##### 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: Submitted By:**

Ms. DEEPIKA SETHI RAJESH GUPTA

Asst. Professor 12103061

Department of CSE 6th Semester

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| S. No | Objective | Page No. | Signature |
| 1. | Write a program to implement Caesar Cipher. | 2 |  |
| 2. | Write a program to implement Playfair Cipher. | 4 |  |
| 3. | Write a program to implement Railfence Cipher. | 8 |  |
| 4. | Write a program to implement Hill Cipher. | 11 |  |
| 5. | Write a program to implement Extended Eucledian Algorithm. | 14 |  |
| 6. | Write a program to implement Random Number Generator. | 16 |  |
| 7. | Write a program to implement Chinese Remainder Theorem. | 18 |  |
| 8. | Write a program to implement Primality Testing. | 21 |  |
| 9. | Write a program to implement RSA Algorithm. | 23 |  |
| 10. | Write a program to implement DES Algorithm. | 27 |  |
| 11. | Write a program to implement MD5 Algorithm. | 33 |  |

**PROGRAM 1**

**CAESAR CIPHER**

**Description:**

In cryptography, a Caesar 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 plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on. The method is named after Julius Caesar, who used it in his private correspondence.

The encryption step performed by a Caesar cipher is often incorporated as part of more complex schemes, such as the Vigenère cipher, and still has modern application in the ROT13 system. As with all single alphabet substitution ciphers, the Caesar cipher is easily broken and in modern practice offers essentially no communication security.

**Program:**

#include<stdio.h>

#include<string.h>

#include<math.h>

void encrypt(FILE \*fp1,FILE \*fp2);

void decrypt(FILE \*fp1,FILE \*fp2);

int main(){

int key;

FILE \*fp1,\*fp2;

encrypt(fp1,fp2);

decrypt(fp1,fp2);

return 0;

}

void encrypt(FILE \*fp1,FILE \*fp2){

int key;char s;

fp1=fopen("input.txt","r");

fp2=fopen("encrypt.txt","w");

key = 3;

while(!feof(fp1)){

fscanf(fp1,"%c",&s);

if(s>=97&&s<=122){

s -= 97;s = (s+ key)%26;s +=97;

}

else if(s>=65&&s<=90){

s -= 65;s = (s+ key)%26;s +=65;

}

fprintf(fp2,"%c",s);

}

fclose(fp1);

fclose(fp2);

}

void decrypt(FILE \*fp1,FILE \*fp2){

int key;char s;

fp1=fopen("encrypt.txt","r");

fp2=fopen("decrypt.txt","w");

key = 3;

while(!feof(fp1)){

fscanf(fp1,"%c",&s);

if(s>=97&&s<=123){

s -= 97;s = (s- key);

if(s<0){

s = 26- (s\*-1)%26;

}

s %= 26;

s +=97;

}

else if(s>=65&&s<=91){

s -= 65;s = (s- key);

if(s<0){

s = 26 - (s\*-1)%26;

}

s %= 26;

s +=65;

}

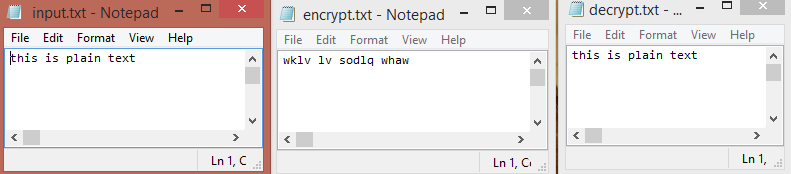
fprintf(fp2,"%c",s);

}

fclose(fp1);

fclose(fp2);

}



Caesar Cipher

**PROGRAM 2**

**PLAYFAIR CIPHER**

**Description**:

The technique encrypts pairs of letters (digraphs), instead of single letters as in the simple [substitution cipher](http://en.wikipedia.org/wiki/Substitution_cipher). The Playfair is thus significantly harder to break since the [frequency analysis](http://en.wikipedia.org/wiki/Frequency_analysis) used for simple substitution ciphers does not work with it. Frequency analysis can still be undertaken, but on the 600[[1]](http://en.wikipedia.org/wiki/Playfair_cipher#cite_note-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**:

#include<stdio.h>

#include<string.h>

int encrypt(char s[200],char k[5][5]);

int decrypt(char s[200],char k[5][5]);

int main(){

char s[200],str[200],k[5][5],c;

memset(k,'0',sizeof(k));

int i,j,l,r=0,m;

int a[26]={0};

printf("Enter the key:-\n");

scanf("%s",s);

l=strlen(s);

printf("%d\n",l);

for(i=0;i<5;i++){

for(j=0;j<5;j++){

if(r<l){

while(a[s[r]-97]==1&&r<l){

r++;

}

if(r<l){

k[i][j]=s[r];

if(s[r]=='i'||s[r]=='j'){

a['i'-97]=a['j'-97]=1;

k[i][j]='i';

}

a[s[r]-97]=1;

r++;

}

}

if(k[i][j]=='0'){

for(m=0;m<26;m++){

if(a[m]==0){

k[i][j]=97+m;

if(m==9||m==8){

a[9]=a[8]=1;

k[i][j]='i';

}

a[m]=1;

break;

}

}

}

}

}

for(i=0;i<5;i++){

for(j=0;j<5;j++){

printf("%c ",k[i][j]);

}

printf("\n");

}

printf("Enter the plain text:-\n");

scanf("%s",str);

l=strlen(str);

for(m=0;m<l;m+=2){

if(str[m]==str[m+1]){

c=str[m+1];

str[m+1]='x';

l++;

for(i=l;i>m+1;i--){

str[i]=str[i-1];

}

str[m+2]=c;

}

}

if(l%2!=0){

str[l]='x';

str[l+1]='\0';

l++;

}

for(m=0;m<l;m++){

if(str[m]=='j'){

str[m]='i';

}

}

//printf("%s\n",str);

encrypt(str,k);

printf("Encrypted text =\t%s\n",str);

decrypt(str,k);

printf("Decrypted text =\t%s\n",str);

return 0;

