

# PROGRAM 1

## ADDITIVE CIPHER

### Description:

In cryptography, an additive cipher, also known as Caesar's cipher, the shift cipher, Caesar's code or Caesar shift, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plain text is replaced by a letter some fixed number of positions down the alphabet.

For example, with a left shift of 1, D would be replaced by C, E would become D, and so on. The method is named after Julius Caesar.

### Program:

```
#include <bits/stdc++.h>
using namespace std;
static int key;

char encrypt(char c){
    //Encrypt a character given a key
    if (isdigit(c))
        return (c - '0' + key)%10 + '0';

    else if (isupper(c))
        return char((65 + (int(c) - 65 + key)%26 ));

    else return char((97 + (int(c) - 97 + key)%26 ));
}

void solve(vector<char> & input){
    //Decrypt the ciphered input when the key is given
    vector<char>:: iterator it = input.begin();
    vector<char> output;

    cout<<"\n The Decrypted Message: ";

    for(;it!=input.end();it++){
        cout<<*it;

        if (isalnum(*it))
            output.push_back(encrypt(*it));
        else
            output.push_back(*it);
    }
}
```

```
    }  
  
    vector<char>:: iterator ot = output.begin();  
    cout<<"\n The Encrypted Message: ";  
    for(;ot!=output.end();ot++)  
        cout<<*ot;  
}
```

```
int main(){  
  
    cout << "\n Enter Key(shift): \t";  
    cin >> key;  
  
    string s;  
    cout << "\n Enter input string: \t";  
    cin.ignore();  
    getline(cin, s);  
    vector<char>input (s.begin(), s.end());  
  
    solve(input);  
    cout<<endl<<endl;  
    return 0;  
}
```

### OUTPUT:

```
aman@aman ~/Desktop/prog/ISS/2-29 ADDITIVE_CIPHER $ ./caesar  
  
Enter Key(shift):      12  
  
Enter input string:    information and security systems  
  
The Decrypted Message: information and security systems  
The Encrypted Message: uzradymfuaz mzp eqogdufk ekefqye
```

**PROGRAM 2****MULTIPLICATIVE CIPHER****Description :**

It is a substitution cipher in which each letter in plain text is mapped to another letter in cipher text. The corresponding replacement of the letter is found out using  $C_i = (P_i * K) \bmod 26$

where C corresponds to the cipher text of Plain text P. But for multiplicative cipher, key K should be co prime to 26 because then only inverse of K will exist and the corresponding cipher text can be decrypted to plain text.

$$P_i = (C_i * K^{-1}) \bmod 26$$

**Program :**

```
#include <bits/stdc++.h>
using namespace std;
static int key, keyInv;
#define SIZE 26

// A program to implement multiplicative cipher

int gcd(int a, int b){
    // Given two numbers, finds GCD using Euclidean Method
    if (a < b)
        gcd (b, a);
    if (b == 0)
        return a;
    return gcd(b, a%b);
}

tuple<int, int> extendedGCDUtil(int a, int b, int s1, int s2, int t1, int t2){
    // Util method to find two solutions s1, t1
    if (b == 0)
        return make_tuple(s1, t1);
    return extendedGCDUtil(b, a % b, s2, s1 - (a / b) * s2, t2, t1 - (a / b) * t2);
}

tuple<int, int> extendedGCD(int a, int b){
    // Returns a pair of integers r and s such that a *r + b *s = 1
    // Solutions exist only when gcd(a, b) == 1
    // Let tie (r, s) = extendedGCD(divs[i], mods[i]); then (r + mods[i]) % mods[i] is the mod inv
    if (gcd (a, b) != 1)
        return make_tuple(0,0);
```

```
    return extendedGCDUtil(a, b, 1, 0, 0, 1);  
}
```

```
char encrypt(char c, int key){  
    //Encrypt a character given a key  
    if (isdigit(c))  
        return (c - '0' * key) % 10 + '0';  
  
    else if (isupper(c))  
        return char((65 + ((c - 65) * key) % 26));  
  
    else return char((97 + ((c - 97) * key) % 26));  
  
}
```

```
void solve(string & input){  
    //Decrypt the ciphered input when the key is given  
    string output;  
  
    for(auto a : input){  
        if (isalnum(a))  
            output += encrypt(a, key);  
        else  
            output += a;  
        // cout << a << " conv to: " << *output.rbegin() << endl;  
    }  
  
    cout<<"\n The Encrypted Message: ";  
    cout << output << endl;  
  
    /*Decrypt now*/  
    cout<<"\n The Decrypted Message: ";  
    for (auto a : output){  
        if (isalnum(a))  
            cout << encrypt(a, keyInv);  
        else  
            cout << a;  
    }  
  
}
```

```
int main(){  
  
    cout << "\n Enter Key(shift): \t";  
    cin >> key;
```

```
int b = 0;

while (gcd(key, SIZE) != 1){
    cout << "\n Key should be coprime to alphabet size";
    cout << "\n Enter Key(shift): \t";
    cin >> key;
}
tie(keyInv, b) = extendedGCD(key, SIZE);
keyInv = (keyInv + SIZE) % SIZE;

string s;
cout << "\n Enter input string: \t";
cin.ignore();
getline(cin, s);

solve(s);
cout<<endl<<endl;
return 0;
}
```

**OUTPUT:**

```
aman@aman ~/Desktop/prog/ISS/3-7 MULTI_CIPHER $ ./multi

Enter Key(shift):      2

Key should be coprime to alphabet size
Enter Key(shift):      3

Enter input string:    information and security systems

The Encrypted Message: ynpqzkafyqn anj cmgizyfu cucfmkc

The Decrypted Message: information and security systems
```

## PROGRAM 3

### STEGANOGRAPHY

**Description:**

Steganography is the practice of concealing a file, message, image, or video within another file, message, image, or video. The advantage of steganography over cryptography alone is that the intended secret message does not attract attention to itself as an object of scrutiny. Plainly visible encrypted messages—no matter how unbreakable—arouse interest, and may in themselves be incriminating in countries where encryption is illegal. Thus, whereas cryptography is the practice of protecting the contents of a message alone, steganography is concerned with concealing the fact that a secret message is being sent, as well as concealing the contents of the message.

In this program, an input file is prepared which serves as the carrier for hiding the data. The program then asks for the string to be hidden in the file. The input string is converted into binary and its representation is stored. Then the input file is modified to create a new updated input file which is the same file except that a space denotes a '0' and a double space denotes a '1'. This updated input is sent. At the receiving end, same rules are applied to create the binary representation from the updated file. Thus, the original message is reconstructed.

