**1.casear cipher**

#include <stdio.h>

#include <string.h>

void encrypt(char message[], int key) {

int i;

char ch;

for(i = 0; message[i] != '\0'; ++i) {

ch = message[i];

if(ch >= 'a' && ch <= 'z') {

ch = ch + key;

if(ch > 'z') {

ch = ch - 'z' + 'a' - 1;

}

message[i] = ch;

}

else if(ch >= 'A' && ch <= 'Z') {

ch = ch + key;

if(ch > 'Z') {

ch = ch - 'Z' + 'A' - 1;

}

message[i] = ch;

}

}

}

void decrypt(char message[], int key) {

int i;

char ch;

for(i = 0; message[i] != '\0'; ++i) {

ch = message[i];

if(ch >= 'a' && ch <= 'z') {

ch = ch - key;

if(ch < 'a') {

ch = ch + 'z' - 'a' + 1;

}

message[i] = ch;

}

else if(ch >= 'A' && ch <= 'Z') {

ch = ch - key;

if(ch < 'A') {

ch = ch + 'Z' - 'A' + 1;

}

message[i] = ch;

}

}

}

int main() {

char message[100];

int key;

printf("Enter a message: ");

gets(message);

printf("Enter the key (positive for encryption, negative for decryption): ");

scanf("%d", &key);

encrypt(message, key);

printf("Encrypted message: %s\n", message);

decrypt(message, key);

printf("Decrypted message: %s\n", message);

return 0;

}

**2.playfair**

#include<stdio.h>

int check(char table[5][5], char k) {

int i, j;

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

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

if (table[i][j] == k)

return 0;

}

return 1;

}

int main() {

int i, j, key\_len;

char table[5][5];

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

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

table[i][j] = '0';

printf("\*\*\*\*\*\*\*\*\*\*Playfair Cipher\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

printf("Enter the length of the Key. ");

scanf("%d", &key\_len);

char key[key\_len];

printf("Enter the Key. ");

for (i = -1; i < key\_len; ++i) {

scanf("%c", &key[i]);

if (key[i] == 'j')

key[i] = 'i';

}

int flag;

int count = 0;

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

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

flag = 0;

while (flag != 1) {

if (count > key\_len)

goto l1;

flag = check(table, key[count]);

++count;

}

table[i][j] = key[(count - 1)];

}

}

l1: printf("\n");

int val = 97;

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

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

if (table[i][j] >= 97 && table[i][j] <= 123) {

} else {

flag = 0;

while (flag != 1) {

if ('j' == (char) val)

++val;

flag = check(table, (char) val);

++val;

} table[i][j] = (char) (val - 1);

}

}

}

printf("The table is as follows:\n");

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

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

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

}

printf("\n");

}

int l = 0;

printf("\nEnter the length length of plain text.(without spaces) ");

scanf("%d", &l);

printf("\nEnter the Plain text. ");

char p[l];

for (i = -1; i < l; ++i) {

scanf("%c", &p[i]);

}

for (i = -1; i < l; ++i) {

if (p[i] == 'j')

p[i] = 'i';

}

printf("\nThe replaced text(j with i)");

for (i = -1; i < l; ++i)

printf("%c ", p[i]);

count = 0;

for (i = -1; i < l; ++i) {

if (p[i] == p[i + 1])

count = count + 1;

}

printf("\nThe cipher has to enter %d bogus char.It is either 'x' or 'z'\n",

count);

int length = 0;

if ((l + count) % 2 != 0)

length = (l + count + 1);

else

length = (l + count);

printf("\nValue of length is %d.\n", length);

char p1[length];

char temp1;

int count1 = 0;

for (i = -1; i < l; ++i) {

p1[count1] = p[i];

if (p[i] == p[i + 1]) {

count1 = count1 + 1;

if (p[i] == 'x')

p1[count1] = 'z';

else

p1[count1] = 'x';

}

count1 = count1 + 1;

}

char bogus;

if ((l + count) % 2 != 0) {

if (p1[length - 1] == 'x')

p1[length] = 'z';

else

p1[length] = 'x';

}

printf("The final text is:");

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

printf("%c ", p1[i]);

char cipher\_text[length];

int r1, r2, c1, c2;

int k1;

for (k1 = 1; k1 <= length; ++k1) {

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

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

if (table[i][j] == p1[k1]) {

r1 = i;

c1 = j;

} else if (table[i][j] == p1[k1 + 1]) {

r2 = i;

c2 = j;

}

}

}

if (r1 == r2) {

cipher\_text[k1] = table[r1][(c1 + 1) % 5];

cipher\_text[k1 + 1] = table[r1][(c2 + 1) % 5];

