; i++) {

for (int j = 0; j < RAWcols; j++) {

matrix[i][j] = 0;

} // end for

} // end for

} // end initializeRaw()

//-------------------------------------------------------------------------------------------------------------------

void initializeExp(int matrix[][EXPcols]) { // fills all elements of a [200][16] array with 0

for (int i = 0; i < ROWS; i++) {

for (int j = 0; j < EXPcols; j++) {

matrix[i][j] = 0;

} // end for

} // end for

} // end initializeExp()

//-------------------------------------------------------------------------------------------------------------------

void inputRaw(int matrix[][RAWcols]) { // gets input from "original.txt"

ifstream inFile; // outputs results to "TOasciiRaw.txt"

ofstream outFile;

inFile.open("original.txt"); // change as needed

outFile.open("TOasciiRaw.txt"); // change as needed

char character = 'a'; // current character extracted from the input stream

int ascii[RAWcols] = {0,0,0,0,0,0,0}; // initialized all 7 didgits to 0

int j = 0;

for (int i = 0; i < ROWS; i++) {

for (j = 0; j < RAWcols; j++) {

character = inFile.get();

if (character > 0 && character <= 127 && character != 10) {

outFile << left << setw(6) << static\_cast<int>(character) << left << setw(4) << character << " : ";

passASCII(character, ascii);

for (int k = 0; k < RAWcols; k++) {

matrix[i][k] = ascii[k];

outFile << matrix[i][k];

} // end for

} // end if

else if (character == 10) {

outFile << left << setw(6) << 10 << left << setw(4) << "<>" << " : ";

passASCII(character, ascii);

for (int l = 0; l < RAWcols; l++) {

matrix[i][l] = ascii[l];

outFile << matrix[i][l];

} // end for

} // end else if

else {

outFile << left << setw(6) << 0 << left << setw(4) << "NUL" << " : ";

for (int m = 0; m < RAWcols; m++) {

matrix[i][m] = 0;

outFile << matrix[i][m];

} // end for

} // end esle

j = RAWcols; // break out

} // end for

if (i < ROWS-1) outFile << endl;

} // end for

inFile.close();

outFile.close();

} // end inputRaw()

//-------------------------------------------------------------------------------------------------------------------

void passASCII(char character, int ascii[RAWcols]) { // extracted character and array passed

int number = character; // the character is converted to an decimal integer

for (int i = RAWcols-1; i >= 0; i--) { // the decimal intger is put into binary and stored in array

ascii[i] = number % 2;

number = number / 2;

} // end for

} // end passACII

//-------------------------------------------------------------------------------------------------------------------

void inputExp(int matrix[][RAWcols], int table[][EXPcols]){ // [200][7] array placed into a [200][16] array

ofstream outFile; // results can be seen in "TOasciiExp.txt"

int binary[EXPcols] ={0}; // holds binary of the character

int y = 0; // LCV

char character ='a';

//ofstream outFile;

outFile.open("TOasciiExp.txt");

for (int i = 0; i < ROWS; i++) {

for (int j = 0; j < EXPcols; j++) {

if (j > 8) {

table [i][j] = matrix[i][j-9];

} // end if

else{

table[i][j] = 0;

} // end else

} // end for

} // end for

for (int x = 0; x < ROWS; x++) {

for (y = 0; y < EXPcols; y++) {

binary[y] = table[x][y];

} // end for

character = findChar(binary);

if (character > 0 && character <= 127 && character != 10) {

outFile << left << setw(6) << static\_cast<int>(character) << left << setw(4) << character << " : ";

for (int k = 0; k < EXPcols; k++) {

outFile << binary[k];

//outFile << binary[k];

} // end for

} // end if

else if (character == 10) {

outFile << left << setw(6) << 10 << left << setw(4) << "<>" << " : ";

for (int l = 0; l < EXPcols; l++) {

outFile << binary[l];

} // end for

} // end else if

else {

outFile << left << setw(6) << 0 << left << setw(4) << "NUL" << " : ";

for (int m = 0; m < EXPcols; m++) {

outFile << binary[m];