**Program:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <assert.h>
#include <ctype.h>

static int byteSize;
#define MAX 1000
#define BITS_CHAR 8
#define input_file "input.txt"
#define crypt_file "updated_input.txt"

// A program to hide the input message from the string to a new file.

void getBinary(char * inp, int * binary){
    for (int i = 0; i < strlen(inp); i++) {
        int ascii = (int) inp [i];
        int counter = BITS_CHAR * (i + 1) - 1;
        while (ascii) {
            binary[counter] = ascii % 2;
            ascii /= 2;
            counter -= 1;
        }
    }
}
```

```

    }
}
}

```

### **void readAndCrypt(int \* binary){**

```

    // Creates a crypted file from the given input that contain our message
    // Here, we denote a binary 0 with a single space and 1 with two spaces.
    char myLine[MAX] = {' '};
    FILE * fp = fopen(input_file, "r");
    FILE* fout = fopen(crypt_file, "w");

    int charRead = fread(myLine, sizeof(char), MAX, fp);
    int arrayCounter = 0, i = 0;

    while (i < charRead && arrayCounter < byteSize) {
        if (isspace((int)myLine[i]) && binary[arrayCounter] == 0) {
            // printf("\nWriting single at %d", i);
            fwrite(&myLine[i], sizeof(char), 1, fout);
            arrayCounter += 1;
        }
        else if (isspace((int)myLine[i]) && binary[arrayCounter] == 1) {
            // printf("\nWriting double at %d", i);
            fwrite(&myLine[i], sizeof(char), 1, fout);
            fwrite(&myLine[i], sizeof(char), 1, fout);
            arrayCounter += 1;
        }

        else fwrite(&myLine[i], sizeof(char), 1, fout);
        i++;
    }

    fclose(fp);
    fclose(fout);
}

```

### **void readAndDecrypt(){**

```

    // Decrypts information from the updated input file used as a stegnograph
    // Converts the result back into the value we want
    FILE* fout = fopen(crypt_file, "r");
    assert(fout);

    char myLine[MAX] = {' '};
    int charRead = fread(myLine, sizeof(char), MAX, fout), i = 0;
    int final [MAX] = {0}, finalCounter = 0, ans = 0;

    while (i < charRead && finalCounter < byteSize) {

```

```

        if (isspace((int) myLine[i]) ) {
            if ((i + 1) < charRead && isspace((int)myLine[i + 1])) {
                // printf("\nReading double at %d", i);
                final[finalCounter] = 1;
                i++;
            }
            else {
                // printf("\nReading single at %d", i);
                final[finalCounter] = 0;
            }
            finalCounter += 1;
        }
        i++;
    }

    char decrypt[MAX] = {' '};
    printf("\n\nReceived the message: \t");

    for (int i = 0; i < byteSize; i++) {
        if (final [i] == 1)
            ans += pow(2, BITS_CHAR - 1 - (i % BITS_CHAR));

        if (i > 0 && (i + 1) % BITS_CHAR == 0) {
            char sm [2] = {(char) ans, '\0'};
            strcat(decrypt, sm);
            ans = 0;
        }
    }
    printf("%s\n", decrypt);
}

```

```

int main(){
    char user_input[MAX];

    // Ask user for a text to encrypt

    printf("\nEnter a string to encrypt\t");
    fgets(user_input, MAX, stdin);
    // fscanf(stdin, "%s[^\n]", user_input );
    byteSize = strlen(user_input)* BITS_CHAR;

    int binary[MAX] = {0};
    getBinary(user_input, binary);

    //read carrier file and crypt it into a new file

```



```
readAndCrypt(binary);

// read crypt file and print output
readAndDecrypt();
return 0;
}
```

### OUTPUT:

```
aman@aman ~/Desktop/prog/ISS/1-18 STEG $ cat input.txt
```

```
Cryptography means to convert our plain text data into the unreadable form. This is the subject we
are being taught is this semester. By now you would have realized that I am just playing with you
r mind, and I just wanted some words. In fact, these words are just created so that the spaces bet
ween them can be utilized for hiding messages. Yes, that's right. The input string is converted to
binary. And this file be modified. Wherever, there's a zero, it implies a single space and double
space for 1. I hope my conversion is helpful. And this input file should be long if the message i
s long since it has to have spaces for 1's and 0's
```

```
aman@aman ~/Desktop/prog/ISS/1-18 STEG $ ./steganography
```

```
Enter a string to encrypt      hide me some
```

```
aman@aman ~/Desktop/prog/ISS/1-18 STEG $ cat updated_input.txt
```

```
Cryptography means to convert our plain text data into the unreadable form. This is the subj
ect we are being taught is this semester. By now you would have realized that I am just pl
aying with your mind, and I just wanted some words. In fact, these words are just created s
o that the spaces between them can be utilized for hiding messages. Yes, that's right. The in
put string is converted to binary. And this file be modified. Wherever, there's a zero,
it implies a single space and double space for 1. I hope my conversion is helpful. And th
is input file should be long aman@aman ~/Desktop/prog/ISS/1-18 STEG $
```

```
Received the message:      hide me some
```

## **PROGRAM 4**

### **Play Fair CIPHER**

#### **Description:**

The technique encrypts pairs of letters (digraphs), instead of single letters as in the simple substitution cipher. The Play Fair is thus significantly harder to break since the frequency analysis used for simple substitution ciphers does not work with it. Frequency analysis can still be undertaken, but on the 600 possible digraphs rather than the 26 possible monographs.

The Play Fair cipher uses a 5 by 5 table containing a key word or phrase. Memorization of the keyword and 4 simple rules was all that was required to create the 5 by 5 table and use the cipher.

To generate the key table, one would first fill in the spaces in the table with the letters of the keyword (dropping any duplicate letters), then fill the remaining spaces with the rest of the letters of the alphabet in order (usually putting both "i" and "j" in the same space). The key can be written in the top rows of the table, from left to right, or in some other pattern, such as a spiral beginning in the upper-left-hand corner and ending in the centre. The keyword together with the conventions for filling in the 5 by 5 table constitute the cipher key.

To encrypt a message, one would break the message into digraphs (groups of 2 letters) such that, for example, "hello" becomes "he lx lo", and map them out on the key table. If needed, append a "x" to complete the final digraph. The two letters of the digraph are considered as the opposite corners of a rectangle in the key table. Note the relative position of the corners of this rectangle. Then apply the following 4 rules, in order, to each pair of letters in the plain text.

1. If both letters are the same (or only one letter is left), add an "x" after the first letter. Encrypt the new pair and continue.
2. If the letters appear on the same row of your table, replace them with the letters to their immediate right respectively.
3. If the letters appear on the same column of your table, replace them with the letters immediately below respectively.
4. If the letters are not on the same row or column, replace them with the letters on the same row respectively but at the other pair of corners of the rectangle defined by the original pair.

#### **Program:**

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;
```

```
#define MAT_SIZE 5 //5 * 5 matrix
```

```
#define inputFile "input.txt"
```

```
char * mat[MAT_SIZE];
```

```
static char i_j;
```

```
// A program to implement Play Fair cipher encryption on a string that is read from a file
```

```
void build_matrix(string &str, char ** mat){
```

```
    // Builds a square matrix of size MAT_SIZE using the input string
```

```
    for(int i = 0 ; i<MAT_SIZE; i++)
```

```
        mat[i]= new char[MAT_SIZE];
```

```
    for(int i = 0; i<MAT_SIZE; i++)
```

```
        for(int j = 0; j<MAT_SIZE; j++)
```

```
            mat[i][j] = ' ';
```

```
    set<char> unique;
```