}

int encrypt(char s[200],char K[5][5]){

int k,i,j,a1,b1,a2,b2,l,t;

l= strlen(s);

for(k=0;k<l;k += 2){

for(i=0;i<5;i++){

for(j=0;j<5;j++){

if(K[i][j]==s[k]){

a1=i;b1=j;

}

if(K[i][j]==s[k+1]){

a2=i;b2=j;

}

}

}

if(b1==b2){

a1++;a2++;

a1 %= 5;a2 %=5;

}

else if(a1==a2){

b1++;b2++;

b1 %= 5;b2 %=5;

}

else{

t=b1;

b1=b2;

b2=t;

}

s[k] = K[a1][b1];

s[k+1]= K[a2][b2];

}

return 0;

}

int decrypt(char s[200],char K[5][5]){

int k,i,j,a1,b1,a2,b2,l,t;

l= strlen(s);

for(k=0;k<l;k += 2){

for(i=0;i<5;i++){

for(j=0;j<5;j++){

if(K[i][j]==s[k]){

a1=i;b1=j;

}

if(K[i][j]==s[k+1]){

a2=i;b2=j;

}

}

}

if(b1==b2){

a1--;a2--;a1+=5;a2+=5;

a1 %= 5;a2 %=5;

}

else if(a1==a2){

b1--;b2--;b1+=5;b2+=5;

b1 %= 5;b2 %=5;

}

else{

t=b1;

b1=b2;

b2=t;

}

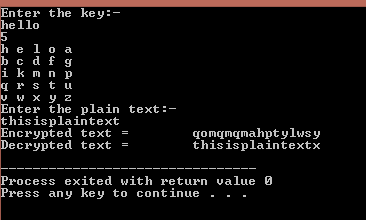
s[k] = K[a1][b1];

s[k+1]= K[a2][b2];

}

return 0;

}



Play Fair Cipher

**PROGRAM 3**

**RAIL FENCE CIPHER**

**Description:**

Rail Fence Cipher (also called a zigzag cipher) generally refers to a form of [transposition cipher](http://en.wikipedia.org/wiki/Transposition_cipher). It derives its name from the way in which it is encoded.In the rail fence cipher, the [plaintext](http://en.wikipedia.org/wiki/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<stdio.h>

#include<math.h>

#include<string.h>

int encrypt(char r[],char s[],int n){

int i,j,x,u,o,y,l=0,k,p;

for(i=0;i<n;i++){

x = n-i-1;

y = n-x-1;

if(x>1)

o = (x-1)\*2+1;

else if(x==1)

o = 1;

else

o = 0;

if(y>1)

u = (y-1)\*2+1;

else if(y==1)

u = 1;

else

u = 0;

k=i;j=0;

for(p=0;p<i;p++)

printf(" ");

while(k<strlen(s)){

r[l++] = s[k];

printf("%c",s[k]);

if(u>0&&o>0){

if(j%2==0){

k += o+1;

for(p=0;p<o+1;p++)

printf(" ");

}

else{

k += u+1;

for(p=0;p<u+1;p++)

printf(" ");

}

}

else{

k += o+u+1;

for(p=0;p<o+u+2;p++)

printf(" ");

}

j++;

//k = strlen(s);

}

printf("\n");

}

r[strlen(s)]='\0';

return 0;

}

int decrypt(char r[],char s[],int n){

int i,j,x,u,o,y,l=0,k,p;

for(i=0;i<n;i++){

x = n-i-1;

y = n-x-1;

if(x>1)

o = (x-1)\*2+1;

else if(x==1)

o = 1;

else

o = 0;

if(y>1)

u = (y-1)\*2+1;

else if(y==1)

u = 1;

else

u = 0;

k=i;j=0;

while(k<strlen(s)){

r[k] = s[l++];

if(u>0&&o>0){

if(j%2==0){

k += o+1;

}

else{

k += u+1;

}

}

else{

k += o+u+1;

}

j++;

}

}

r[strlen(s)]='\0';

return 0;

}

int main(){

int n;

printf("Enter the no. of rails:-\n");

scanf("%d",&n);

char s[10000],r[10000];

printf("Enter the Plain Text:-\n");

scanf("%s",s);

encrypt(r,s,n);

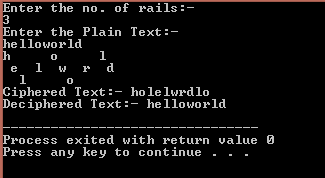
printf("Ciphered Text:- %s\n",r);

decrypt(s,r,n);

printf("Deciphered Text:- %s\n",s);

return 0;

}



RailFence Cipher

**PROGRAM 4**

**HILL CIPHER**

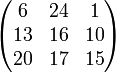
**Description**:

In [classical cryptography](http://en.wikipedia.org/wiki/Classical_cryptography), the Hill cipher is a [polygraphic substitution cipher](http://en.wikipedia.org/wiki/Substitution_cipher" \l "Polygraphic" \o "Substitution cipher) based on [linear algebra](http://en.wikipedia.org/wiki/Linear_algebra).

Each letter is represented by a number [modulo](http://en.wikipedia.org/wiki/Modular_arithmetic) 26. (Often the simple scheme A = 0, B = 1, ..., Z = 25 is used, but this is not an essential feature of the cipher.) To encrypt a message, each block of n letters (considered as an n-component [vector](http://en.wikipedia.org/wiki/Vector_space)) is multiplied by an invertible n × n [matrix](http://en.wikipedia.org/wiki/Matrix_(mathematics)), again [modulus](http://en.wikipedia.org/wiki/Modular_arithmetic) 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.

The matrix used for encryption is the cipher [key](http://en.wikipedia.org/wiki/Key_(cryptography)), and it should be chosen randomly from the set of invertible n × n matrices ([modulo](http://en.wikipedia.org/wiki/Modular_arithmetic) 26). The cipher can, of course, be adapted to an alphabet with any number of letters; all arithmetic just needs to be done [modulo](http://en.wikipedia.org/wiki/Modular_arithmetic) the number of letters instead of [modulo](http://en.wikipedia.org/wiki/Modular_arithmetic) 26.

