INFORMATION SECURITY SYSTEMS

LAB PRACTICALS RECORD

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Dr. B R AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY
JALANDHAR – 144011, PUNJAB (INDIA)

Submitted To:

Ms. Deepika Sethi
Asst. Professor
Department of CSE

Submitted By:

Nikhil Bansal
13103011
6th Semester

PROGRAM 1

ADDITIVE CIPHER

Description:

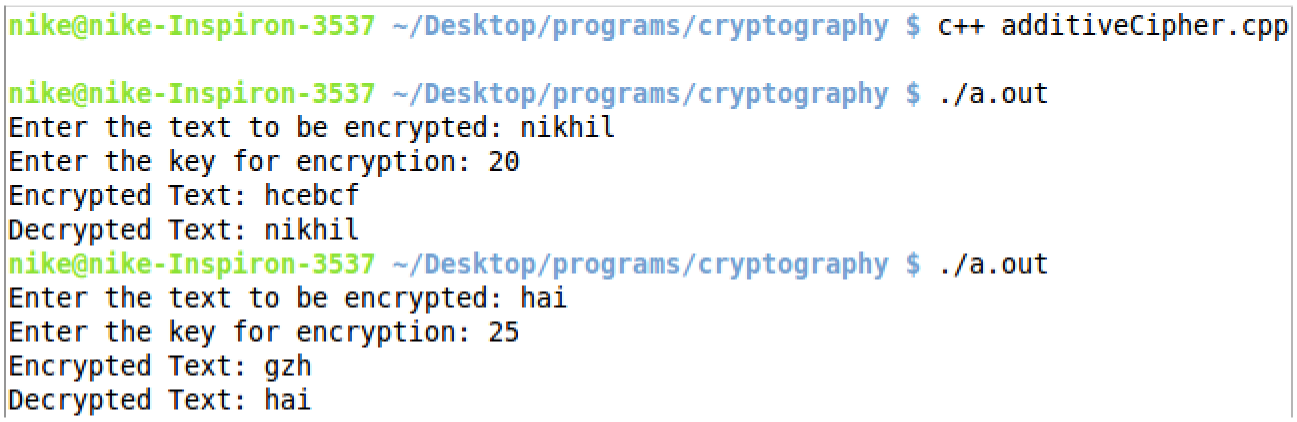
The mono-alphabetic substitution cipher provides the simplest form of cryptography, where the cipher alphabet is simply a rearrangement of the plaintext alphabet. In an additive cipher, the cipher alphabet is a shift of the plaintext alphabet.

Program:

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

// Encrypt using additive cipher
char* additiveCipherEncrypt(char input[], int key){
        int i,inputLen=strlen(input);
        char *output=(char *)malloc((inputLen+1)*sizeof(char));

        for(i=0;i<inputLen;i++){
                output[i]=(input[i]-97+key)%26+97;
        }
        output[i]='\0';
        return output;
}

// Decrypt Text using additive cipher
char* additiveCipherDecrypt(char input[],int key){
        int i,inputLen=strlen(input);
        char *output=(char *)malloc((inputLen+1)*sizeof(char));

        for(i=0;i<inputLen;i++){
                output[i]=(input[i]-97-key+26)%26+97;
        }
        output[i]='\0';
        return output;
}

int main(){
        int key;
        char input[1000],*encryptText,*decryptText;

        // input
        printf("Enter the text to be encrypted: ");
        scanf("%s",input);
```

```
	printf("Enter the key for encryption: ");
	scanf("%d",&key);

	// encryption and decryption function
	encryptText=additiveCipherEncrypt(input,key);
	printf("Encrypted Text: %s\n",encryptText);

	decryptText=additiveCipherDecrypt(encryptText,key);
	printf("Decrypted Text: %s\n",decryptText);

	// free the allocated dynamic memory
	free(encryptText);
	free(decryptText);
	return 0;
}
```

Output:



```
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ c++ additiveCipher.cpp

nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
Enter the text to be encrypted: nikhil
Enter the key for encryption: 20
Encrypted Text: hcebcf
Decrypted Text: nikhil
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
Enter the text to be encrypted: hai
Enter the key for encryption: 25
Encrypted Text: gzh
Decrypted Text: hai
```

PROGRAM 2

MULTIPLICATIVE CIPHER

Description:

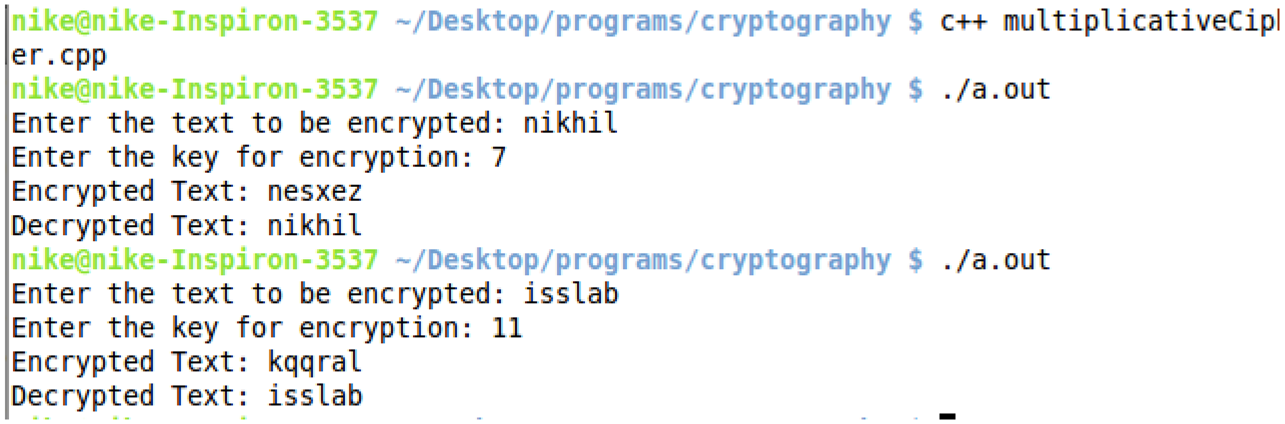
The mono-alphabetic substitution cipher provides the simplest form of cryptography, where the cipher alphabet is simply a rearrangement of the plaintext alphabet. In a multiplicative cipher, the cipher alphabet is a multiple of the plaintext alphabet. It adds a little more security than additive cipher.

Program:

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

// gcd function
int gcd(int a,int b)
{
	if(b==0)
		return a;
	return gcd(b,a%b);
}

// extended eulcidean
pair<int,int> extendedeuclidean(int r1,int r2,int s1,int s2,int t1,int t2){
	if(r2==0){
		pair<int,int> x;
		x.first = s1;
		x.second = t1;
		return x;
	}
	return extendedeuclidean(r2, r1%r2, s2, s1-r1/r2*s2, t2, t1-r1/r2*t2);
}

// multiplicative inverse returns -1 if it doesnot exist
int multiplicativeInverse(int a,int n){
	int inverse;
	if(gcd(n,a)==1){
		inverse=(extendedeuclidean(a,n,1,0,0,1).first)%n;
		if(inverse<0){
			inverse=inverse+n;
		}
		return inverse;
	}
	else{
			return -1;
		}
}

// Encrypt using multiplicative cipher
char* multiplicativeCipherEncrypt(char input[], int key){
		int i,inputLen=strlen(input);
		char *output=(char *)malloc((inputLen+1)*sizeof(char));

		// encryption
		for(i=0;i<inputLen;i++){
			output[i]=((input[i]-97)*key)%26+97;
		}
		output[i]='\0';
		return output;
}

// Decrypt Text using multiplicative cipher
char* multiplicativeCipherDecrypt(char input[],int key){
		int i, inputLen=strlen(input), keyInverse;
		char *output=(char *)malloc((inputLen+1)*sizeof(char));

		// find the inverse key
		keyInverse=multiplicativeInverse(key, 26);

		// decryption
		for(i=0;i<inputLen;i++){
			output[i]=((input[i]-97)*keyInverse)%26+97;
		}
		output[i]='\0';
		return output;
}

int main(){
		int key;
		char input[1000],*encryptText,*decryptText;

		// input
		printf("Enter the text to be encrypted: ");
		scanf("%s",input);
		printf("Enter the key for encryption: ");
		scanf("%d",&key);
		if(gcd(key, 26)!=1){
			printf("key cannot be used for encryption\n");
			return 0;
		}
```

```
        // encryption and decryption function
        encryptText=multiplicativeCipherEncrypt(input,key);
        printf("Encrypted Text: %s\n",encryptText);

        decryptText=multiplicativeCipherDecrypt(encryptText,key);
        printf("Decrypted Text: %s\n",decryptText);

        // free the allocated dynamic memory
        free(encryptText);
        free(decryptText);
        return 0;
}
```

Output:



```
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ c++ multiplicativeCiph
er.cpp
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
Enter the text to be encrypted: nikhil
Enter the key for encryption: 7
Encrypted Text: nesxez
Decrypted Text: nikhil
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
Enter the text to be encrypted: isslab
Enter the key for encryption: 11
Encrypted Text: kqqral
Decrypted Text: isslab
```

PROGRAM 3

STEGNOGRAPHY

Description:

Steganography is the art of covered or hidden writing. The purpose of steganography is covert communication-to hide the existence of a message from a third party. It is directed at forensic computer examiners who need a practical understanding of steganography without delving into the mathematics, although references are provided to some of the ongoing research for the person who needs or wants additional detail.