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

```
        unique.insert(str[i]);
```

```
    set<char>::iterator sit = unique.begin();
```

```
    int last_row, last_col;
```

```
    bool over = false;
```

```
    for(int i = 0; i<MAT_SIZE; i++){
```

```
        for(int j = 0; j<MAT_SIZE; j++)
```

```
            if (sit != unique.end()){
```

```
                mat[i][j] = *sit;
```

```
                sit++;
```

```
            }
```

```
            else{
```

```
                last_col = j;
```

```
                last_row = i;
```

```
                over = true;
```

```
                break;
```

```
            }
```

```
            if (over)break;
```

```
    }
```

```
    vector<char>not_added_lower;
```

```
    for(char x = 'a'; x <= 'z'; x++){
```

```
        // Take special case for i = j
```

```
        if (x == 'i'){
```

```
            if (find(unique.begin(), unique.end(), 'j') == unique.end()){
```

```
                not_added_lower.push_back('i');
```

```
                i_j = 'i';
```

```

        unique.insert('i');
    }
}
else if (x == 'j'){
    if (find(unique.begin(), unique.end(), 'i') == unique.end()){
        not_added_lower.push_back('j');
        i_j = 'j';
        unique.insert('j');
    }
}
else if (find(unique.begin(), unique.end(), x) == unique.end())
    not_added_lower.push_back(x);
}
}

```

```

void encrypt(char a, char b, vector<char>& outputA, vector<char>& outputB){
    // Given two characters a and b of a digraph, encrypt them according to the matrix
    int x1, x2, y1, y2;

    for(int i = 0; i<MAT_SIZE; i++)
        for(int j = 0; j<MAT_SIZE; j++)
            if (mat[i][j] == a){
                x1 = i;
                y1 = j;
                break;
            }

    for(int i = 0; i<MAT_SIZE; i++)
        for(int j = 0; j<MAT_SIZE; j++)
            if (mat[i][j] == b){
                x2 = i;
                y2 = j;
                break;
            }

    if (x1 == x2){
        outputA.push_back(mat[x1][(y1+1) % MAT_SIZE]);
        outputB.push_back(mat[x2][(y2+1) % MAT_SIZE]);
    }

    else if (y1 == y2){
        outputA.push_back(mat[(x1+1) % MAT_SIZE][y1]);
        outputB.push_back(mat[(x2+1) % MAT_SIZE][y2]);
    }

    else{
        outputA.push_back(mat[x2][y1]);
    }
}

```

```
        outputB.push_back(mat[x1][y2]);
    }
}
```

```
void decrypt(char a, char b, vector<char>& inputA, vector<char>& inputB){
    // Given two characters a and b of a digraph, decrypt them according to the matrix
    int x1, x2, y1, y2;

    for(int i = 0; i < MAT_SIZE; i++)
        for(int j = 0; j < MAT_SIZE; j++)
            if (mat[i][j] == a){
                x1 = i;
                y1 = j;
                break;
            }

    for(int i = 0; i < MAT_SIZE; i++)
        for(int j = 0; j < MAT_SIZE; j++)
            if (mat[i][j] == b){
                x2 = i;
                y2 = j;
                break;
            }

    if (x1 == x2){
        inputA.push_back(mat[x1][(y1 - 1) % MAT_SIZE]);
        inputB.push_back(mat[x2][(y2 - 1) % MAT_SIZE]);
    }

    else if (y1 == y2){
        inputA.push_back(mat[(x1 - 1) % MAT_SIZE][y1]);
        inputB.push_back(mat[(x2 - 1) % MAT_SIZE][y2]);
    }

    else{
        inputA.push_back(mat[x2][y1]);
        inputB.push_back(mat[x1][y2]);
    }
}
```

```
void solve(vector<char> & inputA, vector<char> & inputB){
    // Given the digraph, encrypts it according to the Play Fair matrix
    vector<char>:: iterator it = inputA.begin();
    vector<char>:: iterator it2 = inputB.begin();
    vector<char> outputA, reoutA;
```

```
vector<char> outputB, reoutB;

inputA.pop_back();
inputB.pop_back();

cout << "\n The Digraph Message: ";

for(;it != inputA.end();it++, it2++){
    cout << *it << *it2 << " ";
    encrypt(*it, *it2, outputA, outputB);
}

cout << "\n The Encrypt Message: ";
for(int i = 0; i < outputA.size(); i++){
    cout << outputA[i] << outputB[i] << " ";
}

cout << "\n The Decrypt Message: ";
for(int i = 0; i < outputA.size(); i++)
    decrypt(outputA[i], outputB[i], reoutA, reoutB);

it = reoutA.begin();
it2 = reoutB.begin();
for(;it != reoutA.end(); it++, it2++)
    cout << *it << *it2 << " ";
}

void consoleInput(vector<char> &inputA, vector<char> &inputB){
    // Reads input string from a file and prepares the digraph
    ifstream inf(inputFile);
    char c, i1, i2;
    while((c = inf.get())!= EOF){

        if(!isalnum(c))
            continue;
        i1 = c;
        i2 = inf.get();
        while(!isalnum(i2) && i2 != EOF)
            i2 = inf.get();

        if(i1 == i2){
            inputA.push_back(i1);
            inputB.push_back('x');
            inputA.push_back(i2);
            i2 = inf.get();
            while(!isalnum(i2) && i2 != EOF)
                i2 = inf.get();
        }
    }
}
```

```
        inputB.push_back(i2);
    }
    else{
        inputA.push_back(i1);
        if (i2 == EOF)
            inputB.push_back('x');
        else inputB.push_back(i2);
    }
}

inf.close();
}
```

```
int main(){
    string str;
    cout << "\n Enter key string for the matrix: ";
    cin >> str;
    build_matrix(str, mat);

    vector <char> inputA;
    vector <char> inputB;
    consoleInput(inputA, inputB);
    solve(inputA, inputB);
    return 0;
}
```

### OUTPUT:

```
aman@aman ~/Desktop/prog/ISS/2-08 RAIL_PLAY $ cat input.txt
hello this is a test string
```

```
aman@aman ~/Desktop/prog/ISS/2-08 RAIL_PLAY $ ./play_fair
```

```
Enter key string for the matrix: anc
```

```
a c n b d
```

```
e f g h i
```

```
k l m o p Digraph Message:
```

```
q r s t u
```

```
v w x y z
```

```
The Digraph Message: he lx lo th is is at es ts tr in
```

```
The Encrypt Message: if wm mp yo ug ug qb qg ut us dg
```

```
The Decrypt Message: he lx lo th is is at es ts tr in
```