} // end for

} // end esle

if (x < ROWS-1) outFile << endl;

} // end for

outFile.close();

} // end inputExp()

//-------------------------------------------------------------------------------------------------------------------

int findChar(const int list[EXPcols]){ // simple way to get decimal number from array[16]

int decimal = 0; // MSB is in array[0], LSB in array[15]

for (int i = EXPcols - 1; i >= 0; i--){ // \* a char can never be > 127 with std ASCII

decimal = decimal + (list[i] \* (pow(2,(15-i)))); // this allows function to double as a regular converter

} // end for

return decimal;

} // end findChar()

//-------------------------------------------------------------------------------------------------------------------

void row16decTObinary(int toReturn[EXPcols], int decimal){ // give it a decimal, and it puts it in a 16-bit binary array

// returned by reference

for (int i = 0; i < EXPcols; i++) { // clear toReturn

toReturn[i] = 0;

} // end for

for (int i = EXPcols - 1; i >= 0; i--) {

toReturn[i] = decimal % 2;

decimal = decimal / 2;

} // end for

} // end row16decTObinary()

//-------------------------------------------------------------------------------------------------------------------

void encrypt(int toEncrypt[][EXPcols], const int original[][EXPcols]){ // make any method as long as you can decrpyt

int holder[EXPcols]; // holds character on current row in binary // this way adds 1 to MSB of 16-bit binary representing

int holderPass[ROWS]; // holds all 200 characters in decimal // the character

int binTOdec; // decimal result of 16-bit binary number // obviouslly the decrypt is: decimal - {KEY}

ofstream outFile; // keep in mind the MSB is @ array[i][0], not [i][15]

outFile.open("TOcrypt.txt"); // see "TOcrypt.txt" for the results

for (int i = 0; i < EXPcols; i++) { // set all elements of holder[16] to 0

holder[i] = 0;

} // end for

for (int i = 0; i < ROWS; i++) { // set all elements of holderPass[200] to 0

holderPass[i] = 0;

} // end for

binTOdec = 0; // set to 0

// extract deciaml representation of row

for (int i = 0; i < ROWS; i++) {

for (int j = 0; j < EXPcols; j++){

holder[j] = original[i][j]; // pass 16-bit binary character to holder

} // end for

binTOdec = findChar(holder); // puts decimal of row into binTOdec

holderPass[i] = binTOdec; // stores deciaml of row into holderPass[100]

} // end for

// now add the KEY to decimal of each row

for (int i = 0; i < ROWS; i++) {

holderPass[i] = holderPass[i] + KEY;

} // end for

// take decimal of each row and put into binary

for (int i = 0; i < ROWS; i++) {

row16decTObinary(holder,holderPass[i]); // holder[16] will be filled with binary of decimal @ holderPass[i]

for (int j = 0; j < EXPcols; j++) {

toEncrypt[i][j] = holder[j]; // encrpyts with bit at current position

} // end for

} // end for

// displays results

for (int i = 0; i < ROWS; i++){

for (int j = 0; j < EXPcols; j++) {

holder[j] = toEncrypt[i][j];

} // end for

binTOdec = findChar(holder); // gets result

outFile << left << setw(8) << binTOdec << "??? " << " : ";

for (int k = 0; k < EXPcols; k++){

outFile << holder[k];

} // end for

if(i < ROWS -1) outFile << endl;

} // end for

outFile.close();

} // end encrpyt()

//-------------------------------------------------------------------------------------------------------------------

void decrypt(const int encrypted[][EXPcols], int toDecrypt[][EXPcols]){ // insert decryption method here

ofstream outFile; // remember strings are null terminated

outFile.open("TOdecrypt.txt"); // so use a cString or char array

int cipher[16]; // see "TOdecrypt.txt"

char current; // this should match "original.txt"

int pikPin[ROWS];

int handShake;

int y;