Although this paper provides a historical context for steganography, the emphasis is on digital applications, focusing on hiding information in online image or audio files. Examples of software tools that employ steganography to hide data inside of other files as well as software to detect such hidden files will also be presented.

Program:

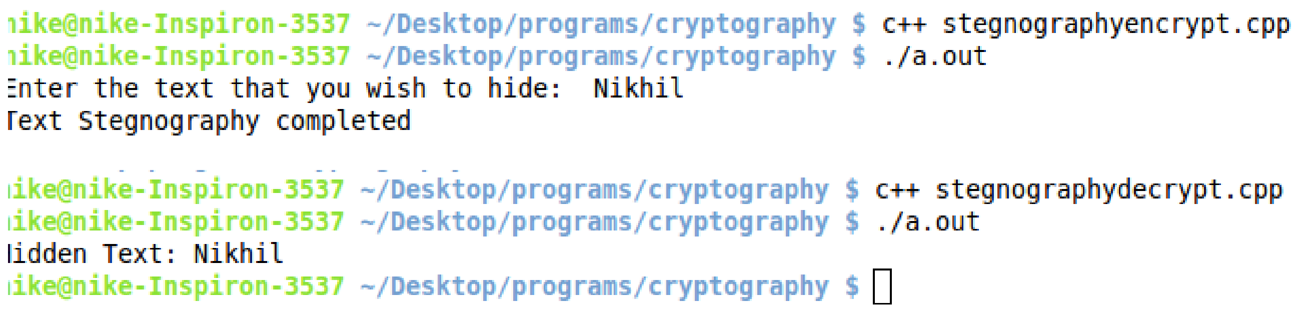
Encryption:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
        // Input/Output files
        FILE *input=fopen("stegnographyinput","r");
        FILE *output=fopen("stegnographyoutput","w");

        char hidden[1000],a[1000];
        int temp,binary[8],i,j;

        // input text to be hidden
        printf("Enter the text that you wish to hide:  ");
        scanf("%[^'\n']s",hidden);

        //algo
        for(j=0;hidden[j]!='\0';j++)
        {
                // convert the character to bit form
                temp=(int)hidden[j];
                for(i=0;i<8;i++)
                {
                        binary[i]=temp%2;
                        temp=temp/2;
                }
                // if bit is 1 then 2 spaces else 1 space after the word
                for(i=7;i>=0;i--)
                {
                        fscanf(input,"%s",a);
                        fprintf(output,"%s",a);
                        if(binary[i]==0)
                        {
                                fprintf(output," ");
                        }
                        else
                        {
                                fprintf(output,"  ");
                        }
                }
        }

        //print output to file
        while(fscanf(input,"%s",a)!=-1)
        {
                fprintf(output,"%s ",a);
        }
        printf("Text Stegnography completed\n");

        //close the files
        fclose(input);
        fclose(output);
        return 0;
}
```

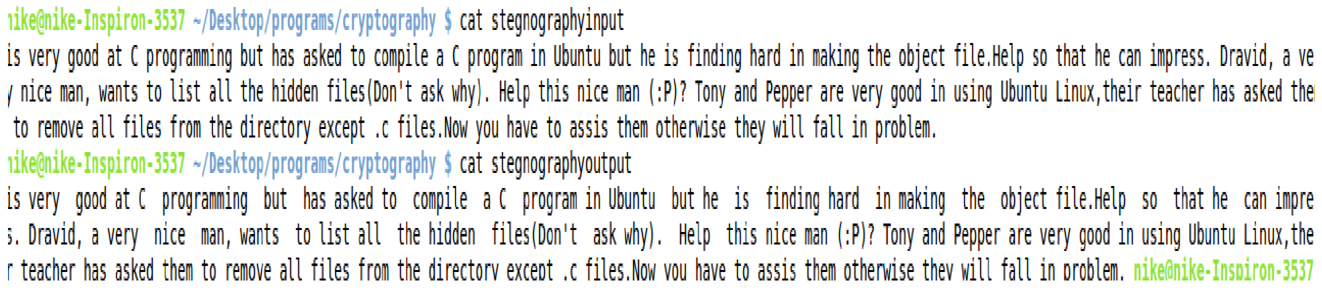
Decryption:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
        FILE *stegnofile=fopen("stegnographyoutput","r");

        char c,output[10000],top=0;
        int ans,count=0;

        //if single space b/w words then bit is 1 else bit in 0
        c = fgetc(stegnofile);
        while(c!=EOF)
        {
                if(c==' ')
                {
                        c = fgetc(stegnofile);
                        if(c==' ')
                        {
                                ans=ans*2+1;
                        }
                        else
                        {
                                ans=ans*2;
                        }
                        count=(count+1)%8;
                        if(count==0)
                        {
                                output[top]=(char)ans;
                                top++;
                                ans=0;
                        }
                }
                c = fgetc(stegnofile);
        }
        output[top]='\0';
        printf("Hidden Text: %s\n",output);

        //close the file
        fclose(stegnofile);
        return 0;
}
```

Output:



```
ıike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ c++ stegnographyencrypt.cpp
ıike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
Enter the text that you wish to hide:  Nikhil
Text Stegnography completed

ıike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ c++ stegnographydecrypt.cpp
ıike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
lidden Text: Nikhil
ıike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ 
```



```
ıike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ cat stegnographyinput
is very good at C programming but has asked to compile a C program in Ubuntu but he is finding hard in making the object file.Help so that he can impress. Dravid, a ve
y nice man, wants to list all the hidden files(Don't ask why). Help this nice man (:P)? Tony and Pepper are very good in using Ubuntu Linux,their teacher has asked the
 to remove all files from the directory except .c files.Now you have to assis them otherwise they will fall in problem.
ıike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ cat stegnographyoutput
is very  good at C  programming  but  has asked to  compile  a C  program in Ubuntu  but he  is  finding hard  in making  the  object file.Help  so  that he  can impre
s. Dravid, a very  nice  man, wants  to list all  the hidden  files(Don't  ask why).  Help  this nice man (:P)? Tony and Pepper are very good in using Ubuntu Linux,the
r teacher has asked them to remove all files from the directory except .c files.Now you have to assis them otherwise they will fall in problem. nike@nike-Inspiron-3537
```

PROGRAM 4

PLAYFAIR CIPHER

Description:

The technique encrypts pairs of letters (digraphs), instead of single letters as in the simple substitution cipher. The Playfair 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[1] possible digraphs rather than the 26 possible monographs.

The Playfair 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 omitting "Q" to reduce the alphabet to fit; other versions put 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 center. 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, "HelloWorld" becomes "HE LL OW OR LD", and map them out on the key table. If needed, append a "Z" 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 plaintext:

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. Some variants of Playfair use "Q" instead of "X", but any uncommon monograph will do.
2. If the letters appear on the same row of your table, replace them with the letters to their immediate right respectively (wrapping around to the left side of the row if a letter in the original pair was on the right side of the row).
3. If the letters appear on the same column of your table, replace them with the letters immediately below respectively (wrapping around to the top side of the column if a letter in the original pair was on the bottom side of the column).
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. The order is important – the first letter of the encrypted pair is the one that lies on the same row as the first letter of the plaintext pair.

To decrypt, use the INVERSE (opposite) of the last 3 rules, and the 1st as-is (dropping any extra "X"s (or "Q"s) that do not make sense in the final message when finished).