**PROGRAM 5****RAIL FENCE CIPHER****Description:**

Rail Fence Cipher (also called a zigzag cipher) generally refers to a form of transposition cipher. It derives its name from the way in which it is encoded. In the rail fence cipher, the plain text is written downwards and diagonally on successive "rails" of an imaginary fence, then moving up when we reach the bottom rail. When we reach the top rail, the message is written downwards again until the whole plaintext is written out. The message is then read off in rows. For example, if we have 3 "rails" and a message of 'WE ARE DISCOVERED. FLEE AT ONCE', the cipher text is

```
W . . . E . . . C . . . R . . . L . . . T . . . E
. E . R . D . S . O . E . E . F . E . A . O . C .
. . A . . I . . V . . D . . E . . N . .
```

Then reads off to get the ciphertext:

```
WECRL TEERD SOEEF EAOCA IVDEN
```

**Program:**

```
#include <bits/stdc++.h>
using namespace std;

int numRails;
string userStr;
vector<char> rail;

void display(vector<char> &rail){
    cout << "\n Rail: ";
    for (auto a : rail)
        cout << a << " ";
    cout << endl;
}

void encrypt(){
    //encrypts the text with the given number of rails
    rail.resize(0);
    int starter, gap;
    cout << "\n*****ENCRYPTION*****\n";
```



```
for (int railNum = 0; railNum < numRails; railNum++){
    if (railNum == 0 || railNum == numRails - 1){
        starter = railNum;
        gap = 2 * (numRails - 1);
        rail.push_back(userStr[starter]);

        starter += gap;
        while (starter < userStr.size()){
            rail.push_back(userStr[starter]);
            starter += gap;
        }
    }

    else{
        bool isOdd = true;
        starter = railNum;
        int railBelow, railAbove;
        while (starter < userStr.size()){
            rail.push_back(userStr[starter]);

            if (isOdd){ //do odd first
                isOdd = false;
                railBelow = numRails - railNum - 1;
                gap = 2 * railBelow;
            }
            else{
                railAbove = railNum;
                gap = 2 * railAbove;
                isOdd = true;
            }
            starter += gap;
        }
    }
    display(rail);
}
```

```
void decrypt(){
    //deccrypts the rails to the original message
    vector<char> deRail(rail.size(), '-');
    int i = 0, starter, gap, j;
    cout << "\n*****DECRYPTION*****\n";
```

```
for (int iter = 0; iter < numRails; iter++){
    if (iter == 0 || iter == numRails -1){
        starter = iter;
        deRail[starter] = rail[i ++];
        gap = 2 * (numRails -1);
        starter += gap;

        while(starter < rail.size()){
            deRail[starter] = rail[i ++];
            starter += gap;
        }
    }
    else{
        bool isOdd = true;
        starter = iter;

        while (starter < deRail.size()){
            if (isOdd){
                gap = 2 * (numRails - iter -1);
                deRail[starter] = rail[i++];
                isOdd = false;
                starter += gap;
            }
            else{
                isOdd = true;
                deRail[starter] = rail[i++];
                gap = 2 * iter;
                starter += gap;
            }
        }
    }
    display(deRail);
}
```

```
int main(){
```

```
    cout <<" Enter String: ";
    cin >> userStr;
    cout <<" Enter number of rails: ";
    cin >> numRails;
```

```
    encrypt();
```

```
decrypt();  
return 0;  
}
```

**OUTPUT:**

```
aman@aman ~/Desktop/prog/ISS/2-08 RAILFENCE $ ./railFence
```

```
Enter String: COMPUTERSCIENCE  
Enter number of rails: 5
```

```
*****ENCRYPTION*****
```

```
Rail: C S  
Rail: C S O R C  
Rail: C S O R C M E I E  
Rail: C S O R C M E I E P T E C  
Rail: C S O R C M E I E P T E C U N
```

```
*****DECRYPTION*****
```

```
Rail: C - - - - - S - - - - -  
Rail: C O - - - - - R S C - - - - -  
Rail: C O M - - - E R S C I - - - E  
Rail: C O M P - T E R S C I E - C E  
Rail: C O M P U T E R S C I E N C E
```

## PROGRAM 6

### COLUMNAR CIPHER

**Description :**

In cryptography, a Columnar cipher is a method of encryption by which the units of plain text (which are commonly characters or groups of characters) are written row by row until the column length is reached which is equal to the key, so that the cipher text constitutes a permutation of the plaintext. That is, the order of the units is changed (the plaintext is reordered).

The input to the program is the key i.e the maximum column length, and the input string which is to be encrypted. If the length of the input is not a multiple of key, 'z' are padded onto the end of the string.

**Program:**

```
#include <bits/stdc++.h>
using namespace std;

static int key;
static string input;
static vector<string> rail;

// A program to implement columnar cipher.

string encrypt(){
    //Encrypt the rails. i.e Read column by column
    string output;

    for (int col = 0; col < key; col ++){
        for (auto a : rail)
            output.push_back(a[col]);
    }
    return output;
}

int main(){
    cout << "\n Enter key: ";
    cin >> key;

    cout << "\n Enter input string: ";
    cin.ignore();
    getline(cin, input);

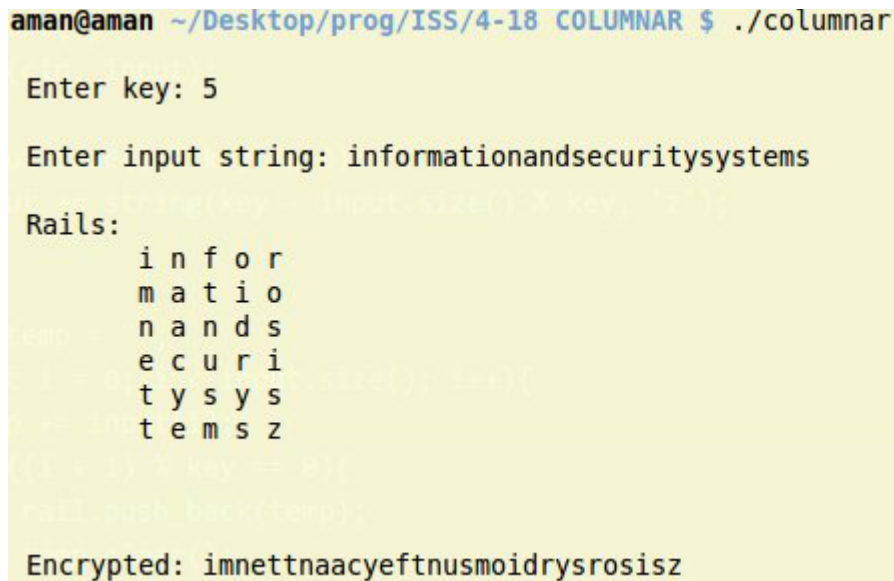
    if (input.size() % key != 0)
        input += string(key - input.size() % key, 'z');
```

```
string temp = "";
for (int i = 0; i < input.size(); i++){
    temp += input[i];
    if ((i + 1) % key == 0){
        rail.push_back(temp);
        temp.clear();
    }
}

cout << "\n Rails: \n";
for (auto a : rail){
    cout << "\t";
    for (auto b : a)
        cout << b << " ";
    cout << endl;
}

cout << endl << endl;
string output = encrypt();
cout << "\n Encrypted: ";
cout << output << endl << endl;

return 0;
}
```