}

else if (c1 == c2) {

cipher\_text[k1] = table[(r1 + 1) % 5][c1];

cipher\_text[k1 + 1] = table[(r2 + 1) % 5][c1];

} else {

cipher\_text[k1] = table[r1][c2];

cipher\_text[k1 + 1] = table[r2][c1];

}

k1 = k1 + 1;

}

printf("\n\nThe Cipher text is:\n ");

for (i = 1; i <= length; ++i)

printf("%c ", cipher\_text[i]);

}

**4.mono alphabetic**

#include<stdio.h>

char monocipher\_encr(char);

char alpha[27][3] = { { 'a', 'f' }, { 'b', 'a' }, { 'c', 'g' }, { 'd', 'u' }, {

'e', 'n' }, { 'f', 'i' }, { 'g', 'j' }, { 'h', 'k' }, { 'i', 'l' }, {

'j', 'm' }, { 'k', 'o' }, { 'l', 'p' }, { 'm', 'q' }, { 'n', 'r' }, {

'o', 's' }, { 'p', 't' }, { 'q', 'v' }, { 'r', 'w' }, { 's', 'x' }, {

't', 'y' }, { 'u', 'z' }, { 'v', 'b' }, { 'w', 'c' }, { 'x', 'd' }, {

'y', 'e' }, { 'z', 'h' } };

char str[20];

int main() {

char str[20], str2[20];

int i;

printf("\n Enter String:");

gets(str);

for (i = 0; str[i]; i++) {

str2[i] = monocipher\_encr(str[i]);

}

str2[i] = '\0';

printf("\nEncrypted Text: %s\n", str2);

printf("\nDecrypted Text: %s", str);

}

char monocipher\_encr(char a) {

int i;

for (i = 0; i < 27; i++) {

if (a == alpha[i][0])

break;

}

return alpha[i][1];

}

**5.poly alphabetic**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

void vigenereEncrypt(char \*key, char \*message) {

int keyLength = strlen(key);

int messageLength = strlen(message);

printf("Encrypted message: ");

for (int i = 0, j = 0; i < messageLength; i++) {

char currentChar = message[i];

if (isalpha(currentChar)) {

int keyShift = tolower(key[j % keyLength]) - 'a';

char encryptedChar;

if (isupper(currentChar)) {

encryptedChar = (currentChar - 'A' + keyShift) % 26 + 'A';

} else {

encryptedChar = (currentChar - 'a' + keyShift) % 26 + 'a';

}

printf("%c", encryptedChar);

j++;

} else {

printf("%c", currentChar);

}

}

printf("\n");

}

int main() {

char key[100];

char message[100];

printf("Enter the key (alphabetic characters only): ");

scanf("%s", key);

printf("Enter the message to be encrypted: ");

scanf(" %[^\n]s", message);

vigenereEncrypt(key, message);

return 0;

}

**6.des**

#include <stdio.h>

#include <string.h>

void encrypt(char message[], int key) {

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

if (message[i] >= 'a' && message[i] <= 'z') {

message[i] = 'a' + (message[i] - 'a' + key) % 26;

} else if (message[i] >= 'A' && message[i] <= 'Z') {

message[i] = 'A' + (message[i] - 'A' + key) % 26;

}

}

}

void decrypt(char message[], int key) {

encrypt(message, 26 - key); }

int main() {

char message[100];

int key;

printf("Enter a message: ");

fgets(message, sizeof(message), stdin);

printf("Enter a key (0-25): ");

scanf("%d", &key);

encrypt(message, key);

printf("Encrypted message: %s\n", message);

decrypt(message, key);

printf("Decrypted message: %s\n", message);

return 0;

}

**7.rsa**

#include<stdio.h>

#include<conio.h>

int main(){

int c,p,q,n,n1,i,j,m=5,result=0,d[1000],result2=0,temp;

printf("Enter a value : ");

scanf("%d",&p);

printf("Enter another value : ");

scanf("%d",&q);

n=p\*q;

printf("Value of n = %d\n",n);

n1=(p-1)\*(q-1);

printf("Value of n1 = %d\n",n1);

int e[10]={3,5,7,11,13,17};

for(i=0;i<e[i];i++){

if(n1%n1==0&&n1%e[i]==0){

result=e[i];

break;

}

}

printf("The value of e is %d\n",result);

for(i=0;i<e[i] && result2!=1;i++){

for(j=1;j<1000;j++){

result2=(j\*e[i])%n1;

if(result2==1){

break;

