

## **EXPERIMENT-1: IMPLEMENTATION OF XOR OPERATION ON THE STRING**

**AIM:** To write a C program that contains a string (char pointer) with a value 'Hello world'. The program should XOR each character in this string with 0 and displays the result

### **DESCRIPTION:**


(exclusive **OR**) A Boolean logic operation that is widely used in cryptography as well as in generating parity bits for error checking and fault tolerance. XOR compares two input bits and generates one output bit. The logic is simple. If the bits are the same, the result is 0. If the bits are different, the result is 1.

X	Y	X^Y
0	0	0
0	1	1
1	0	1
1	1	0

### **PROGRAM:**

```
#include<stdio.h>
#include<string.h>
void main() {
    char a[] = "Hello World",b[11];
    int len = strlen(a);
    for (int i=0;i<len;i++)
    {
        b[i] = a[i] ^ 0;
        printf("%c", b[i]);
    }
}
```

## OUTPUT

A screenshot of a console window with a black background and green text. The text displayed is "Hello World" on the first line, followed by "...Program finished with exit code 0" on the second line, and "Press ENTER to exit console." on the third line. A white cursor is positioned at the end of the third line. The window has a standard title bar with minimize, maximize, and close buttons visible at the top left.

```
Hello World

...Program finished with exit code 0
Press ENTER to exit console.
```

## **EXPERIMENT-2: IMPLEMENTATION OF AND, OR, XOR A STRING WITH A 127**

**AIM:** To Write a C program that contains a string (char pointer) with a value 'Hello world'. The program should perform AND, OR and XOR each character in this string with 127 and display the result.

### **DESCRIPTION:**

#### **AND**

When we use the AND operator, we are effectively saying, if the first argument and the second argument are true, then the result is true, or 1. Otherwise the result is false, or 0.

0 AND 0 = 0  
1 AND 0 = 0  
0 AND 1 = 0  
1 AND 1 = 1

AND is represented by the ampersand -  $1 \& 1 = 1$

#### **OR**

Now the OR operator is saying, if the first argument or the second argument are true, then the result is true.

0 OR 0 = 0  
1 OR 0 = 1  
0 OR 1 = 1  
1 OR 1 = 1

OR is represented by the vertical bar (pipe) -  $1 | 1 = 1$

#### **XOR**

Lastly, the XOR (exclusive OR) operator is saying, if either input is true, then the result is true, but if both inputs are true, then the result is false. Another way to say it is, if one, but not both arguments are true, then the result is true. Or, we could say, if the number of true inputs is odd, then the result will be true.

You choose.  
0 XOR 0 = 0  
1 XOR 0 = 1  
0 XOR 1 = 1  
1 XOR 1 = 0

XOR is represented by the upwards caret -  $1 \wedge 1 = 1$

### **PROGRAM:**

```
#include<stdlib.h>

#include<stdio.h>

#include<string.h>

int main() {

    printf("Enter the text : ");

    char a[100],b[100];

    scanf("%[^\n]%*c",a);

    int len = strlen(a);

    //AND

    printf(" AND operation: ");

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

    {

        b[i] = a[i] & 127;

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

    }

    //OR

    printf("\n OR operation: ");

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

    {

        b[i] = a[i] | 127;

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

    }

    //XOR

    printf("\n XOR operation: ");

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

    {

        b[i] = a[i] ^ 127;

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

    }

    return 0;

}
```

## OUTPUT

```
Enter the text : Hello world
AND operation: Hello world
OR operation:
XOR operation: 7_

...Program finished with exit code 0
Press ENTER to exit console.□
```