Program:

Encryption:

```
#include<stdio.h>
#include<malloc.h>

//Generates the matrix using encryption is done
int generatematrix(char matrix[5][5],char str[])
{
	int i,check[26],counter=0;

	// initialise the matrix
	for(i=0;i<26;i++)
	{
		check[i]=0;
	}

	// input the key to the matrix
	for(i=0;str[i]!='\0';i++)
	{
		if(check[str[i]-97]==1)
		{
			continue;
		}
		else
		{
			check[str[i]-97]=1;
			matrix[counter/5][counter%5]=str[i];
			counter++;
			if(check[8]==1||check[9]==1)
			{
				check[8]=1;
				check[9]=1;
			}
		}
	}
	check[9]=1;

	// put the remaining characters to the matrix
	for(i=0;i<26;i++)
	{
		if(check[i]==0)
		{
			check[i]=1;
			matrix[counter/5][counter%5]=i+97;
			counter++;
```

```
                }
        }
        return 0;
}

// pair structure representing the coordinates in the matrix
struct pair
{
        int row,col;
};

// searches a character in the matrix
struct pair* search(char c, char matrix[5][5])
{
        int i,j;
        struct pair *x=malloc(sizeof(struct pair));
        for(i=0;i<5;i++)
        {
                for(j=0;j<5;j++)
                {
                        if(c==matrix[i][j]||(c=='i'&&matrix[i][j]=='j')||(c=='j'&&matrix[i][j]=='i'))
                        {
                                x->row=i;
                                x->col=j;
                                break;
                        }
                }
        }
        return x;
}

int main()
{
        char str[100],matrix[5][5],a1,a2,end;
        int i,j;
        struct pair *coor1,*coor2;

        // file for input/output
        FILE *input=fopen("playfaircipherinput","r");
        FILE *output=fopen("playfaircipheroutput","w");

        //input string as the key
        printf("Enter the key for encryption of lowercase characters: ");
        scanf("%s",str);

        // generate the matrix and print it
```

```
generatematrix(matrix,str);
for(i=0;i<5;i++)
{
        for(j=0;j<5;j++)
        {
                printf("%c ",matrix[i][j]);
        }
        printf("\n");
}

// do the encryption pairwise
while(1)
{
        a1=' ';
        //ignore spaces in input
        while(a1==' ')
        {
                end=fscanf(input,"%c",&a1);
                if(end==-1)
                        break;
        }
        if(end==-1)
                break;

        a2=' ';
        // ignore spaces in input
        while(a2==' ')
        {
                end=fscanf(input,"%c",&a2);
                if(end==-1)
                {
                        a2='x';
                }
        }

        // if both characters are same, change 1 to bogus character
        if(a1==a2)
                a2='x';

        coor1=search(a1,matrix);
        coor2=search(a2,matrix);

        // output encrypted text pairwise
        if(coor1->row==coor2->row)
        {
```

```
				printf("%c%c -> %c%c\n",matrix[coor1->row][coor1->col],matrix[coor2->row][coor2->col],matrix[coor1->row][(coor1->col+1)%5],matrix[coor1->row][(coor2->col+1)%5] );
				fprintf(output,"%c%c",matrix[coor1->row][(coor1->col+1)%5],matrix[coor1->row][(coor2->col+1)%5]);
			}
			else if(coor1->col==coor2->col)
			{
				printf("%c%c -> %c%c\n",matrix[coor1->row][coor1->col],matrix[coor2->row][coor2->col],matrix[(coor1->row+1)%5][coor1->col],matrix[(coor2->row+1)%5][coor1->col] );
				fprintf(output,"%c%c",matrix[(coor1->row+1)%5][coor1->col],matrix[(coor2->row+1)%5][coor1->col]);
			}
			else
			{
				printf("%c%c -> %c%c\n",matrix[coor1->row][coor1->col],matrix[coor2->row][coor2->col],matrix[coor1->row][coor2->col],matrix[coor2->row][coor1->col] );
				fprintf(output,"%c%c",matrix[coor1->row][coor2->col],matrix[coor2->row][coor1->col]);
			}
		}
		return 0;
}
```

Decryption:

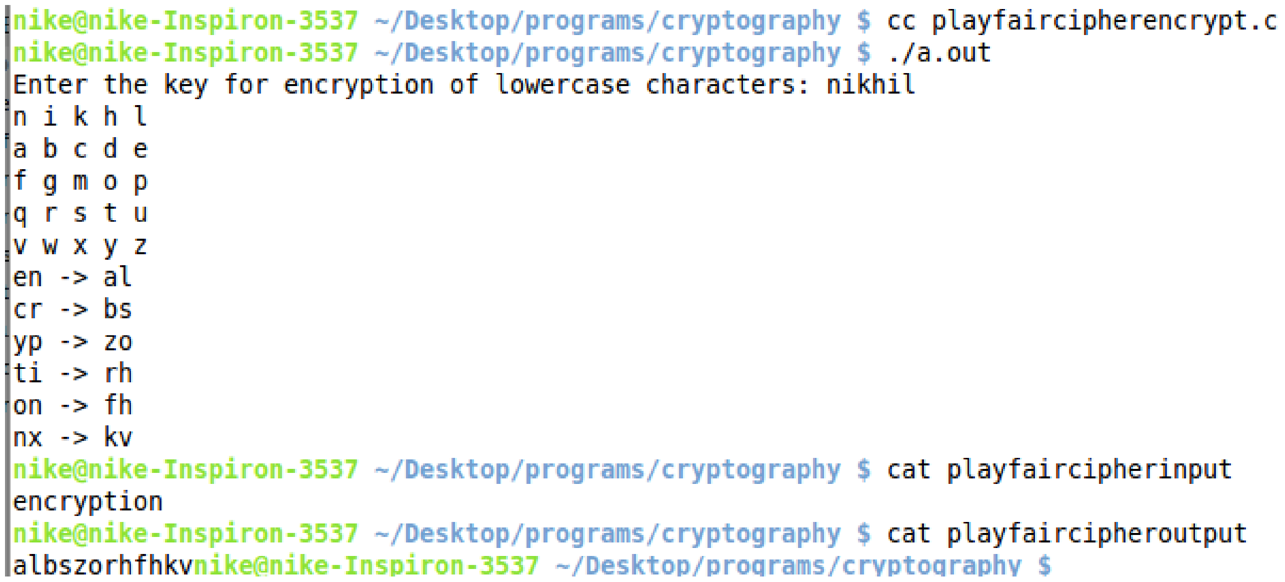
```
#include<stdio.h>
#include<malloc.h>

//Generates the matrix using encryption is done
int generatematrix(char matrix[5][5],char str[])
{
	int i,check[26],counter=0;

	// initialise matrix
	for(i=0;i<26;i++)
	{
		check[i]=0;
	}

	// put the key in the matrix
	for(i=0;str[i]!='\0';i++)
	{
		if(check[str[i]-97]==1)
		{
			continue;
		}
		else
```

```
			{
				check[str[i]-97]=1;
				matrix[counter/5][counter%5]=str[i];
				counter++;
				if(check[8]==1||check[9]==1)
				{
					check[8]=1;
					check[9]=1;
				}
			}
		}
		check[9]=1;

		// put the remaining characters to the matrix
		for(i=0;i<26;i++)
		{
			if(check[i]==0)
			{
				check[i]=1;
				matrix[counter/5][counter%5]=i+97;
				counter++;
			}
		}
		return 0;
}

// pair structure representing the coordinates in the matrix
struct pair
{
		int row,col;
};

// searches a character in the generated matrix
struct pair* search(char c,char matrix[5][5])
{
		int i,j;
		struct pair *x=malloc(sizeof(struct pair));
		for(i=0;i<5;i++)
		{
			for(j=0;j<5;j++)
			{
				if(c==matrix[i][j]||(c=='i'&&matrix[i][j]=='j')||(c=='j'&&matrix[i][j]=='i'))
				{
					x->row=i;
					x->col=j;
					break;
```