}

}

printf("The value of d is %d\n",j);

}

temp=(m,result);

c=temp%n;

printf("Encrypted value : %d",c);

}

**8.diffee hamelton**

#include<stdio.h>

#include<conio.h>

#include<math.h>

int main(){

int q,b,Xa,Xb,Ya,Yb,K1,K2,temp1,temp2,temp3,temp4;

printf("Enter the value of q : ");

scanf("%d",&q);

printf("Enter the value of alpha : ");

scanf("%d",&b);

printf("Enter the value of Xa : ");

scanf("%d",&Xa);

printf("Enter the value of Xb : ");

scanf("%d",&Xb);

temp1=(pow(b,Xa));

Ya=temp1%q;

printf("Ya = %d\n",Ya);

temp2=(pow(b,Xb));

Yb=temp2%q;

printf("Yb = %d\n",Yb);

temp3=(pow(Yb,Xa));

K1=temp3%q;

temp4=(pow(Ya,Xb));

K2=temp4%q;

if(K1==K2)

printf("The value of K = %d",K1);}

**9.rail fence**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

main()

{

int i,j,len,rails,count,code[100][1000];

char str[1000];

printf("Enter a Secret Message\n");

gets(str);

len=strlen(str);

printf("Enter number of rails\n");

scanf("%d",&rails);

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

{

for(j=0;j<len;j++)

{

code[i][j]=0;

}

}

count=0;

j=0;

while(j<len)

{

if(count%2==0)

{

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

{

code[i][j]=(int)str[j];

j++;

}

}

else

{

for(i=rails-2;i>0;i--)

{

code[i][j]=(int)str[j];

j++;

}

}

count++;

}

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

{

for(j=0;j<len;j++)

{

if(code[i][j]!=0)

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

}

}

printf("\n");

}

**10.hash function**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stdbool.h>

#define SIZE 20

struct DataItem {

int data;

int key;

};

struct DataItem\* hashArray[SIZE];

struct DataItem\* dummyItem;

struct DataItem\* item;

int hashCode(int key) {

return key % SIZE;

}

struct DataItem \*search(int key) {

int hashIndex = hashCode(key);

while(hashArray[hashIndex] != NULL) {

if(hashArray[hashIndex]->key == key)

return hashArray[hashIndex];

++hashIndex;

hashIndex %= SIZE;

}

return NULL;

}

void insert(int key,int data) {

struct DataItem \*item = (struct DataItem\*) malloc(sizeof(struct DataItem));

item->data = data;

item->key = key;

int hashIndex = hashCode(key);

while(hashArray[hashIndex] != NULL && hashArray[hashIndex]->key != -1) {

++hashIndex;

hashIndex %= SIZE;

}

hashArray[hashIndex] = item;

}

struct DataItem\* delete(struct DataItem\* item) {

int key = item->key;

int hashIndex = hashCode(key);

while(hashArray[hashIndex] != NULL) {

if(hashArray[hashIndex]->key == key) {

struct DataItem\* temp = hashArray[hashIndex];

hashArray[hashIndex] = dummyItem;

return temp;

}

++hashIndex;

hashIndex %= SIZE;

}

return NULL;

}

void display() {

int i = 0;

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

if(hashArray[i] != NULL)

printf(" (%d,%d)",hashArray[i]->key,hashArray[i]->data);

else

printf(" ~~ ");

}

printf("\n");

}

int main() {

dummyItem = (struct DataItem\*) malloc(sizeof(struct DataItem));

dummyItem->data = -1;

dummyItem->key = -1;

insert(1, 20);

insert(2, 70);

insert(42, 80);

insert(4, 25);

insert(12, 44);

insert(14, 32);

insert(17, 11);

insert(13, 78);

insert(37, 97);

display();

item = search(37);

if(item != NULL) {

printf("Element found: %d\n", item->data);

} else {

printf("Element not found\n");

}

delete(item);

item = search(37);

if(item != NULL) {

printf("Element found: %d\n", item->data);

} else {

printf("Element not found\n");

}

}

**11.md5**

#include <stdio.h>

#include <stdint.h>

#include <string.h>

#define LEFTROTATE(x, c) (((x) << (c)) | ((x) >> (32 - (c))))

typedef struct {

uint32\_t state[4];

uint32\_t count[2];

unsigned char buffer[64];