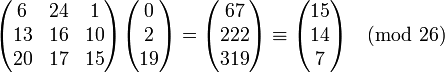
Consider the message 'ACT', and the key below (or GYBNQKURP in letters):



Since 'A' is 0, 'C' is 2 and 'T' is 19, the message is the vector:

\begin{pmatrix} 0 \\ 2 \\ 19 \end{pmatrix}

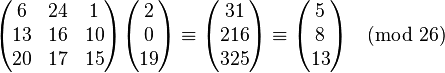
Thus the enciphered vector is given by:



which corresponds to a [ciphertext](http://en.wikipedia.org/wiki/Ciphertext" \o "Ciphertext) of 'POH'. Now, suppose that our message is instead 'CAT', or:

\begin{pmatrix} 2 \\ 0 \\ 19 \end{pmatrix}

This time, the enciphered vector is given by:



which corresponds to a ciphertext of 'FIN'. Every letter has changed.

**Program:**

#include<stdio.h>

#include<string.h>

int key[4][4] = {{3,5,7,2},{1,4,7,2},{6,3,9,17},{13,5,4,16}};

int keyi[4][4] = {{15,21,0,15},{23,9,0,22},{15,16,18,3},{24,7,15,3}};

int main(){

int t[4],i,j,k,l,n,r=0;

char s[100],str[100],str2[100];

printf("Enter string:-\n");

scanf("%s",s);

k = strlen(s);

for(i=0;i<k;i += 4){

t[0]=s[i]-'a';

t[1]=t[2]=t[3]='x'-'a';

if(i+1<k){

t[1]=s[i+1]-'a';

}

if(i+2<k){

t[2]=s[i+2]-'a';

}

if(i+3<k){

t[3]=s[i+3]-'a';

}

for(j=0;j<4;j++){

n=0;

for(l=0;l<4;l++){

n += (t[l]\*key[l][j]);

}

str[r] = (n%26)+'a';r++;

}

}

str[r] = '\0';r=0;

k = strlen(str);

printf("Encrypted :-\t%s\n",str);

for(i=0;i<k;i += 4){

t[0]=str[i]-'a';

t[1]=t[2]=t[3]='x'-'a';

if(i+1<k){

t[1]=str[i+1]-'a';

}

if(i+2<k){

t[2]=str[i+2]-'a';

}

if(i+3<k){

t[3]=str[i+3]-'a';

}

for(j=0;j<4;j++){

n=0;

for(l=0;l<4;l++){

n += t[l]\*keyi[l][j];

}

str2[r++] = (n%26)+'a';

}

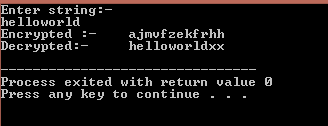
}

str2[r] = '\0';

printf("Decrypted:-\t%s\n",str2);

return 0;

}



**Hill Cipher**

**PROGRAM 5**

**EXTENDED EUCLEDIAN 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 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 <stdio.h>

int main (){

int a,b,q,x,lastx,y,lasty,temp,temp1,temp2,temp3;

printf("Please input a\n");

scanf("%d",&a);

printf("Please input b\n");

scanf("%d",&b);

if (b>a) {

temp=a; a=b; b=temp;

}

x=0;

y=1;

lastx=1;

lasty=0;

while (b!=0) {

q= a/b;

temp1= a%b;

a=b;

b=temp1;

temp2 = x;

x=lastx-q\*x;

lastx = temp2;

temp3 = y;

y = lasty-q\*y;

lasty=temp3; }

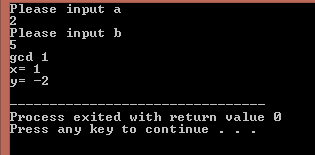
printf("gcd %d\n",a);

printf("x= %d\n",lastx);

printf("y= %d\n",lasty);

return 0;

}



Extended Eucledian

**PROGRAM 6**

**RANDOM NUMBER GENERATOR**

**Description:**

A linear congruential generator (LCG) is an [algorithm](http://en.wikipedia.org/wiki/Algorithm) that yields a sequence of randomized numbers calculated with a [linear equation](http://en.wikipedia.org/wiki/Linear_equation). The method represents one of the oldest and best-known [pseudorandom number generator](http://en.wikipedia.org/wiki/Pseudorandom_number_generator) algorithms.[[1]](http://en.wikipedia.org/wiki/Linear_congruential_generator#cite_note-1) The theory behind them is easy to understand, and they are easily implemented and fast, especially on computer hardware which can provide [modulo arithmetic](http://en.wikipedia.org/wiki/Modulo_arithmetic) by storage-bit truncation.

The generator is defined by the [recurrence relation](http://en.wikipedia.org/wiki/Recurrence_relation):

X_{n+1} = \left( a X_n + c \right)~~\bmod~~m

where X is the [sequence](http://en.wikipedia.org/wiki/Sequence) of pseudorandom values, and

 m,\, 0<m  – the "[modulus](http://en.wikipedia.org/wiki/Modulo_operation)"

 a,\,0 < a < m – the "multiplier"

 c,\,0 \le c < m – the "increment"

 X_0,\,0 \le X_0 < m – the "seed" or "start value"

are [integer](http://en.wikipedia.org/wiki/Integer) constants that specify the generator. If c = 0, the generator is often called a multiplicative congruential generator (MCG), or [Lehmer RNG](http://en.wikipedia.org/wiki/Lehmer_RNG" \o "Lehmer RNG). If c ≠ 0, the method is called a mixed congruential generator.

**Program:**

#include<stdio.h>

#include<string.h>

#include<math.h>

long long a=pow(7,5),b=0,n=pow(2,31)-1,x0=9;

long long random(){

return x0= (a\*x0+b)%n;

}

int main(){

long long i;

char c[2];

while(1){

printf("%lld\nWant another number : y or n ?\n",random());

scanf("%s",c);

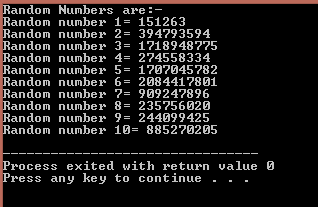
if(c[0]=='n')

break;

}

return 0;

}



Random Number Generator

**PROGRAM 7**

**CHINESE REMAINDER THEOREM**

**Description:**

The Chinese remainder theorem is a result about [congruences](http://en.wikipedia.org/wiki/Modular_arithmetic" \o "Modular arithmetic) in [number theory](http://en.wikipedia.org/wiki/Number_theory) and its generalizations in [abstract algebra](http://en.wikipedia.org/wiki/Abstract_algebra). It was first published in the 3rd to 5th centuries by Chinese mathematician [Sun Tzu](http://en.wikipedia.org/wiki/Sun_Tzu_(mathematician)).