**OUTPUT:**

```
aman@aman ~/Desktop/prog/ISS/4-18 COLUMNAR $ ./columnar

Enter key: 5

Enter input string: informationandsecuritysystems

Rails:
    i n f o r
    m a t i o
    n a n d s
    e c u r i
    t y s y s
    t e m s z

Encrypted: imnettnaacyeftnusmoidrysrosisz
```

**PROGRAM 7****EUCLIDEAN ALGORITHM****Description:**

The Euclidean algorithm, is an efficient method for computing the greatest common divisor (GCD) of two numbers, the largest number that divides both of them without leaving a remainder. It is named after the ancient Greek mathematician Euclid, who first described it in Euclid's Elements (c. 300 BC).

It is an example of an algorithm, a step-by-step procedure for performing a calculation according to well-defined rules, and is one of the oldest algorithms in common use.

It can be used to reduce fractions to their simplest form, and is a part of many other number-theoretic and cryptographic calculations.

**Program :**

```
#include <bits/stdc++.h>
using namespace std;
//A program to compute GCD of two numbers by Euclidean division

static int a, b;

int gcd(int a, int b){
    // Given two numbers, finds GCD using Euclidean Method
    if (a < b)
        gcd (b, a);
    if (b == 0)
        return a;
    return gcd(b, a%b);
}

int main(){
    while (true){
        cout << "\n Enter two space separated numbers: ";
        cin >> a >> b;
        cout << "\n GCD: \t" << gcd(a, b) << endl;
    }
    return 0;
}
```

**OUTPUT:**

```
aman@aman ~/Desktop/prog/ISS/3-14 EGCD $ ./gcd
```

```
Enter two space separated numbers: 60 8
```

```
GCD: 4
```

```
Enter two space separated numbers: 310 111
```

```
GCD: 1
```

```
Enter two space separated numbers: 89 180
```

```
GCD: 1
```

```
Enter two space separated numbers: 100 25
```

```
GCD: 25
```

```
Enter two space separated numbers: 1898 99
```

```
GCD: 1
```

**PROGRAM 8****EXTENDED EUCLIDEAN ALGORITHM****Description:**

In arithmetic and computer programming, the extended Euclidean algorithm is an extension to the Euclidean algorithm, which computes, besides the greatest common divisor of integers  $a$  and  $b$ , the coefficients of Bézout's identity, that is integers  $x$  and  $y$  such that,

$$ax + by = \gcd(a, b)$$

It allows one to compute also, with almost no extra cost, the quotients of  $a$  and  $b$  by their greatest common divisor. The extended Euclidean algorithm is particularly useful when  $a$  and  $b$  are coprime, since  $x$  is the modular multiplicative inverse of  $a$  modulo  $b$ , and  $y$  is the modular multiplicative inverse of  $b$  modulo  $a$ . Similarly, the polynomial extended Euclidean algorithm allows one to compute the multiplicative inverse in algebraic field extensions. It follows that both extended Euclidean algorithms are widely used in cryptography. In particular, the computation of the modular multiplicative inverse is an essential step in RSA public-key encryption method.

**Program :**

```
#include <bits/stdc++.h>
using namespace std;
//A program to compute the extended GCD of two numbers by Euclidean division
// To find a, b such that ax + by = gcd(a, b)

int gcd(int a, int b){
    // Given two numbers, finds GCD using Euclidean Method
    if (a < b)
        gcd (b, a);
    return (b == 0) ? a : gcd(b, a%b);
}

tuple<int, int> extendedGCDUtil(int a, int b, int s1, int s2, int t1, int t2){
    // Util method to find two solutions s1, t1
    if (b == 0)
        return make_tuple(s1, t1);
    return extendedGCDUtil(b, a % b, s2, s1 - (a / b) * s2, t2, t1 - (a / b) * t2);
}

tuple<int, int> extendedGCD(int a, int b){
    // Returns a pair of integers r and s such that a *r + b *s = 1
    // Solutions exist only when gcd(a, b) == 1
    if (gcd (a, b) != 1)
        return make_tuple(0,0);
    return extendedGCDUtil(a, b, 1, 0, 0, 1);
}
```



```
}
```

```
int main(){
    int numA, numB, a, b, gcdVal;

    while (true){
        cout << "\n Enter two space separated numbers: ";
        cin >> numA >> numB;
        gcdVal = gcd(numA, numB);
        cout << "\n GCD: \t" << gcdVal << endl;

        if (gcdVal != 1 )
            printf("\t%d and %d are not coprime. No such a and b exist\n", numA, numB );
        else{
            tie(a, b) = extendedGCD(numA, numB);
            printf("\ta: %d, b: %d => satisfy %d(%d) + %d(%d) = 1\n", a, b, a, numA, b, numB );
        }
    }
    return 0;
}
```

### OUTPUT:

```
aman@aman ~/Desktop/prog/ISS/3-14 EGCD $ ./egcd
Enter two space separated numbers: 10 21
GCD: 1
a: -2, b: 1 => satisfy -2(10) + 1(21) = 1
Enter two space separated numbers: 81 34
GCD: 1
a: -13, b: 31 => satisfy -13(81) + 31(34) = 1
Enter two space separated numbers: 11 20
GCD: 1
a: -9, b: 5 => satisfy -9(11) + 5(20) = 1
Enter two space separated numbers: 5 7
GCD: 1
a: 3, b: -2 => satisfy 3(5) + -2(7) = 1
Enter two space separated numbers: 6 3
GCD: 3
6 and 3 are not coprime. No such a and b exist
```

**PROGRAM 9****CHINESE REMAINDER THEOREM****Description:**

The Chinese remainder theorem is a result about congruences in number theory and its generalizations in abstract algebra. It was first published some time between the 3rd and 5th centuries by the Chinese mathematician Sun Tzu. In its basic form, the Chinese remainder theorem will determine a number  $n$  that, when divided by some given divisors, leaves given remainders.

The program takes as input , the number of equations  $n$  of the form  $x = a \bmod m$ . Then  $n$  equations follow. The prerequisite for solving these equations is that all the  $m$ 's in the equations must be coprime. If they aren't, then no solution exists. Else, it does.

