**Assignment 1**

**1.**

// Ceaser Cipher

#include <bits/stdc++.h>

using namespace std;

string encrypt(string str, int key)

{

    int n = str.size();

    string temp;

    for(int i = 0; i < n; i++)

    {

        if(str[i] == ' ')

        {

            temp += ' ';

        }

        else

        {

            char ch = (str[i] + key);

            temp += ch;

        }

    }

    return temp;

}

string decrypt(string str, int key)

{

    int n = str.size();

    string temp;

    for(int i = 0; i < n; i++)

    {

        if(str[i] == ' ')

        {

            temp += ' ';

        }

        else

        {

            char ch = (str[i] - key);

            temp += ch;

        }

    }

    return temp;

}

int main()

{

    int shiftKey;

    cout << "Enter the shift key value: ";

    cin >> shiftKey;

    cin.ignore(); // to clear the newline character from the buffer

    string str;

    cout << "Enter the string to be encrypted: ";

    getline(cin, str);

    string newStr = encrypt(str, shiftKey);

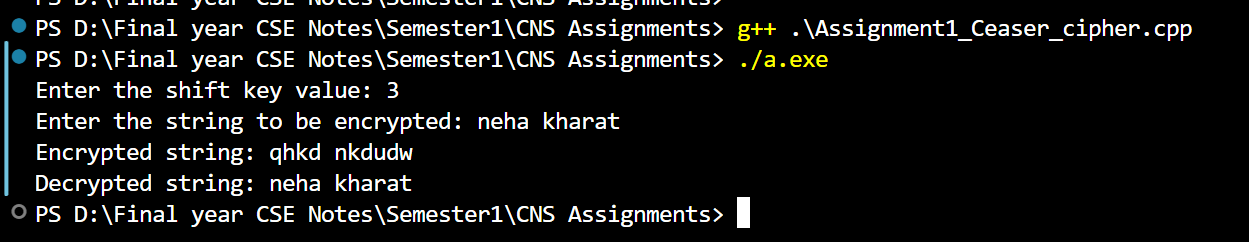
    cout << "Encrypted string: " << newStr << endl;

    string newStr1 = decrypt(newStr, shiftKey);

    cout << "Decrypted string: " << newStr1 << endl;

    return 0;

}

****

**2.**

// C++ code to implement Hill Cipher

#include <iostream>

using namespace std;

// Following function generates the

// key matrix for the key string

void getKeyMatrix(string key, int keyMatrix[][3])

{

    int k = 0;

    for (int i = 0; i < 3; i++)

    {

        for (int j = 0; j < 3; j++)

        {

            keyMatrix[i][j] = (key[k]) % 65;

            k++;

        }

    }

}

// Following function encrypts the message

void encrypt(int cipherMatrix[][1],

            int keyMatrix[][3],

            int messageVector[][1])

{

    int x, i, j;

    for (i = 0; i < 3; i++)

    {

        for (j = 0; j < 1; j++)

        {

            cipherMatrix[i][j] = 0;

            for (x = 0; x < 3; x++)

            {

                cipherMatrix[i][j] +=

                    keyMatrix[i][x] \* messageVector[x][j];

            }

            cipherMatrix[i][j] = cipherMatrix[i][j] % 26;

        }

    }

}

// Function to implement Hill Cipher

void HillCipher(string message, string key)

{

    // Get key matrix from the key string

    int keyMatrix[3][3];

    getKeyMatrix(key, keyMatrix);

    int messageVector[3][1];

    // Generate vector for the message

    for (int i = 0; i < 3; i++)

        messageVector[i][0] = (message[i]) % 65;

    int cipherMatrix[3][1];

    // Following function generates

    // the encrypted vector

    encrypt(cipherMatrix, keyMatrix, messageVector);

    string CipherText;

    // Generate the encrypted text from

    // the encrypted vector

    for (int i = 0; i < 3; i++)

        CipherText += cipherMatrix[i][0] + 65;

    // Finally print the ciphertext

    cout << " Ciphertext:" << CipherText;

}

// Driver function for above code

int main()

{

    // Get the message to be encrypted

    string message = "ACT";