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?

The original form of the theorem, contained in the 5th-century book [Sunzi's Mathematical Classic](http://en.wikipedia.org/wiki/Sunzi%27s_Mathematical_Classic" \o "Sunzi's Mathematical Classic) (孫子算經) by the Chinese mathematician [Sun Tzu](http://en.wikipedia.org/wiki/Sun_Tzu_(mathematician)) and later generalized with a complete solution called Dayanshu (大衍術) in [Qin Jiushao](http://en.wikipedia.org/wiki/Qin_Jiushao)'s 1247 [Mathematical Treatise in Nine Sections](http://en.wikipedia.org/wiki/Mathematical_Treatise_in_Nine_Sections) (數書九章, Shushu Jiuzhang), is a statement about simultaneous congruences.

Suppose n1, n2, …, nk are positive [integers](http://en.wikipedia.org/wiki/Integer) that are [pairwise coprime](http://en.wikipedia.org/wiki/Pairwise_coprime" \o "Pairwise coprime). Then, for any given sequence of integers a1,a2, …, ak, there exists an integer x solving the following system of simultaneous congruences.

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

Furthermore, all solutions x of this system are congruent modulo the product, N = n1n2…nk.

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

Sometimes, the simultaneous congruences can be solved even if the ni'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](http://en.wikipedia.org/wiki/Least_common_multiple) of the ni.

**Program:**

#include<stdio.h>

#include<string.h>

long long gcd(long long a,long long b){

if(a%b==0)

return b;

else

return gcd(b,a%b);

}

long long m\_inv(long long r1,long long r2){

long long r=1,t1=0,t2=1,t,q,x=r1;

while(r!=0){

r = r1%r2;

q = r1/r2;

r1 = r2;

r2 = r;

t = t1 - q\*t2;

t1 = t2;

t2 = t;

}

while(t1<0)

t1 += x;

return t1%x;

}

int main(){

long long n,i;

printf("Enter number of equations:-\n");

scanf("%lld",&n);

long long a[n],s[n],minv[n],m[n],M=1,ans=0;

printf("Enter a and s:-\n");

for(i=0;i<n;i++){

scanf("%lld %lld",&a[i],&s[i]);

}

M = s[0];

for(i=1;i<n;i++){

if(gcd(M,s[i])!=1)

break;

M \*= s[i];

}

if(i<n){

printf("Not co prime\n");

return 0;

}

for(i=0;i<n;i++){

m[i] = M/s[i];

minv[i]=m\_inv(s[i],m[i]%s[i]);

}

for(i=0;i<n;i++){

ans += a[i]\*m[i]\*minv[i];

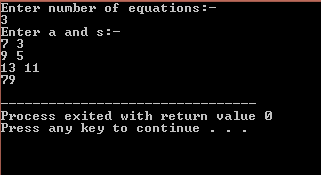
ans %= M;

}

printf("%lld\n",ans);

return 0;

}



Chinese Remainder

**PROGRAM 8**

**PRIMARITY TESTING THEOREM**

**Description:**

The simplest primality test is [trial division](http://en.wikipedia.org/wiki/Trial_division): Given an input number n, check whether any integer m from 2 to n − 1 evenly [divides](http://en.wikipedia.org/wiki/Divisibility) n (the division leaves no [remainder](http://en.wikipedia.org/wiki/Remainder)). If n is divisible by any m then nis [composite](http://en.wikipedia.org/wiki/Composite_number), otherwise it is [prime](http://en.wikipedia.org/wiki/Prime_number).

For example, to test whether 17 is prime, test whether 17 is divisible by 2, or 3, or 4, 5, 6, ..., 16. Since a prime is only divisible by 1 and itself, if we reach 16 without finding a [divisor](http://en.wikipedia.org/wiki/Divisor), then we have proven that 17 is prime. However, we don't actually have to check all numbers up to n. Let's look at another example: all the divisors of 100:

2, 4, 5, 10, 20, 25, 50

here we see that the largest factor is 100/2 = 50. This is true for all n: all divisors are less than or equal to n/2. We can do better though. If we take a closer look at the divisors, we will see that some of them are redundant. If we write the list differently:

100 = 2 × 50 = 4 × 25 = 5 × 20 = 10 × 10 = 20 × 5 = 25 × 4 = 50 × 2

it becomes obvious. Once we reach 10, which is \scriptstyle\sqrt {100}, the divisors just flip around and repeat. Therefore we can further eliminate testing divisors greater than \scriptstyle\sqrt n. We can also eliminate all the even numbers greater than 2, since if an even number can divide n, so can 2.

**Program:**

#include<stdio.h>

#include<string.h>

#include<math.h>

long long a=pow(7,5),b=0,n=pow(2,31)-1,x0=9;

int random(){

return x0= (a\*x0+b)%n;

}

int main(){

int i,x,p;

char c[2];

while(1){

x = random();

printf("%d\n",x);

p=0;

for(i=2;pow(i,2)<=x;i++){

if(x%i==0)

p=1;

}

if(p==1)

printf("Not Prime\n");

else

printf("Prime\n");

printf("Want another number : y or n ?\n");

scanf("%s",c);

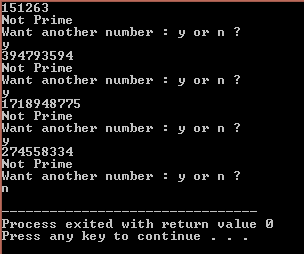
if(c[0]=='n')

break;

}

return 0;