After storing  $M = m_1 * m_2 * m_3$  and so on, find divs such that  $M_1 = M/m_1$  ,  $M_2 = M/m_2$  and so on.. For each entry in divs, find the corresponding inverses mod  $M$ . Now, final step is to find

$$X = a * m_1 * m_1^{(-1)} + b * m_2 * m_2^{(-1)} \dots$$

**Program:**

```
#include <bits/stdc++.h>
using namespace std;
```

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

```
    // Given two numbers, finds GCD using Euclidean Method
```

```
    if (a < b)
```

```
        gcd (b, a);
```

```
    if (b == 0)
```

```
        return a;
```

```
    return gcd(b, a%b);
```

```
}
```

```
tuple<int, int> extendedGCDUtil(int a, int b, int s1, int s2, int t1, int t2){
```

```
    // Util method to find two solutions s1, t1
```

```
    if (b == 0)
```

```
        return make_tuple(s1, t1);
```

```
    return extendedGCDUtil(b, a % b, s2, s1 - (a / b) * s2, t2, t1 - (a / b) * t2);
```

```
}
```

```
tuple<int, int> extendedGCD(int a, int b){
```

```
    // Returns a pair of integers r and s such that a * r + b * s = 1
```

```
    // Solutions exist only when gcd(a, b) == 1
```

```
    if (gcd (a, b) != 1)
```

```
        return make_tuple(0,0);
```

```
    return extendedGCDUtil(a, b, 1, 0, 0, 1);
```

}

```
int main(){
    int numEq, a, b, M = 1, r, s;
    vector<int> nums, mods, inverses, divs;
    // Ask user for n equations of the form X = a mod m1, X = b mod m2
    cout << "\n Enter n (number of equations): \t";
    cin >> numEq;

    for (int i = 0; i < numEq; i++){
        cout << "\n Enter (a, b) such that X = a mod b\t ";
        cin >> a >> b;
        nums.push_back(a);
        mods.push_back(b);
        M *= b;
    }
    // Preliminary display
    cout << "\n Big M: " << M << endl << endl;
    cout << "Nums: " << nums << endl;
    cout << "Mods: " << mods << endl;

    // Check if all smaller mods are coPrime
    int modGCD = mods[0];
    for (int i = 1; i < mods.size(); i++)
        modGCD = gcd(modGCD, mods[i]);

    if (modGCD != 1){
        cout << "\n Given modulii are not coprime to each other";
        cout << "\n No solution exists\n";
        return -1;
    }
    // After storing M = m1 * m2 * m3 and so on
    // Find divs such that M1 = M/M1 , M2 = M/M2 and so on..

    for (auto m : mods)
        divs.push_back(M / m);
    cout << "Divs: " << divs << endl;

    // For each entry in divs, find the corresponding inverses mod M
    for (int i = 0; i < numEq; i++){
        tie(r, s) = extendedGCD(divs[i], mods[i]);
        inverses.push_back((r + mods[i]) % mods[i]);
    }
    cout << "Invs: " << inverses << endl;
```

```
// Now, final step is to find  $X = a * m1 * m1^{(-1)} + b * m2 * m2^{(-1)}...$ 
int ans = 0;
for (int i = 0; i < numEq; i++)
    ans = (ans + nums[i] * divs[i] * inverses[i]) % M;

cout << " Solution to these equations : " << ans << endl;
cout << "\n Verification: \n";
for (auto a : mods)
    cout << " " << ans << " % " << a << " : " << ans % a << endl;
return 0;
}
```

**OUTPUT:**

```
aman@aman ~/Desktop/prog/ISS/2-15 CRT $ ./chineseRem

Enter n (number of equations):          4

Enter (a, b) such that  $X = a \bmod b$       3 5
Enter (a, b) such that  $X = a \bmod b$       4 8
Enter (a, b) such that  $X = a \bmod b$      12 19
Enter (a, b) such that  $X = a \bmod b$      31 7

Big M: 5320

Nums:  3 4 12 31
Mods:  5 8 19 7
Divs:  1064 665 280 760

Invs:  4 1 15 2

Solution to these equations : 1228

Verification:
1228 % 5 : 3
1228 % 8 : 4
1228 % 19 : 12
1228 % 7 : 3
```

**PROGRAM 10****RSA, PRIMALITY TESTING, RANDOM NO GENERATION****Description:**

RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of them can be given to everyone. The other key must be kept private. It is based on the fact that finding the factors of an integer is hard (the factoring problem)

The keys for RSA can be generated as follows:

- 1) Choose two large prime numbers  $p$  and  $q$
- 2) Calculate  $n = pq$  and the totient  $t = (p - 1) * (q - 1)$
- 3) Choose an integer  $e$  such that  $1 < e < t$  and  $\gcd(e, t) = 1$ . This 'e' is the public key exponent
- 4) Compute  $d$  as  $(e \text{ inverse}) \bmod t$ . This 'd' is the private key exponent.
- 5) Compute cipher text as  $C = (m^e) \bmod n$  using fast exponentiation
- 6) Compute decrypted message as  $M = (C^d) \bmod n$

The two large prime numbers are generated randomly using a linear congruential generator (LCG) and then checked for primality using trial division uptill the square root of the number.

The LCG is defined as the following recurrence.

$$X(n + 1) = ( a(Xn) + c ) \bmod m$$

where  $X(n + 1)$  is the next seed,  $a$  is the multiplier,  $c$  is the increment and  $m$  is the maximum range of the random number required.

$X_0$  is the initial seed that is kept at the value of time(NULL) so that it's random at that moment

$a = 1103515245$

$c = 12345$

$m = 65536$  (this sequence is used by GCC to generate random values)

The random values are then checked for primality by trial division which works in  $O(\sqrt{N})$ . This is a deterministic test. Other faster probabilistic tests such as Rabin Miller Karp can also be used.