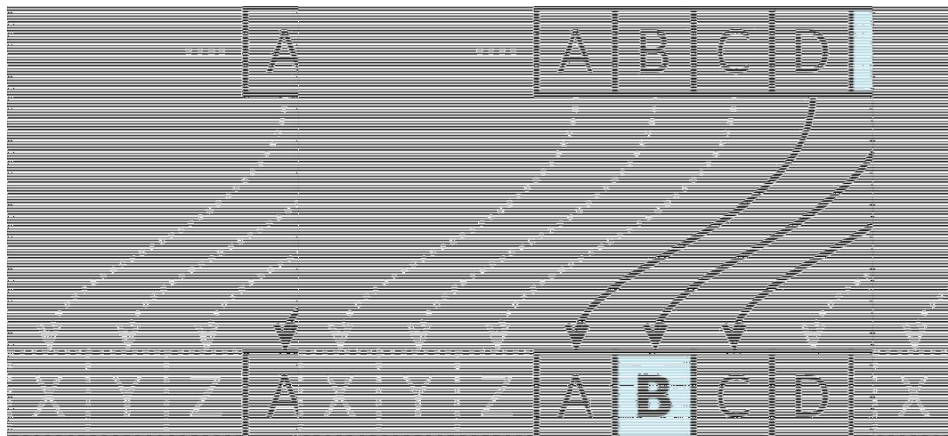
### **EXPERIMENT-3: IMPLEMENTATION OF SUBSTITUTION ALGORITHM(CAESAR CIPHER)**

**AIM-**To implement the Substitution technique named Caesar Cipher using C.

#### **DESCRIPTION**

To encrypt a message with a Caesar cipher, each letter in the message is changed using a simple rule: shift by three. Each letter is replaced by the letter three letters ahead in the alphabet. A becomes D, B becomes E, and so on. For the last letters, we can think of the alphabet as a circle and "wrap around". W becomes Z, X becomes A, Y becomes B, and Z becomes C. To change a message back, each letter is replaced by the one three before it.

#### **EXAMPLE**



#### **PROGRAM:**

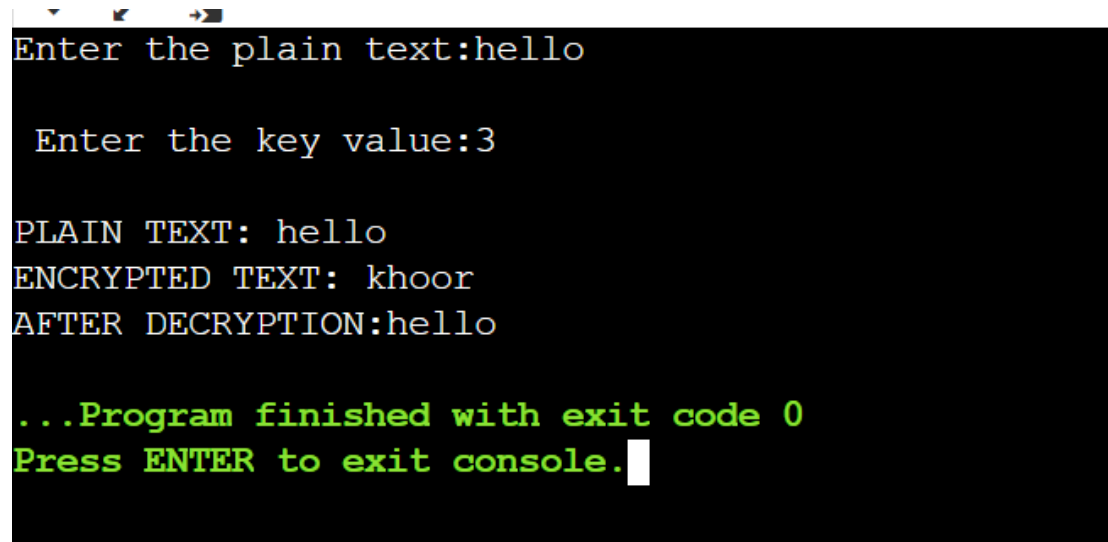
```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <ctype.h>
void main()
{
    char plain[10], cipher[10];
    int key,i,length;
```

```

int result;
printf("Enter the plain text:");
scanf("%s", plain);
printf("\n Enter the key value:");
scanf("%d", &key);
printf("\nPLAIN TEXT: %s", plain);
printf("\nENCRYPTED TEXT: ");
for(i = 0, length = strlen(plain); i < length; i++)
{
    cipher[i]=plain[i] + key;
    if (isupper(plain[i]) && (cipher[i] > 'Z'))
    {
        cipher[i] = cipher[i] - 26;
    }
    if (islower(plain[i]) && (cipher[i] > 'z')){
        cipher[i] = cipher[i] - 26;
    }
    printf("%c", cipher[i]);
}
printf("\nAFTER DECRYPTION:");
for(i=0;i<length;i++)
{
    plain[i]=cipher[i]-key;
    if(isupper(cipher[i])&&(plain[i]<'A'))
    {
        plain[i]=plain[i]+26;
    }
    if(islower(cipher[i])&&(plain[i]<'a'))
    {
        plain[i]=plain[i]+26;
    }
    printf("%c", plain[i]);
}
}

```

Output:



```
Enter the plain text:hello

Enter the key value:3

PLAIN TEXT: hello
ENCRYPTED TEXT: khood
AFTER DECRYPTION:hello

...Program finished with exit code 0
Press ENTER to exit console.
```



### **EXPERIMENT-3: IMPLEMENTATION OF SUBSTITUTION ALGORITHM(PLAYFAIR CIPHER)**

**AIM-**To implement the Substitution technique named Playfair Cipher using C.

#### **DESCRIPTION:**

The Playfair cipher starts with creating a key table. The key table is a 5×5 grid of letters that will act as the key for encrypting your plaintext. Each of the 25 letters must be unique and one letter of the alphabet is omitted from the table (as there are 25 spots and 26 letters in the alphabet).