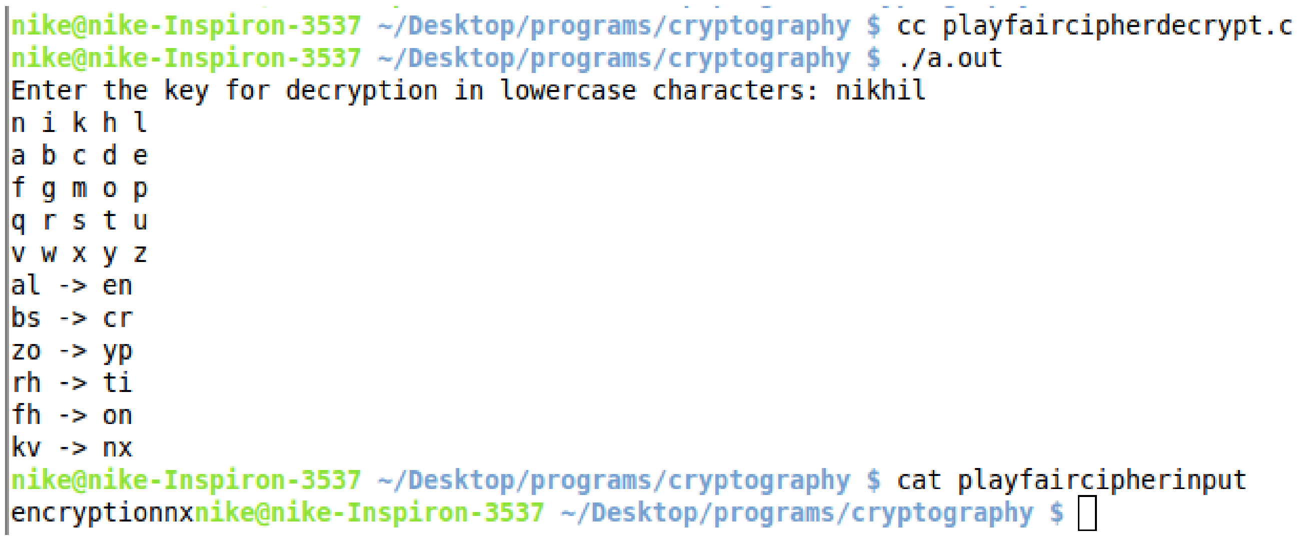
```
                }
            }
        }
        return x;
}

int main()
{
        char str[100],matrix[5][5],a1,a2,end;
        int i,j;
        struct pair *coor1,*coor2;

        // file containing the text to be encrypted
        FILE *input=fopen("playfaircipheroutput","r");
        FILE *output=fopen("playfaircipherinput","w");

        // input string as key for decryption
        printf("Enter the key for decryption in lowercase characters: ");
        scanf("%s",str);

        // generate and print the matrix
        generatematrix(matrix,str);
        for(i=0;i<5;i++)
        {
            for(j=0;j<5;j++)
            {
                printf("%c ",matrix[i][j]);
            }
            printf("\n");
        }

        // decrypt the text pairwise
        while(1)
        {
            a1=' ';
            // ignore spaces
            while(a1==' ')
            {
                end=fscanf(input,"%c",&a1);
                if(end==-1)
                    break;
            }
            if(end==-1)
                break;

            a2=' ';
```

```
				// ignore spaces
				while(a2==' ')
				{
					end=fscanf(input,"%c",&a2);
					if(end==-1)
					{
						a2='x';
					}
				}
				if(a1==a2)
					a2='x';

				coor1=search(a1,matrix);
				coor2=search(a2,matrix);

				// print the decrypted text
				if(coor1->row==coor2->row)
				{
					printf("%c%c -> %c%c\n",matrix[coor1->row][coor1->col],matrix[coor2-
>row][coor2->col],matrix[coor1->row][(coor1->col-1+5)%5],matrix[coor1->row][(coor2->col-1+5)%5] );
					fprintf(output,"%c%c",matrix[coor1->row][(coor1->col-1+5)%5],matrix[coor1-
>row][(coor2->col-1+5)%5]);
				}
				else if(coor1->col==coor2->col)
				{
					printf("%c%c -> %c%c\n",matrix[coor1->row][coor1->col],matrix[coor2-
>row][coor2->col],matrix[(coor1->row-1+5)%5][coor1->col],matrix[(coor2->row-1+5)%5][coor1->col] );
					fprintf(output,"%c%c",matrix[(coor1->row-1+5)%5][coor1->col],matrix[(coor2-
>row-1+5)%5][coor1->col]);
				}
				else
				{
					printf("%c%c -> %c%c\n",matrix[coor1->row][coor1->col],matrix[coor2-
>row][coor2->col],matrix[coor1->row][coor2->col],matrix[coor2->row][coor1->col] );
					fprintf(output,"%c%c",matrix[coor1->row][coor2->col],matrix[coor2->row][coor1-
>col]);
				}
		}
		return 0;
}
```

Output:



```
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ cc playfaircipherencrypt.c
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
Enter the key for encryption of lowercase characters: nikhil
n i k h l
a b c d e
f g m o p
q r s t u
v w x y z
en -> al
cr -> bs
yp -> zo
ti -> rh
on -> fh
nx -> kv
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ cat playfaircipherinput
encryption
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ cat playfaircipheroutput
albszorhfhkvnike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $
```



```
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ cc playfaircipherdecrypt.c
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
Enter the key for decryption in lowercase characters: nikhil
n i k h l
a b c d e
f g m o p
q r s t u
v w x y z
al -> en
bs -> cr
zo -> yp
rh -> ti
fh -> on
kv -> nx
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ cat playfaircipherinput
encryptionnxnike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $
```

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 plaintext 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 cipherer writes out:

```
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
```

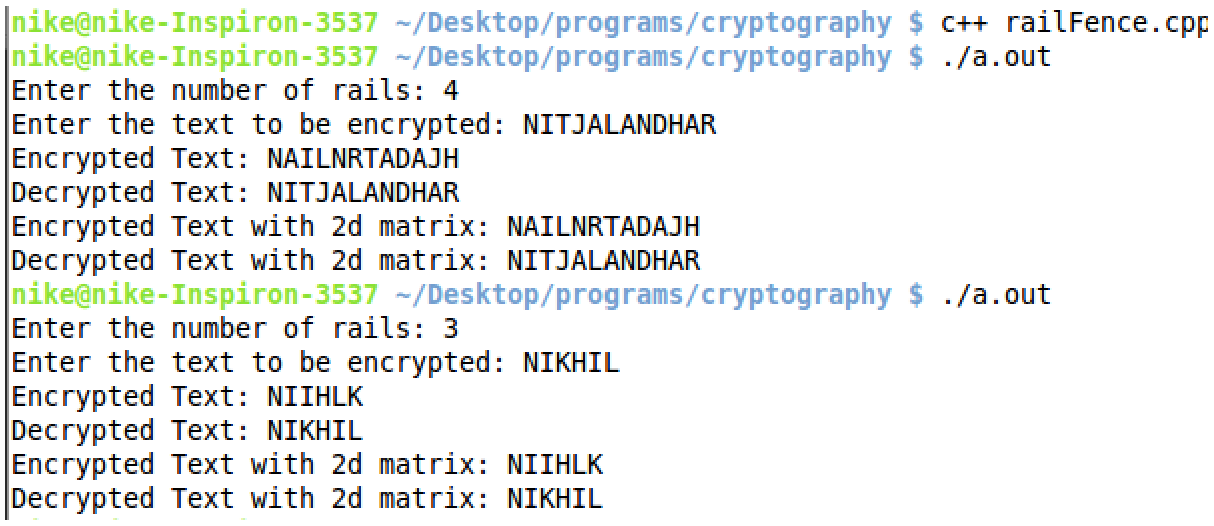
Program:

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

// Encrypt using Rail fence
char* railFenceEncrypt(char input[], int noRails){
	int i,j,dist[2],inputLen=strlen(input),distIndex,outputIndex;
	char *output=(char *)malloc((inputLen+1)*sizeof(char));

	outputIndex=0;
	for(i=0;i<noRails;i++){
		// find the distances of next character in a rail
		dist[0]=2*(noRails-i-1);
		dist[1]=2*(i);
		if(i==0||i==noRails-1){
			dist[0]=2*(noRails-1);
			dist[1]=2*(noRails-1);
		}

		// put the input string in the rails according to the distance space
		j=i;
```

```
                distIndex=0;
                while(j<inputLen){
                        output[outputIndex]=input[j];
                        outputIndex++;
                        j=j+dist[distIndex];
                        distIndex=(distIndex+1)%2;
                }
        }
        output[outputIndex]='\0';
        return output;
}

// Decrypt Text using RailFence
char* railFenceDecrypt(char input[],int noRails){
        int i,j,inputLen=strlen(input),inputIndex=0,dist[2],distIndex;
        char *output=(char *)malloc((inputLen+1)*sizeof(char));

        for(i=0;i<noRails;i++){
                // find the distances of next character in a rail
                dist[0]=2*(noRails-i-1);
                dist[1]=2*(i);
                if(i==0||i==noRails-1){
                        dist[0]=2*(noRails-1);
                        dist[1]=2*(noRails-1);
                }

                // get the output string by traversing the rails
                j=i;
                distIndex=0;
                while(j<inputLen){
                        output[j]=input[inputIndex];
                        inputIndex++;
                        j=j+dist[distIndex];
                        distIndex=(distIndex+1)%2;
                }
        }
        output[inputIndex]='\0';
        return output;
}

// Encrypt Text using Rail fence in 2d
char* railFenceEncryptWith2d(char input[],int noRails){
        int i,j,inputLen=strlen(input),inputIndex,outputIndex;
        int railMatrix[noRails][inputLen+1];
        char *output=(char *)calloc(sizeof(char),(inputLen+1));
        memset(railMatrix, 0, sizeof(int)*(inputLen+1)*noRails);
        // put the input string in the 2d matrix
        inputIndex=0;
        while(inputIndex<inputLen){
                for(i=0;i<noRails&&inputIndex<inputLen;i++){
                        railMatrix[i][inputIndex]=input[inputIndex];
                        inputIndex++;
                }
                for(i=noRails-2;i>0&&inputIndex<inputLen;i--){
                        railMatrix[i][inputIndex]=input[inputIndex];
                        inputIndex++;
                }
        }

        // read the matrix row-wise to get the encrypted text
        outputIndex=0;
        for(i=0;i<noRails;i++){
                for(j=0;j<inputLen;j++){
                        if(railMatrix[i][j]!=0){
                                output[outputIndex]=railMatrix[i][j];
                                outputIndex++;
                        }
                }
        }
        output[outputIndex]='\0';
        return output;
}

// Decrypt Text using RailFence in 2d
char* railFenceDecryptWith2d(char input[],int noRails){
        int i,j,inputLen=strlen(input),inputIndex,outputIndex;
        int railMatrix[noRails][inputLen+1];
        char *output=(char *)calloc(sizeof(char),(inputLen+1));
        memset(railMatrix, 0, sizeof(int)*(inputLen+1)*noRails);