}



Primality Test

**PROGRAM 9**

**RSA ALGORITHM**

**Description:**

RSA is one of the first practicable [public-key cryptosystems](http://en.wikipedia.org/wiki/Public-key_cryptography) and is widely used for secure data transmission. In such a [cryptosystem](http://en.wikipedia.org/wiki/Cryptosystem), the [encryption key](http://en.wikipedia.org/wiki/Encryption_key) is public and differs from the [decryption key](http://en.wikipedia.org/wiki/Decryption_key) which is kept secret. In RSA, this asymmetry is based on the practical difficulty of [factoring](http://en.wikipedia.org/wiki/Factorization) the product of two large [prime numbers](http://en.wikipedia.org/wiki/Prime_number), the [factoring problem](http://en.wikipedia.org/wiki/Factoring_problem). RSA stands for [Ron Rivest](http://en.wikipedia.org/wiki/Ron_Rivest), [Adi Shamir](http://en.wikipedia.org/wiki/Adi_Shamir" \o "Adi Shamir) and[Leonard Adleman](http://en.wikipedia.org/wiki/Leonard_Adleman), who first publicly described the algorithm in 1977. [Clifford Cocks](http://en.wikipedia.org/wiki/Clifford_Cocks), an English mathematician, had developed an equivalent system in 1973, but it wasn't [declassified](http://en.wikipedia.org/wiki/Classified_information) until 1997.[[1]](http://en.wikipedia.org/wiki/RSA_(cryptosystem)#cite_note-1)

A user of RSA creates and then publishes the [product](http://en.wikipedia.org/wiki/Product_(mathematics)) of two large [prime numbers](http://en.wikipedia.org/wiki/Prime_number), along with an auxiliary value, as their public key. The prime factors must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of the prime factors can feasibly decode the message.[[2]](http://en.wikipedia.org/wiki/RSA_(cryptosystem)#cite_note-rsa-2) Breaking RSA[encryption](http://en.wikipedia.org/wiki/Encryption) is known as the [RSA problem](http://en.wikipedia.org/wiki/RSA_problem). It is an open question whether it is as hard as the factoring problem.

Key generation

RSA involves a public key and a [private key](http://en.wikipedia.org/wiki/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](http://en.wikipedia.org/wiki/Prime_number) 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](http://en.wikipedia.org/wiki/Primality_test" \o "Primality test).
2. Compute n = pq.
   * n is used as the [modulus](http://en.wikipedia.org/wiki/Modular_arithmetic) for both the public and private keys. Its length, usually expressed in bits, is the [key length](http://en.wikipedia.org/wiki/Key_length).
3. Compute φ(n) = φ(p)φ(q) = (p − 1)(q − 1) = n - (p + q -1), where φ is [Euler's totient function](http://en.wikipedia.org/wiki/Euler%27s_totient_function).
4. Choose an integer e such that 1 < e < φ(n) and [gcd](http://en.wikipedia.org/wiki/Greatest_common_divisor" \o "Greatest common divisor)(e, φ(n)) = 1; i.e., e and φ(n) are [coprime](http://en.wikipedia.org/wiki/Coprime" \o "Coprime).
   * e is released as the public key exponent.
   * e having a short [bit-length](http://en.wikipedia.org/wiki/Bit-length) and small [Hamming weight](http://en.wikipedia.org/wiki/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]](http://en.wikipedia.org/wiki/RSA_(cryptosystem)#cite_note-Boneh-5)
5. Determine d as d ≡ e−1 (mod φ(n)); i.e., d is the [multiplicative inverse](http://en.wikipedia.org/wiki/Modular_multiplicative_inverse) of e (modulo φ(n)).

* This is more clearly stated as: solve for d given d⋅e ≡ 1 (mod φ(n))
* This is often computed using the [extended Euclidean algorithm](http://en.wikipedia.org/wiki/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.

The public key consists of the modulus n and the public (or encryption) exponent e. The private key consists of the modulus n and the private (or decryption) exponent d, which must be kept secret. p, q, and φ(n) must also be kept secret because they can be used to calculate d.

* An alternative, used by [PKCS#1](http://en.wikipedia.org/wiki/PKCS1), is to choose d matching de ≡ 1 (mod λ) with λ = lcm(p − 1, q − 1), where lcm is the [least common multiple](http://en.wikipedia.org/wiki/Least_common_multiple). Using λ instead of φ(n) allows more choices for d. λ can also be defined using the [Carmichael function](http://en.wikipedia.org/wiki/Carmichael_function), λ(n).
* The [ANSI X9.31](http://en.wikipedia.org/w/index.php?title=ANSI_X9.31&action=edit&redlink=1) standard prescribes, [IEEE 1363](http://en.wikipedia.org/wiki/P1363) describes, and [PKCS#1](http://en.wikipedia.org/wiki/PKCS1) allows, that p and q match additional requirements: being [strong primes](http://en.wikipedia.org/wiki/Strong_prime), and being different enough that [Fermat factorization](http://en.wikipedia.org/wiki/Fermat_factorization) fails.

Encryption:

[Alice](http://en.wikipedia.org/wiki/Alice_and_Bob) transmits her public key (n, e) to [Bob](http://en.wikipedia.org/wiki/Alice_and_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 ≤ m < n by using an agreed-upon reversible protocol known as a [padding scheme](http://en.wikipedia.org/wiki/RSA_(cryptosystem)#Padding_schemes). 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](http://en.wikipedia.org/wiki/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]](http://en.wikipedia.org/wiki/RSA_(cryptosystem)" \l "cite_note-6) 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} 

Given m, she can recover the original message M by reversing the padding scheme.

**Program:**

#include<iostream>

#include <math.h>

#include<string.h>

using namespace std;

long long gcd(long a,long b)

{

if(a%b==0)

return b;

else

return gcd(b,a%b);

}

int main()

{

long long p=19;

long long q=13;

long long n=p\*q;

long long phi=(p-1)\*(q-1);

long long e;

for(long long i=2;i<phi;i++)

{

if(gcd(i,phi)==1)

{

e=i;

break;

}

}

long long d;

for(long long i=2;;i++)

{

long long h=((i\*phi)+1)/e;

//cout<<h;

if((h\*e)%phi==1)

{

d=h;

break;

}

}

cout<<"Want to cipher String(S) or Numbers(N)\n";

char c;

cin>>c;

if(c=='S'||c=='s')

{

char t[1000];

char l[1000];

cout<<"Enter the Char to cipher\n";

cin>>t;

for(int i=0;i<strlen(t);i++){

if(t[i]>=65&&t[i]<=90)