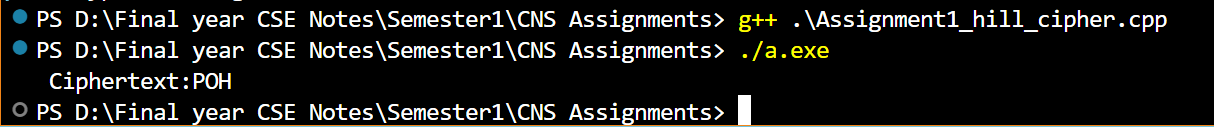
    // Get the key

    string key = "GYBNQKURP";

    HillCipher(message, key);

    return 0;

}

****

**3.**

// C++ program to implement Playfair Cipher

#include <bits/stdc++.h>

using namespace std;

#define SIZE 30

// Function to convert the string to lowercase

void toLowerCase(char plain[], int ps)

{

    int i;

    for (i = 0; i < ps; i++) {

        if (plain[i] > 64 && plain[i] < 91)

            plain[i] += 32;

    }

}

// Function to remove all spaces in a string

int removeSpaces(char\* plain, int ps)

{

    int i, count = 0;

    for (i = 0; i < ps; i++)

        if (plain[i] != ' ')

            plain[count++] = plain[i];

    plain[count] = '\0';

    return count;

}

// Function to generate the 5x5 key square

void generateKeyTable(char key[], int ks, char keyT[5][5])

{

    int i, j, k, flag = 0;

    // a 26 character hashmap

    // to store count of the alphabet

    int dicty[26] = { 0 };

    for (i = 0; i < ks; i++) {

        if (key[i] != 'j')

            dicty[key[i] - 97] = 2;

    }

    dicty['j' - 97] = 1;

    i = 0;

    j = 0;

    // Placing characters of key string in the key table

    for (k = 0; k < ks; k++) {

        if (dicty[key[k] - 97] == 2) {

            dicty[key[k] - 97] -= 1;

            keyT[i][j] = key[k];

            j++;

            if (j == 5) {

                i++;

                j = 0;

            }

        }

    }

    // PLacing remaining charachters

    for (k = 0; k < 26; k++) {

        if (dicty[k] == 0) {

            keyT[i][j] = (char)(k + 97);

            j++;

            if (j == 5) {

                i++;

                j = 0;

            }

        }

    }

}

// Function to search for the characters of a digraph

// in the key square and return their position

void search(char keyT[5][5], char a, char b, int arr[])

{

    int i, j;

    if (a == 'j')

        a = 'i';

    else if (b == 'j')

        b = 'i';

    for (i = 0; i < 5; i++) {

        for (j = 0; j < 5; j++) {

            if (keyT[i][j] == a) {

                arr[0] = i;

                arr[1] = j;

            }

            else if (keyT[i][j] == b) {

                arr[2] = i;

                arr[3] = j;

            }

        }

    }

}

// Function to find the modulus with 5

int mod5(int a) { return (a % 5); }

// Function to make the plain text length to be even

int prepare(char str[], int ptrs)

{

    if (ptrs % 2 != 0) {

        str[ptrs++] = 'z';

        str[ptrs] = '\0';

    }

    return ptrs;

}

// Function for performing the encryption

void encrypt(char str[], char keyT[5][5], int ps)

{

    int i, a[4];

    for (i = 0; i < ps; i += 2) {

        search(keyT, str[i], str[i + 1], a);

        // if both the characters are in the same row

        if (a[0] == a[2]) {

            str[i] = keyT[a[0]][mod5(a[1] + 1)];

            str[i + 1] = keyT[a[0]][mod5(a[3] + 1)];

        }

        // if they are in the same column

        else if (a[1] == a[3]) {

            str[i] = keyT[mod5(a[0] + 1)][a[1]];

            str[i + 1] = keyT[mod5(a[2] + 1)][a[1]];

        }

        // they form a rectangle. i.e. if they are neither in the same nor in the same col

        else {

            str[i] = keyT[a[0]][a[3]];

            str[i + 1] = keyT[a[2]][a[1]];