**PROGRAM:**

```
#include <bits/stdc++.h>
#define seed rand()
typedef long long unsigned ll;
using namespace std;

ll nextSeed = 1;
```

```
int myRand() {
```

```
    //generates a random number with the given seed using linear congruential generator
```

```
    nextSeed = nextSeed * 1103515245 + 12345;
```

```
    return ((nextSeed/65536) % 65536);
```

```
}
```

```
tuple<ll, ll> extendedGCDUtil(ll a, ll b, ll s1, ll s2, ll t1, ll t2){
```

```
    // Util method to find two solutions s1, t1
```

```
    if (b == 0)
```

```
        return make_tuple(s1, t1);
```

```
    return extendedGCDUtil(b, a % b, s2, s1 - (a / b) * s2, t2, t1 - (a / b) * t2);
```

```
}
```

```
tuple<ll, ll> extendedGCD(ll a, ll b){
```

```
    // Returns a pair of llers r and s such that a *r + b *s = 1
```

```
    // Solutions exist only when gcd(a, b) == 1
```

```
    // Let tie (r, s) = extendedGCD(divs[i], mods[i]); then (r + mods[i]) % mods[i] is the mod inv
```

```
    if (gcd (a, b) != 1)
```

```
        return make_tuple(0,0);
```

```
    return extendedGCDUtil(a, b, 1, 0, 0, 1);
```

```
}
```

```
ll fastExp(ll a, ll b, ll mod){
```

```
    //fast modular exponentiation of a ^ b % mod
```

```
    if (b == 1)
```

```
        return a % mod;
```

```
    else{
```

```
        if (b % 2 == 0)
```

```
            return fastExp((a * a) % mod, b/2, mod) % mod;
```

```
        else
```

```
            return (a % mod * (fastExp(a, b -1, mod) % mod)) % mod;
```

```
    }
```

```
}
```

```
bool primalityTest(ll n){
```

```
    //Given a number n, tests whether it is prime or not
```

```
    if (n < 3)
```

```
        return n == 2;
```

```
    else if (n % 2 == 0)
```

```
        return false;
```

```
    for (ll i = 3; i * i <= n; i++){
```

```
        if (n % i == 0)
```

```
            return false;
```

```
}  
    return true;  
}
```

```
void rsaSolver(ll n1, ll n2){
```

```
    //Rsa algorithm given two large prime numbers
```

```
    ll msg;
```

```
    cout << "\n Enter number to encode: ";
```

```
    cin >> msg;
```

```
    ll n = n1 * n2, phyN = (n1 - 1) * (n2 - 1);
```

```
    ll e;
```

```
    for (e = 2; e < phyN; e++)
```

```
        if (gcd(e, phyN) == 1)
```

```
            break;
```

```
    ll d, s;
```

```
    tie (d, s) = extendedGCD(e, phyN);
```

```
    d = (d + phyN) % phyN;
```

```
    ll enc = fastExp(msg, e, n);
```

```
    cout << "\n The crypted message is: " << enc << endl;
```

```
    ll origMsg = fastExp(enc, d, n);
```

```
    cout << "\n The decrypted message is: " << origMsg << endl;
```

```
}
```

```
int main(){
```

```
    ll n1 = 0, n2 = 0;
```

```
    nextSeed = time(NULL);
```

```
    while(true){
```

```
        n1 = myRand();
```

```
        if (primalityTest(n1))
```

```
            break;
```

```
    }
```

```
    nextSeed = time(NULL);
```

```
    while(true){
```

```
        n2 = myRand();
```

```
        if (primalityTest(n2) and n2 != n1)
```

```
            break;
```

```
    }
```

```
    assert (gcd(n1, n2) == 1 && primalityTest(n1) && primalityTest(n2));
```

```
cout << "\n Generated Prime Random n1: " << n1 << endl;
cout << "\n Generated Prime Random n2: " << n2 << endl;
rsaSolver(n1, n2);
return 0;
}
```

**OUTPUT:**

```
aman@aman ~/Desktop/prog/ISS/2-22 PRIME_RSA $ ./primeMod
```

```
Generated Prime Random n1: 62311
```

```
Generated Prime Random n2: 59771
```

```
Enter number to encode: 2341
```

```
N: 3724390781
```

```
phyN: 3724268700
```

```
e: 7
```

```
d: 2128153543
```

```
The crypted message is: 403727544
```

```
The decrypted message is: 2341
```

```
aman@aman ~/Desktop/prog/ISS/2-22 PRIME_RSA $ ./primeMod
```

```
Generated Prime Random n1: 53653
```

```
Generated Prime Random n2: 9733
```

```
Enter number to encode: 224
```

```
N: 522204649
```

```
phyN: 522141264
```

```
e: 5
```

```
d: 104428253
```

```
The crypted message is: 490522353
```

```
The decrypted message is: 224
```



## PROGRAM 11

### MESSAGE DIGEST (MD5) Algorithm

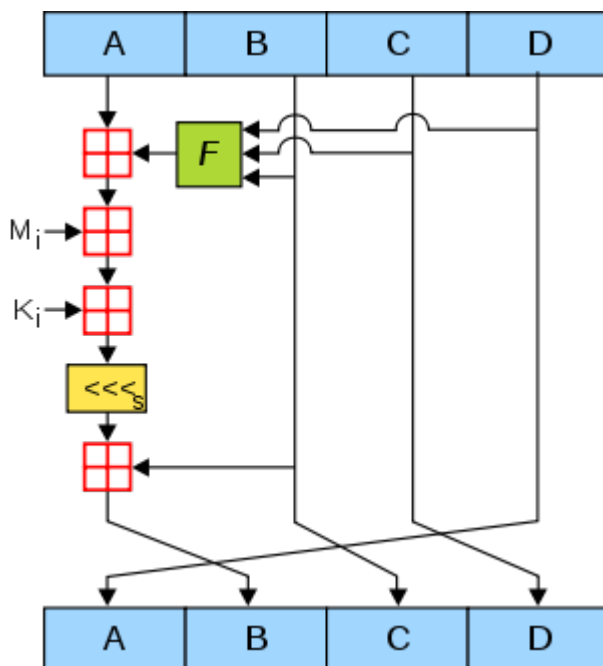
#### Description:

The MD5 message-digest algorithm is a widely used cryptographic hash function producing a 128-bit (16-byte) hash value, typically expressed in text format as a 32-digit hexadecimal number. MD5 has been utilized in a wide variety of cryptographic applications and is also commonly used to verify data integrity. MD5 is a one-way function; it is neither encryption nor encoding. It cannot be reversed other than by brute-force attack.

MD5 processes a variable-length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit blocks (sixteen 32-bit words); the message is padded so that its length is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits fewer than a multiple of 512. The remaining bits are filled up with 64 bits representing the length of the original message, modulo 264.

The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted A, B, C, and D. These are initialized to certain fixed constants. The main algorithm then uses each 512-bit message block in turn to modify the state. The processing of a message block consists of four similar stages, termed rounds; each round is composed of 16 similar operations based on a non-linear function F, modular addition, and left rotation. Figure illustrates one operation within a round. There are four possible functions F and a different one is used in each round:

The security of the MD5 hash function is severely compromised.



$$F(B, C, D) = (B \wedge C) \vee (\neg B \wedge D)$$

$$G(B, C, D) = (B \wedge D) \vee (C \wedge \neg D)$$

$$H(B, C, D) = B \oplus C \oplus D$$

$$I(B, C, D) = C \oplus (B \vee \neg D)$$

**PROGRAM:**

```

#include <bits/stdc++.h>
using namespace std;
#define INPUT_FILE "input"
/* A program to generate the message digest as governed by the MD5 algorithm */

// Initialise chaining variables
uint a0, b0, c0, d0;
vector<uint> K(64); //handles constants
vector<uint> S(64); //handles shift values

uint getPaddingLength(uint inputBits){
    // Given the number of inputBits, returns the required padding bits in MD5
    int paddingMust = inputBits < 512 ? 512 : 0;
    return 512 - (inputBits + 64) % 512 + paddingMust;
}

string getPaddedString(uint bitCount){
    // Given the number of inputBits, returns the padded string for that input
    uint numZero = bitCount - 1;
    int paddingMust = bitCount < 512 ? 512 : 0;
    return '1' + string(numZero + paddingMust, '0');
}

uint binaryToChar(string &bin){
    // Given a binary form, return the decimal expansion
    uint num = 0, len = bin.size();
    for (uint i = 0; i < len; i++){
        if (bin[i] == '1')
            num += 1 << (len - 1 - i);
    }
    return num;
}

string numToBinary(uint num, uint padLength){
    // Given an integer, return a string denoting its binary representation padded appropriately
    string binary;
    while(num){
        binary += (num % 2) ? "1" : "0";
        num /= 2;
    }
    reverse(binary.begin(), binary.end());
    uint padZero = padLength - binary.size();