{

long long k=t[i]-64;

unsigned long long h=1;

for(long long i=1;i<=e;i++)

h=(h\*k)%n;

cout<<(char)(h%27+64);

k=h;

h=1;

for(long long i=1;i<=d;i++)

h=(h\*k)%n;

l[i]=(char)(h%27+64);

}

else

{

long long k=t[i]-96;

unsigned long long h=1;

for(long long i=1;i<=e;i++)

h=(h\*k)%n;

cout<<(char)(h%27+96);

k=h;

h=1;

for(long long i=1;i<=d;i++)

h=(h\*k)%n;

l[i]=(char)(h%27+96);

}}

cout<<"\n";

l[strlen(t)]='\0';

cout<<l;

cout<<"\n";

}

if(c=='N'||c=='n');

{

long long k;

cout<<"Enter the Number to cipher\n";

cin>>k;

unsigned long long h=1;

for(long long i=1;i<=e;i++)

h=(h\*k)%n;

cout<<"Cipher Text :"<<h<<"\n";

k=h;

h=1;

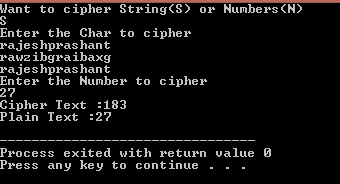
for(long long i=1;i<=d;i++)

h=(h\*k)%n;

cout<<"Plain Text :"<<h<<"\n";

}

}



RSA Algorithm

**PROGRAM 10**

**DES ALGORITHM**

**Description:**

The Data Encryption Standard (DES, [/](http://en.wikipedia.org/wiki/Help:IPA_for_English)[ˌdiːˌiːˈɛs](http://en.wikipedia.org/wiki/Help:IPA_for_English#Key)[/](http://en.wikipedia.org/wiki/Help:IPA_for_English) or [/](http://en.wikipedia.org/wiki/Help:IPA_for_English)[ˈdɛz](http://en.wikipedia.org/wiki/Help:IPA_for_English#Key)[/](http://en.wikipedia.org/wiki/Help:IPA_for_English)) is a previously predominant [symmetric-key algorithm](http://en.wikipedia.org/wiki/Symmetric-key_algorithm) for the [encryption](http://en.wikipedia.org/wiki/Encryption) of electronic data. It was highly influential in the advancement of modern [cryptography](http://en.wikipedia.org/wiki/Cryptography) in the academic world. Developed in the early 1970s at[IBM](http://en.wikipedia.org/wiki/IBM) and based on an earlier design by [Horst Feistel](http://en.wikipedia.org/wiki/Horst_Feistel), the algorithm was submitted to the [National Bureau of Standards](http://en.wikipedia.org/wiki/National_Bureau_of_Standards) (NBS) following the agency's invitation to propose a candidate for the protection of sensitive, unclassified electronic government data. In 1976, after consultation with the [National Security Agency](http://en.wikipedia.org/wiki/National_Security_Agency) (NSA), the NBS eventually selected a slightly modified version, which was published as an official [Federal Information Processing Standard](http://en.wikipedia.org/wiki/Federal_Information_Processing_Standard) (FIPS) for the [United States](http://en.wikipedia.org/wiki/United_States) in 1977. The publication of an NSA-approved encryption standard simultaneously resulted in its quick international adoption and widespread academic scrutiny. Controversies arose out of[classified](http://en.wikipedia.org/wiki/Classified_information) design elements, a relatively short [key length](http://en.wikipedia.org/wiki/Key_length) of the [symmetric-key](http://en.wikipedia.org/wiki/Symmetric-key_algorithm) [block cipher](http://en.wikipedia.org/wiki/Block_cipher) design, and the involvement of the NSA, nourishing suspicions about a [backdoor](http://en.wikipedia.org/wiki/Backdoor_(computing)). The intense academic scrutiny the algorithm received over time led to the modern understanding of block ciphers and their [cryptanalysis](http://en.wikipedia.org/wiki/Cryptanalysis).

DES is now considered to be insecure for many applications. This is chiefly due to the 56-bit key size being too small; in January, 1999,[distributed.net](http://en.wikipedia.org/wiki/Distributed.net) and the [Electronic Frontier Foundation](http://en.wikipedia.org/wiki/Electronic_Frontier_Foundation) collaborated to publicly break a DES key in 22 hours and 15 minutes (see[chronology](http://en.wikipedia.org/wiki/Data_Encryption_Standard#Chronology)). There are also some analytical results which demonstrate theoretical weaknesses in the cipher, although they are infeasible to mount in practice. The algorithm is believed to be practically secure in the form of [Triple DES](http://en.wikipedia.org/wiki/Triple_DES), although there are theoretical attacks. In recent years, the cipher has been superseded by the [Advanced Encryption Standard](http://en.wikipedia.org/wiki/Advanced_Encryption_Standard) (AES). Furthermore, DES has been withdrawn as a standard by the [National Institute of Standards and Technology](http://en.wikipedia.org/wiki/National_Institute_of_Standards_and_Technology) (formerly the National Bureau of Standards).

Some documentation makes a distinction between DES as a standard and DES as an algorithm, referring to the algorithm as the DEA

**Program:**

#include<stdio.h>

#include<string.h>

char key[9],plaintext[9];

int keyu[64]={0},keys[56],keyp[48],rpt48[48],pt[64],ipt[64],sb[32],pt32[32];

int PC1[8][7]={

57,49,41,33,25,17,9,

1,58,50,42,34,26,18,

10,2,59,51,43,35,27,

19,11,3,60,52,44,36,

63,55,47,39,31,23,15,

7,62,54,46,38,30,22,

14,6,61,53,45,37,29,

21,13,5,28,20,12,4

};

int PC2[8][6]={

14,17,11,24,1,5,

3,28,15,6,21,10,

23,19,12,4,26,8,

16,7,27,20,13,2,

41,52,31,37,47,55,

30,40,51,45,33,48,

44,49,39,56,34,53,

46,42,50,36,29,32

};