        }

    }

}

// Function to encrypt using Playfair Cipher

void encryptByPlayfairCipher(char str[], char key[])

{

    int ps, ks;

    char keyT[5][5];

    // Key

    ks = strlen(key);

    ks = removeSpaces(key, ks);

    toLowerCase(key, ks);

    // Plaintext

    ps = strlen(str);

    toLowerCase(str, ps);

    ps = removeSpaces(str, ps);

    ps = prepare(str, ps);

    generateKeyTable(key, ks, keyT);

    encrypt(str, keyT, ps);

}

// Function to decrypt

void decrypt(char str[], char keyT[5][5], int ps)

{

    int i, a[4];

    for (i = 0; i < ps; i += 2) {

        search(keyT, str[i], str[i + 1], a);

        if (a[0] == a[2]) {

            str[i] = keyT[a[0]][mod5(a[1] - 1)];

            str[i + 1] = keyT[a[0]][mod5(a[3] - 1)];

        }

        else if (a[1] == a[3]) {

            str[i] = keyT[mod5(a[0] - 1)][a[1]];

            str[i + 1] = keyT[mod5(a[2] - 1)][a[1]];

        }

        else {

            str[i] = keyT[a[0]][a[3]];

            str[i + 1] = keyT[a[2]][a[1]];

        }

    }

}

// Function to call decrypt

void decryptByPlayfairCipher(char str[], char key[])

{

    char ps, ks, keyT[5][5];

    // Key

    ks = strlen(key);

    ks = removeSpaces(key, ks);

    toLowerCase(key, ks);

    // ciphertext

    ps = strlen(str);

    toLowerCase(str, ps);

    ps = removeSpaces(str, ps);

    generateKeyTable(key, ks, keyT);

    decrypt(str, keyT, ps);

}

// Driver code

int main()

{

    char str[SIZE], key[SIZE];

    // Key to be encrypted

    strcpy(key, "Monarchy");

    cout << "Key text: " << key << "\n";

    // Plaintext to be encrypted

    strcpy(str, "instruments");

    cout << "Plain text: " << str << "\n";

    // encrypt using Playfair Cipher

    encryptByPlayfairCipher(str, key);

    cout << "Cipher text: " << str << "\n";

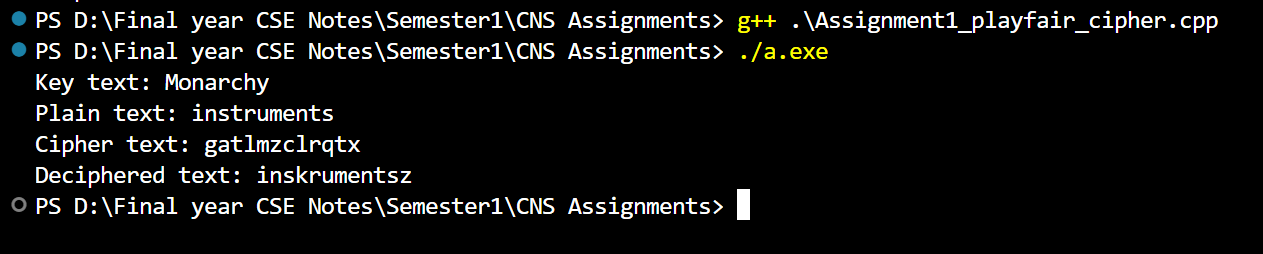
    // encrypt using Playfair Cipher

    decryptByPlayfairCipher(str, key);

    cout << "Deciphered text: " << str << endl;

    return 0;

}

****

**4.**

// C++ code to implement Vigenere Cipher

#include <bits/stdc++.h>

using namespace std;

// This function generates the key in

// a cyclic manner until it's length isn't

// equal to the length of original text

string generateKey(string str, string key)

{

    int x = str.size();

    for (int i = 0;; i++) {

        if (x == i)

            i = 0;

        if (key.size() == str.size())

            break;

        key.push\_back(key[i]);

    }

    return key;

}

// This function returns the encrypted text

// generated with the help of the key

string cipherText(string str, string key)

{

    string cipher\_text;

    for (int i = 0; i < str.size(); i++) {

        // converting in range 0-25

        char x = (str[i] + key[i]) % 26;