```

```

    return string(padZero, '0').append(binary);
}

```

```

uint leftRotate (uint x, uint c){

```

```

    // To left rotate a number x by c bits

```

```

    return (x << c) | (x >> (32 -c));

```

```

}

```

```

void setupConstants(){

```

```

    // Setup K and shift array for all 64 rounds

```

```

    a0 = 1732584193;    //0x67452301

```

```

    b0 = 4023233417;    //0xefcdab89

```

```

    c0 = 2562383102;    //0x98badcfe

```

```

    d0 = 271733878;     //0x10325476

```

```

    for (uint i = 0; i < 64; i++)

```

```

        K[i] = uint(4294967296 * abs(sin(i + 1)));

```

```

    S[0] = 7; S[1] = 12; S[2] = 17; S[3] = 22; S[4] = 7; S[5] = 12; S[6] = 17; S[7] = 22;

```

```

    S[8] = 7; S[9] = 12; S[10] = 17; S[11] = 22; S[12] = 7; S[13] = 12; S[14] = 17; S[15] = 22;

```

```

    S[16] = 5; S[17] = 9; S[18] = 14; S[19] = 20; S[20] = 5; S[21] = 9; S[22] = 14; S[23] = 20;

```

```

    S[24] = 5; S[25] = 9; S[26] = 14; S[27] = 20; S[28] = 5; S[29] = 9; S[30] = 14; S[31] = 20;

```

```

    S[32] = 4; S[33] = 11; S[34] = 16; S[35] = 23; S[36] = 4; S[37] = 11; S[38] = 16; S[39] = 23;

```

```

    S[40] = 4; S[41] = 11; S[42] = 16; S[43] = 23; S[44] = 4; S[45] = 11; S[46] = 16; S[47] = 23;

```

```

    S[48] = 6; S[49] = 10; S[50] = 15; S[51] = 21; S[52] = 6; S[53] = 10; S[54] = 15; S[55] = 21;

```

```

    S[56] = 6; S[57] = 10; S[58] = 15; S[59] = 21; S[60] = 6; S[61] = 10; S[62] = 15; S[63] = 21;

```

```

}

```

```

void md5(vector<string> &block){

```

```

    /* Given a message broken into 512 bit blocks, perform MD5 algorithm */

```

```

    //Process the message in successive 512-bit chunks

```

```

    setupConstants();

```

```

    for (auto a : block){

```

```

        cout << "\n BLOCK => " << a << endl;

```

```

        // for each 512-bit chunk of message

```

```

        //break chunk into sixteen 32-bit words M[j], 0 ≤ j ≤ 15

```

```

        vector<uint> M(16);

```

```

        string subBlock;

```

```

        for (uint i = 0, j = 0; i < a.size(); i++){

```

```

            subBlock += a[i];

```

```

            if ((i + 1) % 32 == 0){

```

```

                M[j] = binaryToChar(subBlock);

```

```

                cout << "\n\t Subblock: " << subBlock << " : " << M[j];

```

```

                subBlock.clear();
            }
        }
    }
}

```

```

    }
}
cout << endl;

//Initialize hash value for this chunk:
uint A = a0, B = b0, C = c0, D = d0;
uint F, G, dTemp;

//16 iterations per four major rounds = 64 rounds
for (uint i = 0; i < 64; i++){
    if (i <= 15){
        F = (B & C) | (~B & D);
        G = i;
    }
    else if (i >= 16 && i <= 31){
        F = (D & B) | (~D & C);
        G = (5 * i + 1) % 16;
    }
    else if (i >= 32 && i <= 47){
        F = B ^ C ^ D;
        G = (3 * i + 5) % 16;
    }
    else if (i >= 48 && i <= 63){
        F = C ^ (B | ~D);
        G = (7 * i) % 16;
    }
    dTemp = D;
    D = C;
    C = B;
    B = B + leftRotate(A + F + K[i] + M[G], S[i]);
    A = dTemp;
}
//Add this chunk's hash to result so far:
a0 += A;
b0 += B;
c0 += C;
d0 += D;
}
// The final chaining variables are a0, b0, c0, d0 (32 bit). Convert them to hex and append
cout << "\n Final Chaining variable: \n";
cout << "\t\tA: " << a0 << ": " << hex << a0 << endl;
cout << "\t\tB: " << b0 << ": " << hex << b0 << endl;
cout << "\t\tC: " << c0 << ": " << hex << c0 << endl;
cout << "\t\tD: " << d0 << ": " << hex << d0 << endl;

```

```
stringstream md;
md << hex << a0 << hex << b0 << hex << c0 << hex << d0;
cout << "\n MD5 Digest: " << md.str() << endl << endl;
}

int main(){
    // Read input file char by char and process it into 512 bit (64 bytes) inputS
    // 1 char = 1 byte. So array[64] to hold one block
    ifstream inf(INPUT_FILE);
    char c; string inputString;

    vector< string > block; //64 bytes per block
    static uint byteCount = -1;
    uint bitCount = 0;

    while( (c = inf.get())!= EOF){
        inputString += c;
        byteCount += 1;
    }
    inputString.erase(remove(inputString.begin(), inputString.end(), '\n'));
    bitCount = byteCount * 8;
    cout << "\n Number of bytes in original message: " << byteCount;
    cout << "\n Number of bits in original message: " << bitCount << endl;
    cout << "\n Length of message to be appended: 64 bits ";
    cout << "\n Number of bits to be padded : " << getPaddingLength(bitCount) << endl;

    string byteTobit;
    for (char c : inputString)
        byteTobit += numToBinary(c, 8);

    inputString = byteTobit;
    cout << "\n ORIGINAL: \n" << inputString << endl;

    string paddedString = getPaddedString(getPaddingLength(bitCount));
    cout << "\n PADDED: \n" << paddedString << endl;
    inputString += paddedString;

    string lengthString = numToBinary(bitCount, 64);
    inputString += lengthString;
    cout << "\n LENGTH(64): \n" << lengthString << endl;
    // Process non padded bytes into block
    string blockString = "";

    for (uint i = 0; i < inputString.size(); i++){
```

**OUTPUT:**

[illegible]



MD5 Digest: c70f454797b8931a301a15e97c2260c