        // make the pattern i.e. mark the places where character will be written
        inputIndex=0;
        while(inputIndex<inputLen){
                for(i=0;i<noRails&&inputIndex<inputLen;i++){
                        railMatrix[i][inputIndex]=1;
                        inputIndex++;
                }
                for(i=noRails-2;i>0&&inputIndex<inputLen;i--){
                        railMatrix[i][inputIndex]=1;
                        inputIndex++;
                }
```

```
	}

	// put the text at the marked places in the matrix row-wise
	inputIndex=0;
	for(i=0;i<noRails;i++){
		for(j=0;j<inputLen;j++){
			if(railMatrix[i][j]!=0){
				railMatrix[i][j]=input[inputIndex];
				inputIndex++;
			}
		}
	}

	// read column-wise to get the decrypted text
	outputIndex=0;
	while(outputIndex<inputLen){
		for(i=0;i<noRails&&outputIndex<inputLen;i++){
			output[outputIndex]=railMatrix[i][outputIndex];
			outputIndex++;
		}
		for(i=noRails-2;i>0&&outputIndex<inputLen;i--){
			output[outputIndex]=railMatrix[i][outputIndex];
			outputIndex++;
		}
	}
	output[outputIndex]='\0';
	return output;
}

int main(){
	int noRails;
	char input[1000],*encryptText,*decryptText;

	// input
	printf("Enter the number of rails: ");
	scanf("%d",&noRails);
	printf("Enter the text to be encrypted: ");
	scanf("%s",input);

	// Using 1d array in encryption and decryption function
	encryptText=railFenceEncrypt(input,noRails);
	printf("Encrypted Text: %s\n",encryptText);

	decryptText=railFenceDecrypt(encryptText,noRails);
	printf("Decrypted Text: %s\n",decryptText);
	// free the allocated dynamic memory
	free(encryptText);
	free(decryptText);

	// Using 2d array in encryption and decryption function
	encryptText=railFenceEncryptWith2d(input,noRails);
	printf("Encrypted Text with 2d matrix: %s\n",encryptText);

	decryptText=railFenceDecryptWith2d(encryptText, noRails);
	printf("Decrypted Text with 2d matrix: %s\n",decryptText);

	// free the allocated dynamic memory
	free(encryptText);
	free(decryptText);
	return 0;
}
```

Output:



```
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ c++ railFence.cpp
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
Enter the number of rails: 4
Enter the text to be encrypted: NITJALANDHAR
Encrypted Text: NAILNRTADAJH
Decrypted Text: NITJALANDHAR
Encrypted Text with 2d matrix: NAILNRTADAJH
Decrypted Text with 2d matrix: NITJALANDHAR
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
Enter the number of rails: 3
Enter the text to be encrypted: NIKHIL
Encrypted Text: NIIHLK
Decrypted Text: NIKHIL
Encrypted Text with 2d matrix: NIIHLK
Decrypted Text with 2d matrix: NIKHIL
```

PROGRAM 6

COLUMNAR CIPHER

Description:

In a columnar transposition, the message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order. Both the width of the rows and the permutation of the columns are usually defined by a keyword. For example, the word ZEBRAS is of length 6 (so the rows are of length 6), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be "6 3 2 4 1 5".

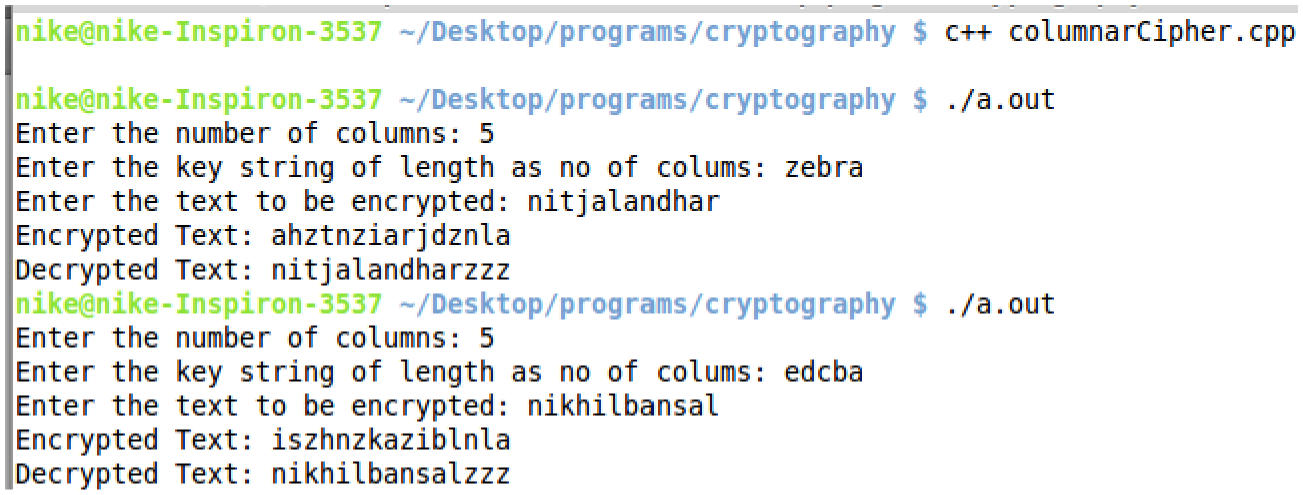
Program:

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

// Encrypt Text using columnar cipher
char* columnarEncrypt(char input[],int noCol, char key[]){
        int i, j, inputLen=strlen(input), outputIndex, keyIndex=-1, prevKey=-1, minKey, noRow;
        int colMatrix[inputLen+1][noCol];
        char *output=(char *)calloc(sizeof(char),(inputLen+1));
        memset(colMatrix, 0, sizeof(int)*(inputLen+1)*noCol);

        // fill the matrix with input
        for(i=0;i<inputLen;i++){
                colMatrix[i/noCol][i%noCol]=input[i];
        }
        noRow=i/noCol+1;
        for(j=i%noCol;j<noCol;j++){
                colMatrix[noRow-1][j]='z';
        }

        // fill the output array
        outputIndex=0;
        for(i=0;i<noCol;i++){
                minKey=1000;
                for(j=0;j<noCol;j++){
                        if(key[j]<=prevKey)
                                continue;
                        if(key[j]<minKey){
                                minKey=key[j];
                                keyIndex=j;
                        }
                }
                prevKey=minKey;
                for(j=0;j<noRow;j++){
                        output[outputIndex]=colMatrix[j][keyIndex];
                        outputIndex++;
                }
```

```
		}
		output[outputIndex]='\0';
		return output;
}

// Decrypt Text using columnar cipher
char* columnarDecrypt(char input[],int noCol, char key[]){
		int i, j, inputLen=strlen(input), inputIndex, keyIndex=-1, prevKey=-1, minKey, noRow;
		int colMatrix[inputLen+1][noCol];
		noRow=inputLen/noCol;
		char *output=(char *)calloc(sizeof(char),(noRow*noCol)+1);
		memset(colMatrix, 0, sizeof(int)*(inputLen+1)*noCol);

		// fill the matrix
		inputIndex=0;
		for(i=0;i<noCol;i++){
				minKey=1000;
				for(j=0;j<noCol;j++){
						if(key[j]<=prevKey)
								continue;
						if(key[j]<minKey){
								minKey=key[j];
								keyIndex=j;
						}
				}
				prevKey=minKey;
				for(j=0;j<noRow;j++){
						colMatrix[j][keyIndex]=input[inputIndex];
						inputIndex++;
				}
		}

		// fill the output matrix
		for(i=0;i<noRow*noCol;i++){
				output[i]=colMatrix[i/noCol][i%noCol];
		}
		output[i]='\0';
		return output;
}

int main(){
		int noCol;
		char input[1000],*encryptText,*decryptText,key[1000];

		// input
		printf("Enter the number of columns: ");
		scanf("%d",&noCol);
		printf("Enter the key string of length as no of colums: ");
		scanf("%s", key);
		if(strlen(key)!=noCol){
				printf("Key length is not equal to number of columns\n");
				return 0;
```

```
        }
        printf("Enter the text to be encrypted: ");
        scanf("%s",input);

        // Using encryption and decryption function
        encryptText=columnarEncrypt(input, noCol, key);
        printf("Encrypted Text: %s\n",encryptText);

        decryptText=columnarDecrypt(encryptText, noCol, key);
        printf("Decrypted Text: %s\n",decryptText);

        // free the allocated dynamic memory
        free(encryptText);
        free(decryptText);
        return 0;
}
```

Output:



```
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ c++ columnarCipher.cpp

nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
Enter the number of columns: 5
Enter the key string of length as no of colums: zebra
Enter the text to be encrypted: nitjalandhar
Encrypted Text: ahztnziarjdznla
Decrypted Text: nitjalandharzzz
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
Enter the number of columns: 5
Enter the key string of length as no of colums: edcba
Enter the text to be encrypted: nikhilbansal
Encrypted Text: iszhnzkaziblnla
Decrypted Text: nikhilbansalzzz
```