int IP[8][8]={

58,50,42,34,26,18,10,2,

60,52,44,36,28,20,12,4,

62,54,46,38,30,22,14,6,

64,56,48,40,32,24,16,8,

57,49,41,33,25,17,9,1,

59,51,43,35,27,19,11,3,

61,53,45,37,29,21,13,5,

63,55,47,39,31,23,15,7};

int E\_bit[8][6]={

32,1,2,3,4,5,

4,5,6,7,8,9,

8,9,10,11,12,13,

12,13,14,15,16,17,

16,17,18,19,20,21,

20,21,22,23,24,25,

24,25,26,27,28,29,

28,29,30,31,32,1};

int s[8][4][16]={{14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,

0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,

4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,

15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13},

{15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10,

3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5,

0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15,

13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9},

{10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8,

13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1,

13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7,

1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12},

{7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15,

13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9,

10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4,

3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14},

{2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9,

14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6,

4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14,

11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3},

{12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11,

10,15,4,2,7,12,9,5,6,1,12,14,0,11,3,8,

9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6,

4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13},

{4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1,

13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6,

1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2,

6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12},

{13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7,

1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2,

7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8,

2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11}};

int sb\_permutation[8][4]={

16,7,20,21,

29,12,28,17,

1,15,23,26,

5,18,31,10,

2,8,24,14, 32,27,3,9,

19,13,30,6,

22,11,4,25};

int ip\_inverse[8][8]={

40,8,48,16,56,24,64,32,

39,7,47,15,55,23,63,31,

38,6,46,14,54,22,62,30,

37,5,45,13,53,21,61,29,

36,4,44,12,52,20,60,28,

35,3,43,11,51,19,59,27,

34,2,42,10,50,18,58,26,

33,1,41,9,49,17,57,25

};

int lshift(int a[],int l,int s,int e){

int i,b[l],c[e-s+1-l],j=0,k=0;

for(i=s;i<s+l;i++){

b[j++]=a[i];

}

for(i=s+l;i<=e;i++){

c[k++]=a[i];

}

for(i=0;i<k;i++){

a[i+s]=c[i];

}

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

a[i+s+k]=b[i];

}

return 0;

}

int getkey(){

int i,j,l;

printf("Enter Key:-\n");

scanf("%s",key);

for(i=0;i<8;i++){

l=key[i];

for(j=7;j>=0;j--){

keyu[8\*i+j]=l%2;

l /=2;

}

}

l=0;

for(i=0;i<8;i++){

for(j=0;j<7;j++){

keys[l++]=keyu[PC1[i][j]-1];

}

}

printf("56 key -");

for(i=0;i<56;i++){

printf("%d",keys[i]);

if(i==27)

printf(" " );

}

printf("\n");

}

unsigned long long keygen(int round){

if(round==1||round==2||round==9||round==16){

lshift(keys,1,0,27);

lshift(keys,1,28,55);

}

else{

lshift(keys,2,0,27);

lshift(keys,2,28,55);

}

int i,j,l=0;

for(i=0;i<8;i++){

for(j=0;j<6;j++){

keyp[l++]=keys[PC2[i][j]-1];

}

}

printf("Key shift- ");

for(i=0;i<28;i++){

printf("%d",keys[i]);

}

printf(" ");

for(;i<56;i++){

printf("%d",keys[i]);

}

printf("\n");

return 0;

}

int main(){

getkey();

int i,j,l,k,r,c,t;

printf("Enter Plain Text:-\n");

scanf("%s",plaintext);

for(i=0;i<8;i++){

l=plaintext[i];

for(j=7;j>=0;j--){

ipt[8\*i+j]=l%2;

l /=2;

}

}

printf("Plain text 1- ");

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

printf("%d",ipt[i]);

}

printf("\n");

l=0;

for(i=0;i<8;i++){

for(j=0;j<8;j++){

pt[l++]=ipt[IP[i][j]-1];

}

}

printf("Plain text 2- ");

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

printf("%d",pt[i]);

}

printf("\n");

for(i=1;i<=16;i++){

printf("Round %d-\n ",i);

l=0;

for(r=0;r<8;r++){

for(j=0;j<6;j++){

rpt48[l++]=pt[E\_bit[r][j]+31];

}

}

printf("Plain text48r- ");

for(j=0;j<48;j++){

printf("%d",rpt48[j]);

}

printf("\n");

keygen(i);

printf("Key - ");

for(j=0;j<48;j++){

printf("%d",keyp[j]);

}

printf("\n");

for(j=0;j<48;j++){

rpt48[j] ^= keyp[j];

}

printf("XOR text - ");

for(j=0;j<48;j++){

printf("%d",rpt48[j]);

}

printf("\n");

for(j=0;j<8;j++){

t = 6\*j;

r = 2\*rpt48[t]+rpt48[t+5];

c = 8\*rpt48[t+1]+4\*rpt48[t+2]+2\*rpt48[t+3]+rpt48[t+4];

l = s[j][r][c];

t = 4\*j;

printf("l=%d r=%d c=%d ",l,r,c);

sb[t+3] = l%2;l /= 2;

sb[t+2] = l%2;l /= 2;

sb[t+1] = l%2;l /= 2;

sb[t] = l%2;l /= 2;

}

printf("\n");

printf("Sbox text - ");

for(j=0;j<32;j++){

printf("%d",sb[j]);

}

printf("\n");

r=0;

for(l=0;l<8;l++){

for(j=0;j<4;j++,r++){

pt32[r]=sb[sb\_permutation[l][j]-1];

}

}

printf("Sbox textp - ");

for(j=0;j<32;j++){

printf("%d",pt32[j]);

}

printf("\n");

for(j=0;j<32;j++){

pt[j]^=pt32[j];

}

if(i!=16)

lshift(pt,32,0,63);

printf("Plain end text- ");

for(j=0;j<64;j++){

printf("%d",pt[j]);

if(j==31)

printf(" " );

}

printf("\n");

}

l=0;

for(i=0;i<8;i++){

for(j=0;j<8;j++,l++){

ipt[l]=pt[ip\_inverse[i][j]-1];

}

}

printf("After Final Permutation - \n");

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

printf("%d",ipt[i]);

}

printf("\n");

return 0;