        // convert into alphabets(ASCII)

        x += 'A';

        cipher\_text.push\_back(x);

    }

    return cipher\_text;

}

// This function decrypts the encrypted text

// and returns the original text

string originalText(string cipher\_text, string key)

{

    string orig\_text;

    for (int i = 0; i < cipher\_text.size(); i++) {

        // converting in range 0-25

        char x = (cipher\_text[i] - key[i] + 26) % 26;

        // convert into alphabets(ASCII)

        x += 'A';

        orig\_text.push\_back(x);

    }

    return orig\_text;

}

// Driver program to test the above function

int main()

{

    string str = "GEEKSFORGEEKS";

    string keyword = "AYUSH";

    if (any\_of(str.begin(), str.end(), ::islower))

        transform(str.begin(), str.end(), str.begin(),

                  ::toupper);

    if (any\_of(keyword.begin(), keyword.end(), ::islower))

        transform(keyword.begin(), keyword.end(),

                  keyword.begin(), ::toupper);

    string key = generateKey(str, keyword);

    string cipher\_text = cipherText(str, key);

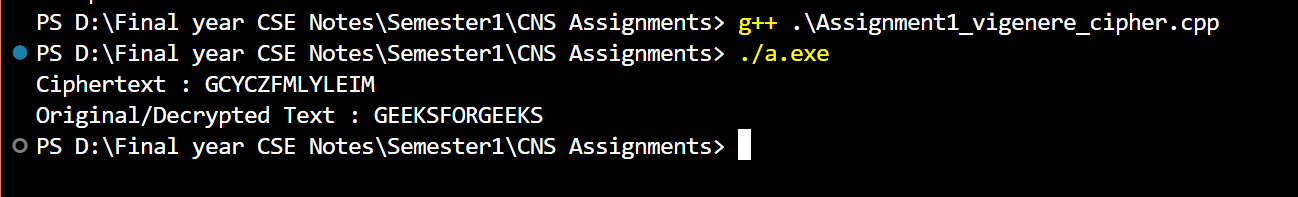
    cout << "Ciphertext : " << cipher\_text << "\n";

    cout << "Original/Decrypted Text : "

         << originalText(cipher\_text, key);

    return 0;

}

****

**Assignment 2:**

**1.**

#include <bits/stdc++.h>

using namespace std;

// function to encrypt a message

string encryptRailFence(string text, int key)

{

    char rail[key][(text.size())];

    for(int i=0;i<key;i++)

    {

        for(int j=0;j<text.size();j++)

        {

            rail[i][j] = '\n';

        }

    }

    bool dir\_down = false;

    int row=0, col=0;

    for(int i=0;i<text.length();i++)

    {

        if(row == 0 || row == key-1)

          dir\_down = !dir\_down;

        rail[row][col++] = text[i];

        dir\_down? row++ : row--;

    }

    string result;

    for(int i=0;i<key;i++)

    {

        for(int j=0;j<text.size();j++)

        {

            if(rail[i][j] != '\n')

               result.push\_back(rail[i][j]);

        }

    }

    return result;

}

// This function receives cipher-text and key

// and returns the original text after decryption

string decryptRailFence(string cipher, int key)

{

    // create the matrix to cipher plain text

    // key = rows , length(text) = columns

    char rail[key][cipher.length()];

    // filling the rail matrix to distinguish filled

    // spaces from blank ones

    for (int i=0; i < key; i++)

        for (int j=0; j < cipher.length(); j++)

            rail[i][j] = '\n';

    // to find the direction

    bool dir\_down;

    int row = 0, col = 0;

    // mark the places with '\*'

    for (int i=0; i < cipher.length(); i++)

    {

        // check the direction of flow

        if (row == 0)

            dir\_down = true;

        if (row == key-1)

            dir\_down = false;

        // place the marker

        rail[row][col++] = '\*';

        // find the next row using direction flag

        dir\_down?row++ : row--;

    }

    // now we can construct the fill the rail matrix

    int index = 0;

    for (int i=0; i<key; i++)

        for (int j=0; j<cipher.length(); j++)

            if (rail[i][j] == '\*' && index<cipher.length())

                rail[i][j] = cipher[index++];