PROGRAM 7

EUCLIDEAN GCD

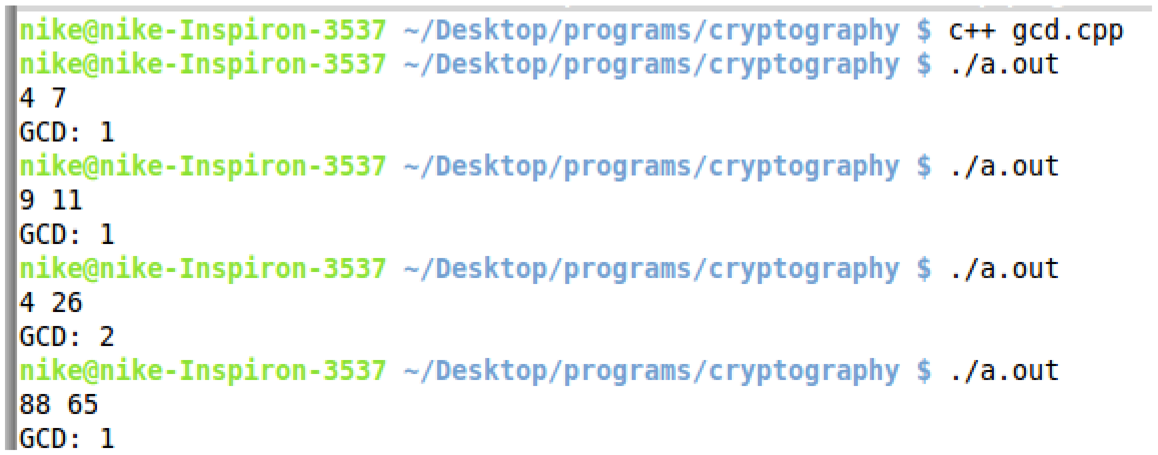
Description:

In mathematics, the greatest common divisor (gcd) of two or more integers, when at least one of them is not zero, is the largest positive integer that divides the numbers without a remainder. For example, the GCD of 8 and 12 is 4.

Program:

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

// gcd function
int gcd(int a,int b)
{
	if(b==0)
		return a;
	return gcd(b,a%b);
}

int main()
{
	int a,b,x;
	scanf("%d%d",&a,&b);
	x=gcd(a,b);
	printf("GCD: %d\n",x);
}
```

Output:



```
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ c++ gcd.cpp
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
4 7
GCD: 1
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
9 11
GCD: 1
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
4 26
GCD: 2
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
88 65
GCD: 1
```

PROGRAM 8

EXTENDED EUCLIDEAN

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

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

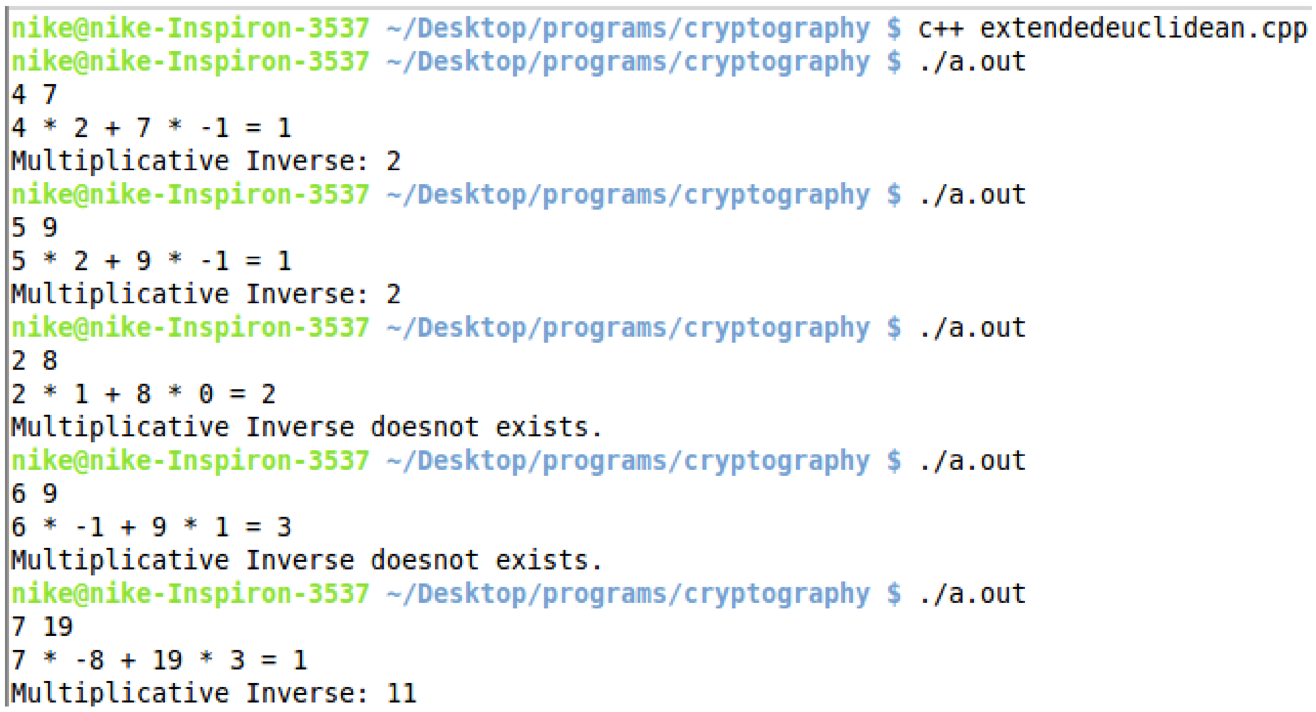
The extended Euclidean algorithm is particularly useful when a and b are coprime, since x is the modular multiplicative inverse of a modulo b, and y is the modular multiplicative inverse of b modulo a. Similarly, the polynomial extended Euclidean algorithm allows one to compute the multiplicative inverse in algebraic field extensions and, in particular in finite fields of non prime order. 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;

pair<int,int> extendedeuclidean(int r1,int r2,int s1,int s2,int t1,int t2){
	if(r2==0){
		pair<int,int> x;
		x.first = s1;
		x.second = t1;
		return x;
	}
	return extendedeuclidean(r2, r1%r2, s2, s1-r1/r2*s2, t2, t1-r1/r2*t2);
}

int gcd(int a,int b){
	if(b==0)
		return a;
	return gcd(b,a%b);
}

int main(){
	int a,b,result_gcd,multiplicative_inverse;
	pair<int,int> result;

	// input
	scanf("%d%d",&a,&b);
```

```
        result = extendedeuclidean(a,b,1,0,0,1);
        result_gcd = gcd(a,b);

        printf("%d * %d + %d * %d = %d\n",a,result.first,b,result.second,result_gcd);

        // multiplicative inverse
        if(result_gcd==1){
                multiplicative_inverse = result.first%b;
                while(multiplicative_inverse<0)
                {
                        multiplicative_inverse = multiplicative_inverse+b;
                }
                multiplicative_inverse = multiplicative_inverse%b;
                printf("Multiplicative Inverse: %d\n",multiplicative_inverse);
        }
        else{
                printf("Multiplicative Inverse doesnot exists.\n");
        }
        return 0;
}
```

Output:



```
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ c++ extendedeuclidean.cpp
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
4 7
4 * 2 + 7 * -1 = 1
Multiplicative Inverse: 2
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
5 9
5 * 2 + 9 * -1 = 1
Multiplicative Inverse: 2
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
2 8
2 * 1 + 8 * 0 = 2
Multiplicative Inverse doesnot exists.
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
6 9
6 * -1 + 9 * 1 = 3
Multiplicative Inverse doesnot exists.
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
7 19
7 * -8 + 19 * 3 = 1
Multiplicative Inverse: 11
```

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 in the 3rd to 5th centuries by 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. For example, what is the lowest number n that when divided by 3 leaves a remainder of 2, when divided by 5 leaves a remainder of 3, and when divided by 7 leaves a remainder of 2?

Suppose n_1, n_2, …, n_k are positive integers that are pairwise coprime. Then, for any given sequence of integers a_1,a_2, …, a_k, there exists an integer x solving the following system of simultaneous congruences.

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

Furthermore, all solutions x of this system are congruent modulo the product, $N = n_1 n_2 \ldots n_k$.

Hence $x \equiv y \pmod{n_i}$ for all $1 \leq i \leq k$, if and only if $x \equiv y \pmod{N}$.