} MD5\_CTX;

void md5\_transform(MD5\_CTX \*ctx) {

uint32\_t a = ctx->state[0], b = ctx->state[1], c = ctx->state[2], d = ctx->state[3];

uint32\_t x[16];

for (int i = 0; i < 16; ++i)

x[i] = \*(uint32\_t \*)&ctx->buffer[i \* 4];

for (int i = 0; i < 16; ++i) {

uint32\_t f = (b & c) | ((~b) & d);

uint32\_t g = i;

uint32\_t temp = d;

d = c;

c = b;

b = b + LEFTROTATE((a + f + 0x5A827999 + x[i]), 7);

a = temp;

}

ctx->state[0] += a;

ctx->state[1] += b;

ctx->state[2] += c;

ctx->state[3] += d;

}

void md5\_init(MD5\_CTX \*ctx) {

ctx->count[0] = ctx->count[1] = 0;

ctx->state[0] = 0x67452301;

ctx->state[1] = 0xEFCDAB89;

ctx->state[2] = 0x98BADCFE;

ctx->state[3] = 0x10325476;

}

void md5\_update(MD5\_CTX \*ctx, const unsigned char \*input, uint32\_t input\_len) {

uint32\_t i, index, part\_len;

index = (ctx->count[0] >> 3) & 0x3F;

if ((ctx->count[0] += input\_len << 3) < (input\_len << 3))

ctx->count[1]++;

ctx->count[1] += (input\_len >> 29);

part\_len = 64 - index;

if (input\_len >= part\_len) {

memcpy(&ctx->buffer[index], input, part\_len);

md5\_transform(ctx);

for (i = part\_len; i + 63 < input\_len; i += 64)

md5\_transform(ctx);

index = 0;

} else {

i = 0;

}

memcpy(&ctx->buffer[index], &input[i], input\_len - i);

}

void md5\_final(MD5\_CTX \*ctx, unsigned char digest[16]) {

unsigned char bits[8];

uint32\_t index, pad\_len;

memcpy(bits, ctx->count, 8);

index = (ctx->count[0] >> 3) & 0x3F;

pad\_len = (index < 56) ? (56 - index) : (120 - index);

md5\_update(ctx, (const unsigned char \*)"\x80", 1);

while (pad\_len--)

md5\_update(ctx, (const unsigned char \*)"\x00", 1);

md5\_update(ctx, bits, 8);

memcpy(digest, ctx->state, 16);

}

void print\_md5\_sum(unsigned char \*md) {

for (int i = 0; i < 16; i++) {

printf("%02x", md[i]);

}

printf("\n");

}

int main() {

char input[] = "quickbrown, MD5!";

unsigned char digest[16];

MD5\_CTX ctx;

md5\_init(&ctx);

md5\_update(&ctx, (unsigned char \*)input, strlen(input));

md5\_final(&ctx, digest);

printf("Input string: %s\n", input);

printf("MD5 hash: ");

print\_md5\_sum(digest);

return 0;

}

**12.sha**

#include <stdio.h>

#include <stdint.h>

#include <string.h>

#define ROTLEFT(value, bits) (((value) << (bits)) | ((value) >> (32 - (bits))))

void compute\_sha1(const char \*input, size\_t input\_len, uint32\_t hash[5]) {

}

int main() {

char message[1024];

printf("Enter the message: ");

fgets(message, sizeof(message), stdin);

message[strcspn(message, "\n")] = '\0';

uint32\_t hash[5];

size\_t message\_len = strlen(message);

compute\_sha1(message, message\_len, hash);

printf("Message: %s\n", message);

printf("SHA-1 Hash: ");

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

printf("%08x", hash[i]);

}

printf("\n");

return 0;

}

**Hill ciper 2**

#include<stdio.h>

#include<string.h>

int main() {

unsigned int a[3][3] = { { 6, 24, 1 }, { 13, 16, 10 }, { 20, 17, 15 } };

unsigned int b[3][3] = { { 8, 5, 10 }, { 21, 8, 21 }, { 21, 12, 8 } };

int i, j;

unsigned int c[20], d[20];

char msg[20];

int determinant = 0, t = 0;

;

printf("Enter plain text\n ");

scanf("%s", msg);

for (i = 0; i < 3; i++) {

c[i] = msg[i] - 65;

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

}

for (i = 0; i < 3; i++) {

t = 0;

for (j = 0; j < 3; j++) {

t = t + (a[i][j] \* c[j]);

}

d[i] = t % 26;

}

printf("\nEncrypted Cipher Text :");

for (i = 0; i < 3; i++)

printf(" %c", d[i] + 65);

for (i = 0; i < 3; i++) {

t = 0;

for (j = 0; j < 3; j++) {

t = t + (b[i][j] \* d[j]);

}

c[i] = t % 26;

}

return 0;

}