    // now read the matrix in zig-zag manner to construct

    // the resultant text

    string result;

    row = 0, col = 0;

    for (int i=0; i< cipher.length(); i++)

    {

        // check the direction of flow

        if (row == 0)

            dir\_down = true;

        if (row == key-1)

            dir\_down = false;

        // place the marker

        if (rail[row][col] != '\*')

            result.push\_back(rail[row][col++]);

        // find the next row using direction flag

        dir\_down?row++: row--;

    }

    return result;

}

//driver program to check the above functions

int main()

{

    cout << encryptRailFence("attack at once", 2) << endl;

    cout << encryptRailFence("GeeksforGeeks ", 3) << endl;

    cout << encryptRailFence("defend the east wall", 3) << endl;

    //Now decryption of the same cipher-text

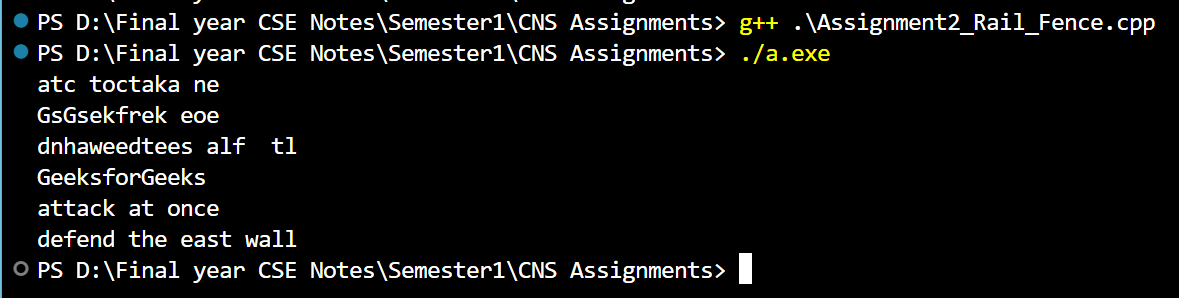
    cout << decryptRailFence("GsGsekfrek eoe",3) << endl;

    cout << decryptRailFence("atc toctaka ne",2) << endl;

    cout << decryptRailFence("dnhaweedtees alf  tl",3) << endl;

    return 0;

}

****

**2. Row\_column cipher**

#include <bits/stdc++.h>

using namespace std;

string enc(string str, string key)

{

    bool flg = false;

    int ind = 0;

    // Find the maximum digit in the key

    int mxnum = 0;

    for (int i = 0; i < key.size(); i++)

    {

        mxnum = max(mxnum, key[i] - '0');

    }

    // Calculate the number of rows required

    int nrow = ceil((double)str.size() / mxnum);

    vector<vector<char>> vec(nrow, vector<char>(mxnum, '$')); // Fill with space instead of '0'

    for (int i = 0; i < nrow; i++)

    {

        for (int j = 0; j < mxnum; j++)

        {

            if (ind >= str.size())

            {

                flg = true;

                break;

            }

            vec[i][j] = str[ind++];

        }

        if (flg == true)

            break;

    }

    cout << "Matrix : \n";

    for (int i = 0; i < vec.size(); i++)

    {

        for (int j = 0; j < vec[i].size(); j++)

        {

            cout << vec[i][j] << " ";

        }

        cout << "\n";

    }

    string res;

    for (int k = 0; k < key.size(); k++)

    {

        int col = key[k] - '1'; // Adjust for 0-based index

        for (int row = 0; row < nrow; row++)

        {

            if(vec[row][col] != '$') // Avoid adding spaces from the matrix

                res += vec[row][col];

        }

    }

    return res;

}

string dec(string str, string key)

{

    bool flg = false;

    int ind = 0;

    // Find the maximum digit in the key

    int mxnum = 0;

    for (int i = 0; i < key.size(); i++)

    {

        mxnum = max(mxnum, key[i] - '0');