Sometimes, the simultaneous congruences can be solved even if the n_i's are not pairwise coprime. A solution x exists if and only if:

$$a_i \equiv a_j \pmod{\gcd(n_i, n_j)} \qquad \text{for all } i \text{ and } j$$

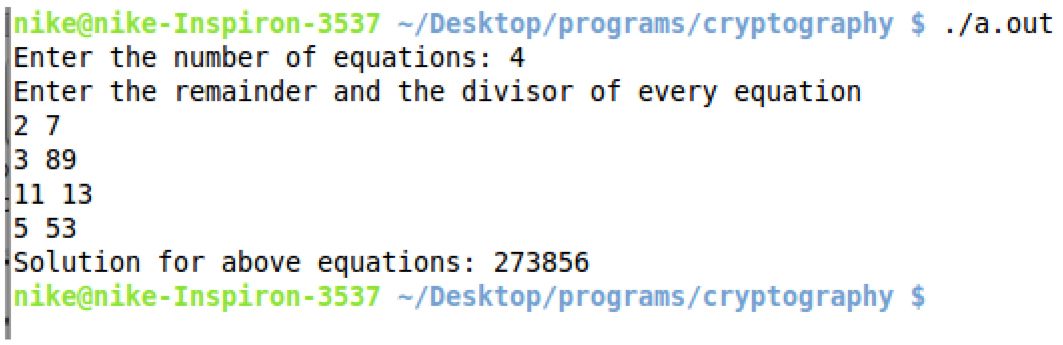
All solutions x are then congruent modulo the least common multiple of the n_i.

Program:

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

// gcd function
int gcd(int a,int b)
{
        if(b==0)
                return a;
        return gcd(b,a%b);
}

// extended eulcidean
```

```
pair<int,int> extendedeuclidean(int r1,int r2,int s1,int s2,int t1,int t2){
	if(r2==0){
		pair<int,int> x;
		x.first = s1;
		x.second = t1;
		return x;
	}
	return extendedeuclidean(r2, r1%r2, s2, s1-r1/r2*s2, t2, t1-r1/r2*t2);
}

// multiplicative inverse returns -1 if it doesnot exist
int multiplicativeInverse(int a,int n){
	int inverse;
	if(gcd(n,a)==1){
		inverse=(extendedeuclidean(a,n,1,0,0,1).first)%n;
		if(inverse<0){
			inverse=inverse+n;
		}
		return inverse;
	}
	else{
		return -1;
	}
}

// chinese remainder theorm returns -1 if no solution
int chineseRemainder(int no_eq,int r[],int divisor[]){
	int i,j,temp,M,m[no_eq],m_inverse[no_eq],result;

	// check if the solution exists by checking if all divisors are coprime to each other
	for(i=0;i<no_eq;i++){
		for(j=0;j<no_eq;j++){
			if(i==j)
				continue;
			temp=gcd(divisor[i],divisor[j]);
			if(temp!=1)
				return -1;
		}
	}

	// find the M
	M=1;
	for(i=0;i<no_eq;i++){
		M=M*divisor[i];
	}

	// find array m
	for(i=0;i<no_eq;i++){
		m[i]=M/divisor[i];
	}

	// find array m_inverse
	for(i=0;i<no_eq;i++){
		m_inverse[i]=multiplicativeInverse(m[i],divisor[i]);
	}

	// get the answer
	result=0;
	for(i=0;i<no_eq;i++){
		result=(result+r[i]*m[i]*m_inverse[i])%M;
	}
	return result;
}

int main(){
	int no_eq,i,r[100],divisor[100],result;

	// input
	printf("Enter the number of equations: ");
	scanf("%d",&no_eq);
	printf("Enter the remainder and the divisor of every equation\n");
	for(i=0;i<no_eq;i++){
		scanf("%d%d",&r[i],&divisor[i]);
	}

	result=chineseRemainder(no_eq,r,divisor);

	// output
	if(result==-1){
		printf("No solution Exists\n");
	}
	else{
		printf("Solution for above equations: %d\n",result);
	}
	return 0;
}
```

Output:



```
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
Enter the number of equations: 4
Enter the remainder and the divisor of every equation
2 7
3 89
11 13
5 53
Solution for above equations: 273856
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $
```

PROGRAM 10

RSA ALGORITHM

Description:

RSA is one of the first practicable public-key cryptosystems and is widely used for secure data transmission. In such a cryptosystem, the encryption key is public and differs from the decryption key which is kept secret. In RSA, this asymmetry is based on the practical difficulty of factoring the product of two large prime numbers, the factoring problem. RSA stands for Ron Rivest, Adi Shamir andLeonard Adleman, who first publicly described the algorithm in 1977. Clifford Cocks, an English mathematician, had developed an equivalent system in 1973, but it wasn't declassified until 1997.

Key generation:

RSA involves a public key and a private key. The public key can be known by everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key. The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers p and q.
   - For security purposes, the integers p and q should be chosen at random, and should be of similar bit-length. Prime integers can be efficiently found using a primality test.
2. Compute n = pq.
   - n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
3. Compute $\varphi(n) = \varphi(p)\varphi(q) = (p - 1)(q - 1) = n - (p + q - 1)$, where φ is Euler's totient function.
4. Choose an integer e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$; i.e., e and φ(n) are coprime.
   - e is released as the public key exponent.
   - e having a short bit-length and small Hamming weight results in more efficient encryption – most commonly 216 + 1 = 65,537. However, much smaller values of e (such as 3) have been shown to be less secure in some settings.[5]
5. Determine d as $d \equiv e-1 \pmod{\varphi(n)}$; i.e., d is the multiplicative inverse of e (modulo φ(n)).
   - This is more clearly stated as: solve for d given $d\cdot e \equiv 1 \pmod{\varphi(n)}$
   - This is often computed using the extended Euclidean algorithm. Using the pseudocode in the Modular integers section, inputs a and n correspond to e and φ(n), respectively.
   - d is kept as the private key exponent.

Encryption:

Alice transmits her public key (n, e) to Bob and keeps the private key secret. Bob then wishes to send message M to Alice.

He first turns M into an integer m, such that $0 \leq m < n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext ccorresponding to

$$c \equiv m^e \pmod{n}$$

This can be done quickly using the method of exponentiation by squaring. Bob then transmits c to Alice.

Note that at least nine values of m will yield a ciphertext c equal to m,[note 1] but this is very unlikely to occur in practice.

Decryption:

Alice can recover m from c by using her private key exponent d via computing

$$m \equiv c^d \pmod{n}$$

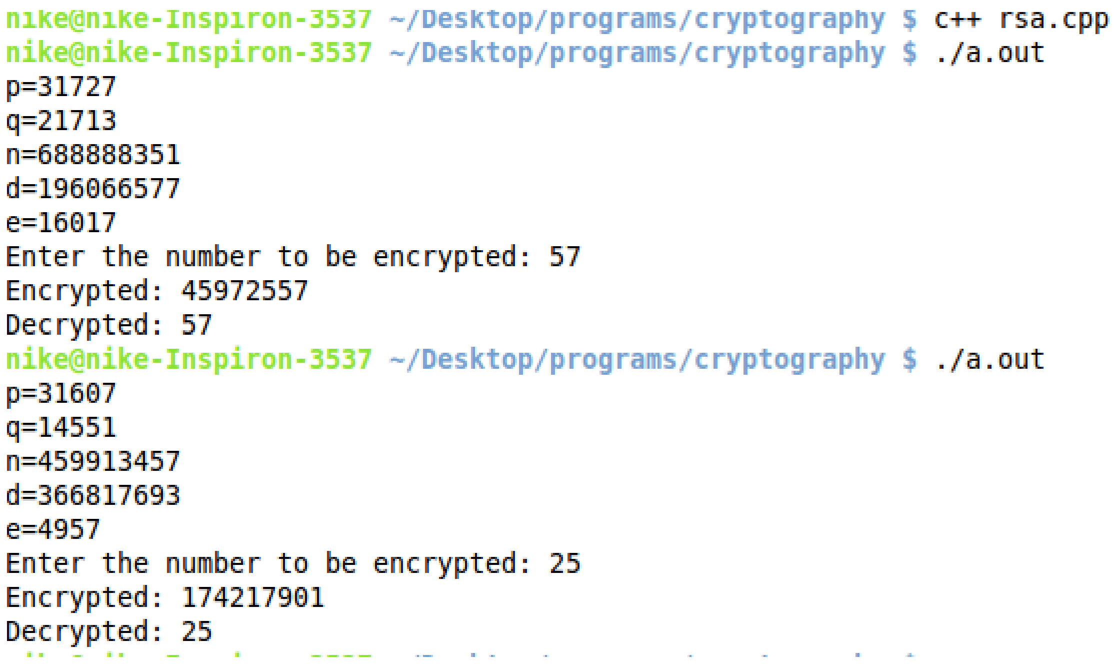
Program:

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

// gcd function
int gcd(int a,int b)
{
	if(b==0)
		return a;
	return gcd(b,a%b);
}

// extended eulcidean
pair<int,int> extendedeuclidean(int r1,int r2,int s1,int s2,int t1,int t2){
	if(r2==0){
		pair<int,int> x;
		x.first = s1;
		x.second = t1;
		return x;
	}
	return extendedeuclidean(r2, r1%r2, s2, s1-r1/r2*s2, t2, t1-r1/r2*t2);
}

// multiplicative inverse returns -1 if it doesnot exist
int multiplicativeInverse(int a,int n){
	int inverse;
	if(gcd(n,a)==1){
		inverse=(extendedeuclidean(a,n,1,0,0,1).first)%n;
		if(inverse<0){
			inverse=inverse+n;
		}
		return inverse;
	}
	else{
		return -1;
	}
}

class randomGenerator{
	int seed;
	public:
		randomGenerator(){
			seed=10005;
		}
		randomGenerator(int x){
			seed=x;
		}
		unsigned int getRandom(){
			seed=seed*1103515245+12345;
			return ((unsigned)(seed/65536) % 32768);
		}
};

// check if prime number
int checkPrime(int x){
	int i;
	for(i=2;i<=sqrt(x);i++){
		if(x%i==0)
			return 0;
	}
	return 1;
}

// fast exponentiation with mod
int power(int p,int q,int mod){
	if(q==0)
		return 1;
	long long int temp=(int)power(p,q/2,mod);
	if(q%2==0)
		temp=(temp*temp)%mod;
	else{
		temp=(temp*temp)%mod;
		temp=(temp*p)%mod;
	}
	return (int)temp;
}

// encrypt the number with rsa
int encryptRsa(int input,int e,int n){
	return power(input,e,n);
}

// decrypt the number with dsa
int decryptRsa(int input,int d,int n){
	return power(input,d,n);
}

int main(){
	int temp,prime,phi,p,q,n,d,e;
	randomGenerator myrand = randomGenerator(time(NULL));
```

```
// get the prime number p
while(1){
        // find the random number
        temp=myrand.getRandom();

        // check if the random number is prime number
        prime=checkPrime(temp);
        if(prime==1)
                break;
}
p=temp;

// get the prime number q
while(1){
        // find the random number
        temp=myrand.getRandom();

        // check if the random number is prime number
        prime=checkPrime(temp);
        if(prime==1)
                break;
}
q=temp;

phi=(p-1)*(q-1);
n=p*q;
printf("p=%d\nq=%d\nn=%d\n",p,q,n);

// get e
while(1){
        // e belongs > 1 and < phi
        e=myrand.getRandom()%(phi-2)+2;
        if(gcd(e,phi)==1)
                break;
}

d=multiplicativeInverse(e, phi);
printf("d=%d\ne=%d\n",d,e);

int input,encrypt,decrypt;

// take input
printf("Enter the number to be encrypted: ");
scanf("%d", &input);

// encrypt using rsa
encrypt=encryptRsa(input, e, n);
printf("Encrypted: %d\n",encrypt);

// decrypt using rsa
decrypt=decryptRsa(encrypt, d, n);
printf("Decrypted: %d\n",decrypt);
```

```
        return 0;
}
```