}



DES Algorithm

**PROGRAM 11**

**MD5 ALGORITHM**

**Description:**

MD5 is an algorithm that is used to verify data integrity through the creation of a 128-bit message digest from data input (which may be a message of any length) that is claimed to be as unique to that specific data as a fingerprint is to the specific individual. MD5, which was developed by Professor Ronald L. Rivest of MIT, is intended for use with digital signature applications, which require that large files must be compressed by a secure method before being encrypted with a secret key, under a public key cryptosystem. MD5 is currently a standard, Internet Engineering Task Force (IETF) Request for Comments (RFC) 1321. According to the standard, it is "computationally infeasible" that any two messages that have been input to the MD5 algorithm could have as the output the same message digest, or that a false message could be created through apprehension of the message digest. MD5 is the third message digest algorithm created by Rivest. All three (the others are MD2 and MD4) have similar structures, but MD2 was optimized for 8-bit machines, in comparison with the two later formulas, which are optimized for 32-bit machines. The MD5 algorithm is an extension of MD4, which the critical review found to be fast, but possibly not absolutely secure. In comparison, MD5 is not quite as fast as the MD4 algorithm, but offers much more assurance of data security.

**Program:**

#include<stdio.h>

#include<string.h>

#include<math.h>

unsigned int binary(unsigned int msg\_512[],unsigned int num,unsigned int pos){

if(num==0){

return 0;

}

msg\_512[pos]=num%2;

binary(msg\_512,num/2,pos-1);

return 0;

}

unsigned int lshift(unsigned int a[],unsigned int l,unsigned int s,unsigned int e){

int i,b[l],c[e-s+1-l],j=0,k=0;

for(i=s;i<s+l;i++){

b[j++]=a[i];

}

for(i=s+l;i<=e;i++){

c[k++]=a[i];

}

for(i=0;i<k;i++){

a[i+s]=c[i];

}

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

a[i+s+k]=b[i];

}

return 0;

}

int main(){

unsigned int A=0x01234567,B=0x89ABCDEF,C=0xFEDCBA98,D=0x76543210;

unsigned int shift[4][4]={{7,12,17,22},{5,9,14,20},{4,11,16,23},{6,10,15,21}};

unsigned int P[4],temp;

unsigned int t[65];

t[1]=0xd76aa478;

t[2]=0xe8c7b756;

t[3]=0x242070db;

t[4]=0xc1bdceee;

t[5]=0xf57c0faf;

t[6]=0x4787c62a;

t[7]=0xa8304613;

t[8]=0xfd469501;

t[9]=0x698098d8;

t[10]=0x8b44f7af;

t[11]=0xffff5bb1;

t[12]=0x895cd7be;

t[13]=0x6b901122;

t[14]=0xfd987193;

t[15]=0xa679438e;

t[16]=0x49b40821;

t[17]=0xf61e2562;

t[18]=0xc040b340;

t[19]=0x265e5a51;

t[20]=0xe9b6c7aa;

t[21]=0xd62f105d;

t[22]=0x2441453;

t[23]=0xd8a1e681;

t[24]=0xe7d3fbc8;

t[25]=0x21e1cde6;

t[26]=0xc33707d6;

t[27]=0xf4d50d87;

t[28]=0x455a14ed;

t[29]=0xa9e3e905;

t[30]=0xfcefa3f8;

t[31]=0x676f02d9;

t[32]=0x8d2a4c8a;

t[33]=0xfffa3942;

t[34]=0x8771f681;

t[35]=0x6d9d6122;

t[36]=0xfde5380c;

t[37]=0xa4beea44;

t[38]=0x4bdecfa9;

t[39]=0xf6bb4b60;

t[40]=0xbebfbc70;

t[41]=0x289b7ec6;

t[42]=0xeaa127fa;

t[43]=0xd4ef3085;

t[44]=0x4881d05;

t[45]=0xd9d4d039;

t[46]=0xe6db99e5;

t[47]=0x1fa27cf8;

t[48]=0xc4ac5665;

t[49]=0xf4292244;

t[50]=0x432aff97;

t[51]=0xab9423a7;

t[52]=0xfc93a039;

t[53]=0x655b59c3;

t[54]=0x8f0ccc92;

t[55]=0xffeff47d;

t[56]=0x85845dd1;

t[57]=0x6fa87e4f;

t[58]=0xfe2ce6e0;

t[59]=0xa3014314;

t[60]=0x4e0811a1;

t[61]=0xf7537e82;

t[62]=0xbd3af235;

t[63]=0x2ad7d2bb;

t[64]=0xeb86d391;

char msg[100];

printf("Enter the message:\n");

scanf("%s",msg);

unsigned int n=0,i,len = strlen(msg),ln[64],k=0,x,j,r,m,p,arr[32];

n = (len\*8)+64;

x = i = n/512+(n%512!=0);

n = (512\*i)-64;

unsigned int msg\_512[n+64];

memset(msg\_512,0,sizeof(msg\_512));

for(i=0;i<len;i++){

k = k+8;

binary(msg\_512,msg[i],k-1);

}

if(len\*8<n)

msg\_512[len\*8]=1;

n=n+64;

binary(msg\_512,len\*8,n-1);

unsigned int mod=pow(2,32);

for(i=0;i<x;i++){

for(r=0;r<4;r++){

for(j=0;j<16;j++){

P[0]=(B&C)|((!B)&D);

P[1]=(B&D)|(C&(!D));

P[2]=B^C^D;

P[3]=C^(B|(!D));

m=0;

for(p=i\*512+32\*j;p<i\*512+32\*j+32;p++){

m += msg\_512[p]\*pow(2,31-(p-(i\*512+32\*j)));

}

A = (A+P[r]+m+t[r\*16+j+1])%mod;

memset(arr,0,sizeof(arr));

binary(arr,A,31);

lshift(arr,shift[r][j%4],0,31);

m=0;

for(p=0;p<32;p++){

m += arr[p]\*pow(2,31-p);

}

A = (m+B)%mod;

temp = B;

B = A;

A = D;

D = C;

C = temp;

}

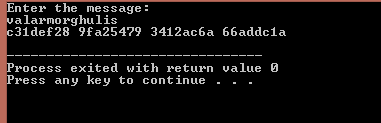
}

}

printf("%x %x %x %x\n",A,B,C,D);

return 0;

}



Md5 Algorithm