    }

    // Calculate the number of rows required

    int nrow = ceil((double)str.size() / mxnum);

    vector<vector<char>> vec(nrow, vector<char>(mxnum, '$')); // Fill with '$' instead of '0'

    for(int i=0; i<key.size(); i++)

    {

        int col = key[i] - '1';

        for(int row=0; row<nrow; row++)

        {

            if(ind >= str.size())

            {

                flg = true;

                break;

            }

            vec[row][col] = str[ind++];

        }

        if(flg == true)

          break;

    }

    string res;

    for(int i=0; i<vec.size(); i++)

    {

        for(int j=0; j<vec[i].size(); j++)

        {

            if(vec[i][j] != '$')

              res += vec[i][j];

        }

    }

    return res;

}

int main()

{

    string s;

    cout << "Input Text: " << endl;

    getline(cin, s);

    string key;

    cout << "Input Key: " << endl;

    cin >> key;

    string ciphertext = enc(s, key);

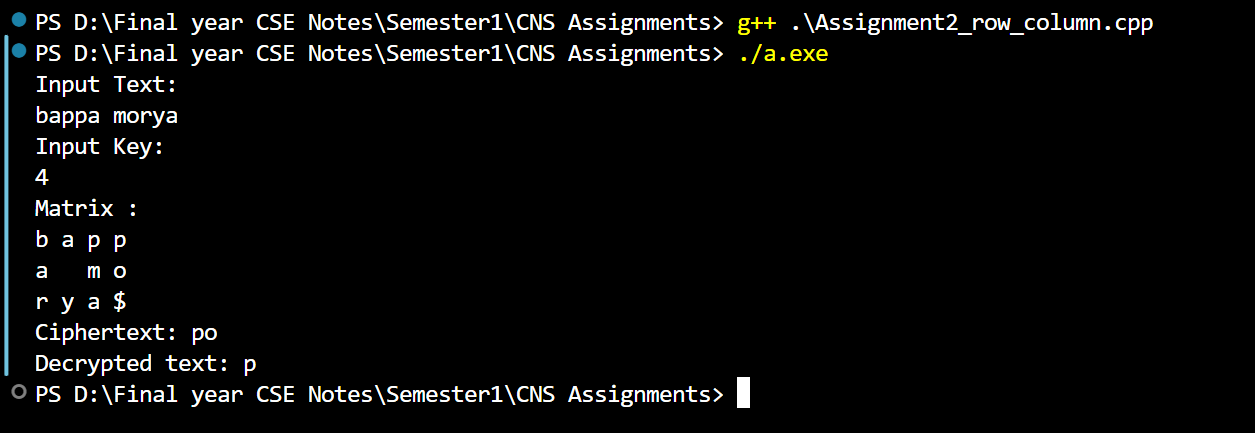
    cout << "Ciphertext: " << ciphertext << endl;

    string decryptedtext = dec(ciphertext, key);

    cout << "Decrypted text: " << decryptedtext << endl;

    return 0;

}

****

**Assignment 3:**

* **If we subtract a smaller number from a larger one (we reduce a larger number), GCD doesn’t change. So if we keep subtracting repeatedly the larger of two, we end up with GCD.**
* **Now instead of subtraction, if we divide the larger number, the algorithm stops when we find the remainder 0.**

#include <iostream>

using namespace std;

// Function to implement the Euclidean algorithm

// int gcd(int a, int b) {

//     while (b != 0) {

//         int temp = b;

//         b = a % b;

//         a = temp;

//     }

//     return a;

// }

// Function to return

// gcd of a and b

int gcd(int a, int b)

{

    if (a == 0)

        return b;

    return gcd(b % a, a);

}

int main() {

    int num1, num2;

    cout << "Enter two integers: ";

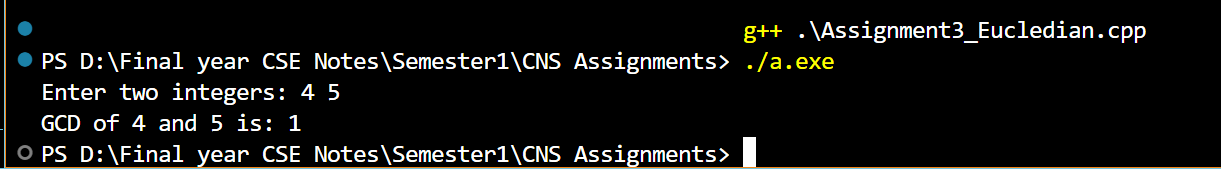
    cin >> num1 >> num2;

    int result = gcd(num1, num2);

    cout << "GCD of " << num1 << " and " << num2 << " is: " << result << endl;

    return 0;