Output:



```
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ c++ rsa.cpp
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
p=31727
q=21713
n=688888351
d=196066577
e=16017
Enter the number to be encrypted: 57
Encrypted: 45972557
Decrypted: 57
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
p=31607
q=14551
n=459913457
d=366817693
e=4957
Enter the number to be encrypted: 25
Encrypted: 174217901
Decrypted: 25
```

PROGRAM 11

MD5(MESSAGE DIGEST 5)

Description:

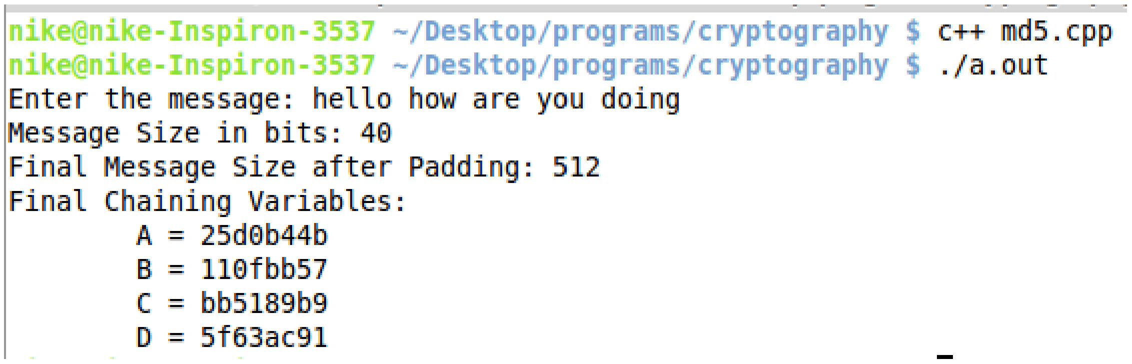
The MD5 message-digest algorithm is a widely used cryptographic hash function producing a 128-bit (16-byte) hash value, typically expressed in text format as a 32-digit hexadecimal number. MD5 has been utilized in a wide variety of cryptographic applications and is also commonly used to verify data integrity.

Program:

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

unsigned int pFunction(unsigned int b,unsigned int c,unsigned int d, int round){
	switch(round){
		case 1:
			return (b&c)|((!b)&d);
		case 2:
			return (b&c)|(c&(!d));
		case 3:
			return b^c^d;
		case 4:
			return c^(b|(!d));
		default:
			printf("Illegal round\n");
			return -1;
	}
}

unsigned int leftCircularShift(unsigned int x, int shift){
	return (x<<shift) + (x>>(32-shift));
}

bool* convertToBitWise(char message[], unsigned int bitMessageSize){
	int count=0,temp;
	bool *bitMessage=(bool *)malloc(bitMessageSize);
	for(int i=0;message[i]!='\0';i++){
		temp = message[i];
		for(int j=7;j>=0;j--){
			bitMessage[count+j]=temp%2;
			temp=temp/2;
		}
		count+=8;
	}
	return bitMessage;
}

unsigned int bitToUnsignedInt(bool *bitMessage){
	unsigned int temp=0;
	for(int i=0;i<32;i++){
		temp=temp*2+bitMessage[i];
	}
	return temp;
}

bool* doPadding(bool bitMessage[], unsigned int *bitMessageSize){
	int padddingSize=512-(*bitMessageSize+64)%512;
	bitMessage=(bool *)realloc(bitMessage, (*bitMessageSize)+padddingSize+64 );
	memset(bitMessage+(*bitMessageSize), 0, padddingSize);
	bitMessage[*bitMessageSize]=1;
	*bitMessageSize=(*bitMessageSize)+padddingSize+64;
	return bitMessage;
}

int main(){
	bool *bitMessage;
	char message[1000];
	unsigned int bitMessageSize, noOfMessageBlock;
	unsigned int A=0x67452301, B=0xefcdab89, C=0x98badcfe, D=0x10325476, a=0, b=0, c=0, d=0,
temp;
	unsigned int shift[4][4]={{7,12,17,22},{5,9,14,20},{4,11,16,23},{6,10,15,21}};
	unsigned int t[64];

	// initialise t array
	long long int maxUnsignedInt=(long long int)pow(2,32);
	for(int i=0;i<64;i++)
		t[i] = floor(maxUnsignedInt * fabs(sin(i + 1)));

	// get the input and convert in bit form
	printf("Enter the message: ");
	scanf("%s",message);
	bitMessageSize=strlen(message)*8;
	printf("Message Size in bits: %u\n", bitMessageSize);
	bitMessage=convertToBitWise(message, bitMessageSize);
	bitMessage=doPadding(bitMessage, &bitMessageSize);
	printf("Final Message Size after Padding: %u\n", bitMessageSize);

	// find the message digest
	noOfMessageBlock=bitMessageSize/512;
	for(int i=0;i<noOfMessageBlock;i++){
		//  initialise variable for this message block
		a=A;
		b=B;
		c=C;
		d=D;

		// 4 rounds
		for(int round=1;round<=4;round++){
			for(int iter=0;iter<16;iter++){
				temp=pFunction(b, c, d, round);
```

```
                                        temp+=a;
                                        temp+=bitToUnsignedInt(bitMessage+i*512+iter*32);
                                        temp+=t[(round-1)*16+iter];
                                        temp=leftCircularShift(temp, shift[round-1][iter%4]);
                                        temp+=b;
                                        a=d;
                                        d=c;
                                        c=b;
                                        b=temp;
                              }
                    }

                    // update chaining variable for next message block
                    A=A+a;
                    B=B+b;
                    C=C+c;
                    D=D+d;
          }

          printf("Final Chaining Variables: \n");
          printf("\tA = %x\n\tB = %x\n\tC = %x\n\tD = %x\n",a,b,c,d);
          return 0;
}
```

Output:



```
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ c++ md5.cpp
nike@nike-Inspiron-3537 ~/Desktop/programs/cryptography $ ./a.out
Enter the message: hello how are you doing
Message Size in bits: 40
Final Message Size after Padding: 512
Final Chaining Variables:
        A = 25d0b44b
        B = 110fbb57
        C = bb5189b9
        D = 5f63ac91
```