// initialize ...they should be initialized to 0

// decrpyt "encrypted[][EXPcols]" to "toDecrypt[][EXPcols]"

for (int i = 0; i < ROWS; i++) {

for (int j = 0; j < EXPcols; j++) {

cipher[j] = encrypted[i][j]; // get current row in binary

} // end for

handShake = findChar(cipher); // turn binary to decimal

pikPin[i] = handShake;

pikPin[i] = pikPin[i] - KEY; // it is decrypted now

} // end for

for (int i = 0; i < ROWS; i++) {

row16decTObinary(cipher, pikPin[i]);

for (int j = 0; j < EXPcols; j++) {

toDecrypt[i][j] = cipher[j];

} // end for

} // end for

// just print now print to "TOdecrypt.txt" make sure it matches original.txt

for (int x = 0; x < ROWS; x++) {

for (y = 0; y < EXPcols; y++) {

cipher[y] = toDecrypt[x][y];

} // end for

handShake = findChar(cipher);

current = handShake;

if (current > 0) outFile << current;

} // end for

outFile.close();

} // end decrypt()

//-------------------------------------------------------------------------------------------------------------------

void printRaw(int matrix[][RAWcols]) {

for (int i = 0; i < ROWS; i++) {

for (int j = 0; j < RAWcols; j++) {

cout << matrix[i][j];

} // end for

cout << endl;

} // end for

} // end printRaw()

//-------------------------------------------------------------------------------------------------------------------

void printExp(int matrix[][EXPcols]){

for (int i = 0; i < ROWS; i++) {

for (int j = 0; j < EXPcols; j++) {

cout << matrix[i][j];

} // end for

cout << endl;

} // end for

} // end printExp

// GRAVE YARD , most of this is not worth resurrecting.

/\*

//ifstream inFile;

//inFile.open("originalASCII.txt");

int extractor = 9; // holds current integer from user input...don't see a 1 or 0, there is a problem

int i = 0; // row index

int j = 0; // col index

while (i < 2) {

while (j < RAWcols) {

cin >> extractor;

//inFile >> extractor;

if (!cin) matrix[i][j] = 0;

else matrix[i][j] = extractor;

j++; // inc

} // end while

i++; // inc

j = 0; // rst

} // end while

//inFile.close()

while (i < ROWS && !inFile.eof()) {

while (j < RAWcols && !inFile.eof()) {

character = inFile.get();

if (!inFile.eof() && (character >= 0 && character <= 127) ) {

//outFile << left << setw(6) << character;

passASCII(character, ascii);

for (int k = 0; k < RAWcols; k++) {

matrix[i][k] = ascii[k];

//outFile << matrix[i][k];

} // end for

//outFile << endl;

j = j + RAWcols; // breaks out

} // end if

else { i = ROWS;}

} // end while

i++; // inc

j = 0; // rst

} // end while

for (int x = 0; x < ROWS; x++) {

for (int y = 0; y < RAWcols; y++){

outFile << matrix[x][y];

} // end for

outFile << endl;

} // end for

while (i < ROWS && !inFile.eof()) {

while (j < RAWcols && !inFile.eof()) {

character = (!inFile.eof()) ? inFile.get() : 0;

if (!inFile.eof() && (character >= 0 && character <= 127) ) {

outFile << left << setw(6) << static\_cast<int>(character) << left << setw(4) << character << " : ";

passASCII(character, ascii);

for (int k = 0; k < RAWcols; k++) {

matrix[i][k] = ascii[k];

outFile << matrix[i][k];

} // end for

outFile << endl;

j = j + RAWcols; // breaks out

} // end if

else {

outFile << left << setw(6) << "NULL" << left << setw(4) << 0 << " : ";

for (int x = 0; x < RAWcols; x++) {

matrix[i][x] = 0;

outFile << matrix[i][x];

} // end for

} // end else

} // end while

i++; // inc

j = 0; // rst

} // end while

for(int i = 0; i < ROWS; i++){

for(int j = 0; j < RAWcols; j++){

character = (!inFile) ? inFile.get() : 0;

if (character != 0) {

} // end if

else {

outFile << left << setw(6) << "NULL" << left << setw(4) << 0 << " : ";

for (int l = 0; l < RAWcols; l++) {

matrix[i][l] = 0;

outFile << matrix[i][l];

} // end else

}// end for

if (i < (ROWS - 1)) outFile << endl;

} // end for

\*/