}

****

**2.**

// C++ program to demonstrate working of

// extended Euclidean Algorithm

#include <bits/stdc++.h>

using namespace std;

// Function for extended Euclidean Algorithm

int gcdExtended(int a, int b, int \*x, int \*y)

{

    // Base Case

    if (a == 0)

    {

        \*x = 0;

        \*y = 1;

        return b;

    }

    int x1, y1; // To store results of recursive call

    int gcd = gcdExtended(b%a, a, &x1, &y1);

    // Update x and y using results of

    // recursive call

    \*x = y1 - (b/a) \* x1;

    \*y = x1;

    return gcd;

}

// Driver Code

int main()

{

    int x, y, a = 35, b = 15;

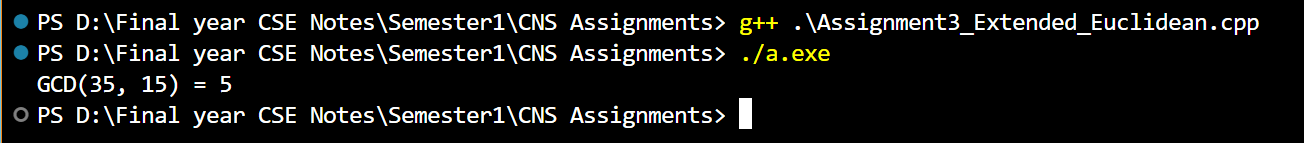
    int g = gcdExtended(a, b, &x, &y);

    cout << "GCD(" << a << ", " << b

         << ") = " << g << endl;

    return 0;

}

****

**Assignment 4:**

// A C++ program to demonstrate

// working of Chinese remainder

// Theorem

#include <bits/stdc++.h>

using namespace std;

// Returns modulo inverse of a

// with respect to m using

// extended Euclid Algorithm.

// Refer below post for details:

// https://www.geeksforgeeks.org/

// multiplicative-inverse-under-modulo-m/

int inv(int a, int m)

{

    int m0 = m, t, q;

    int x0 = 0, x1 = 1;

    if (m == 1)

        return 0;

    // Apply extended Euclid Algorithm

    while (a > 1) {

        // q is quotient

        q = a / m;

        t = m;

        // m is remainder now, process same as

        // euclid's algo

        m = a % m, a = t;

        t = x0;

        x0 = x1 - q \* x0;

        x1 = t;

    }

    // Make x1 positive

    if (x1 < 0)

        x1 += m0;

    return x1;

}

// k is size of num[] and rem[]. Returns the smallest

// number x such that:

// x % num[0] = rem[0],

// x % num[1] = rem[1],

// ..................

// x % num[k-2] = rem[k-1]

// Assumption: Numbers in num[] are pairwise coprime

// (gcd for every pair is 1)

int findMinX(int num[], int rem[], int k)

{

    // Compute product of all numbers

    int prod = 1;

    for (int i = 0; i < k; i++)

        prod \*= num[i];

    // Initialize result

    int result = 0;

    // Apply above formula

    for (int i = 0; i < k; i++) {

        int pp = prod / num[i];

        result += rem[i] \* inv(pp, num[i]) \* pp;

    }

    return result % prod;

}

// Driver method

int main(void)

{

    int num[] = { 3, 4, 5 };

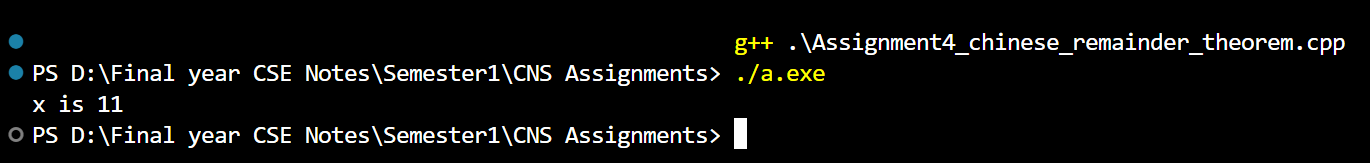
    int rem[] = { 2, 3, 1 };

    int k = sizeof(num) / sizeof(num[0]);

    cout << "x is " << findMinX(num, rem, k);

    return 0;

}

****

**Assignment 5:**

from Crypto.Cipher import DES

from Crypto.Util.Padding import pad, unpad

from Crypto.Random import get\_random\_bytes

def DES\_encrypt(plain\_text, key):

    key = key.ljust(8)[:8]

    iv = get\_random\_bytes(DES.block\_size)

    cipher\_encrypt = DES.new(key.encode(), DES.MODE\_CBC, iv)

    padded\_text = pad(plain\_text.encode(), DES.block\_size)

    encrypted\_message = cipher\_encrypt.encrypt(padded\_text)

    return iv + encrypted\_message

def DES\_decrypt(cipher\_text, key):

    key = key.ljust(8)[:8]

    iv = cipher\_text[:DES.block\_size]

    encrypted\_message = cipher\_text[DES.block\_size:]

    cipher\_decrypt = DES.new(key.encode(), DES.MODE\_CBC, iv)

    decrypted\_message = unpad(cipher\_decrypt.decrypt(encrypted\_message), DES.block\_size)

    return decrypted\_message.decode()

if \_\_name\_\_ == "\_\_main\_\_":

    plain\_text = input("Enter the message to encrypt: ")

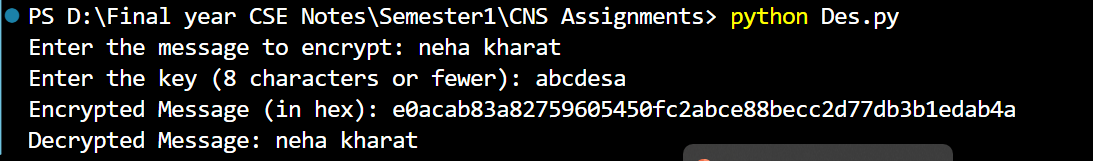
    key = input("Enter the key (8 characters or fewer): ")

    encrypted\_message = DES\_encrypt(plain\_text, key)

    print(f"Encrypted Message (in hex): {encrypted\_message.hex()}")

    decrypted\_message = DES\_decrypt(encrypted\_message, key)

    print(f"Decrypted Message: {decrypted\_message}")

****

**2. AES**

from Crypto.Cipher import AES

from Crypto.Random import get\_random\_bytes

def AES\_encrypt\_decrypt(message, key):

    key = key.ljust(16)[:16]

    iv = get\_random\_bytes(AES.block\_size)

    cipher\_encrypt = AES.new(key.encode(), AES.MODE\_CFB, iv)

    encrypted\_message = cipher\_encrypt.encrypt(message.encode())

    print(f"Encrypted Message (in hex): {encrypted\_message.hex()}")

    cipher\_decrypt = AES.new(key.encode(), AES.MODE\_CFB, iv)

    decrypted\_message = cipher\_decrypt.decrypt(encrypted\_message).decode()

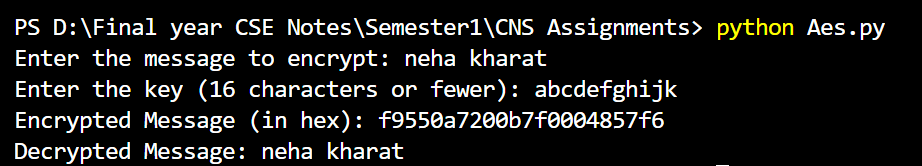
    print(f"Decrypted Message: {decrypted\_message}")

if \_\_name\_\_ == "\_\_main\_\_":

    message = input("Enter the message to encrypt: ")

    key = input("Enter the key (16 characters or fewer): ")

    AES\_encrypt\_decrypt(message, key)

****