To encrypt a message, one would break the message into digrams (groups of 2 letters) such that, for example, "HelloWorld" becomes "HE LL OW OR LD", and map them out on the key table. The two letters of the diagram 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:

- i. If both letters are the same (or only one letter is left), add an "X" after the first letter
- ii. If the letters appear on the same row of your table, replace them with the letters to their immediate right respectively
- iii. If the letters appear on the same column of your table, replace them with the letters immediately below respectively
- iv. 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.

#### **EXAMPLE**

D. Playfair Cipher

**Example1: Plaintext:** CRYPTO IS TOO EASY    **Key =** INFOSEC    **Ciphertext: ??**

**Grouped text:** CR YP TO IS TO XO EA SY

**Ciphertext:** AQ TV YB NI YB YF CB OZ

I / J	N	F	O	S
E	C	A	B	D
G	H	K	L	M
P	Q	R	T	U
V	W	X	Y	Z

**PROGRAM: (Playfair Cipher)**

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
int remrep(int size,int a[]);
int positioning(int position,int a[],int size);
void main()
{
    int i,j,k,ns[100],nc[100],nk[100],lk,tl,tk[100],flag=-
1,size,cipherkey[5][5],lenns,row1,row2,col1,col2;
    char str[100],key[100];
    printf("Enter the message\n");
    gets(str);
    for(i=0,j=0;i<strlen(str);i++)
    {
        if(str[i]!=' ')
        {
            str[j]=toupper(str[i]);
            j++;
        }
    }
    str[j]='\0';
    printf("Entered String is %s\n",str);
    size=strlen(str);
    for(i=0;i<size;i++)
    {
        if(str[i]!=' ')
        ns[i]=str[i]-'A';
    }
    lenns=i;
    printf("Enter the key\n");
    gets(key);
    for(i=0,j=0;i<strlen(key);i++)
    {
        if(key[i]!=' ')
        {
            key[j]=toupper(key[i]);
            j++;
        }
    }
    key[j]='\0';
    printf("%s\n",key);
    k=0;
    for(i=0;i<strlen(key)+26;i++)
    {
        if(i<strlen(key))
        {
            if(key[i]=='J')
            {
                flag=8;
                printf("%d",flag);
            }
            nk[i]=key[i]-'A';
        }
        else
```

```

{
    if(k!=9 && k!=flag)
    {
        nk[i]=k;
    }
    k++;
}
}
tl=i;
lk=remrep(tl,nk);
printf("Play Fair Cipher rule\n");
for(i=0;i<lk;i++)
{
    printf("%c",nk[i]+'A');
}
printf("\n");
k=0;
for(i=0;i<5;i++)
{
    for(j=0;j<5;j++)
    {
        cipherkey[i][j]=nk[k];
        k++;
    }
}
printf("Arranged key\n");
for(i=0;i<5;i++)
{
    for(j=0;j<5;j++)
    {
        printf("%c ",cipherkey[i][j]+'A');
    }
    printf("\n");
}
for(i=0;i<lenns;i+=2)
{
    if(ns[i]==ns[i+1])
    {
        positioning(i+1,ns,lenns);
        lenns++;
    }
}
if(lenns%2!=0)
{
    positioning(lenns,ns,lenns);
    lenns++;
}
printf("Entered Message After Processing according to Play fair cipher rule\n");
for(i=0;i<lenns;i++)
{
    printf("%c",ns[i]+'A');
}
for(k=0;k<lenns;k+=2)
{
    for(i=0;i<5;i++)

```

```

{
for(j=0;j<5;j++)
{
if(ns[k]==cipherkey[i][j])
{
row1=i;
col1=j;
}
if(ns[k+1]==cipherkey[i][j])
{
row2=i;
col2=j;
}
}
}
if(row1==row2)
{
col1=(col1-1)%5;
col2=(col2-1)%5;
if(col1<0)
{
col1=5+col1;
}
if(col2<0)
{
col2=5+col2;
}
nc[k]=cipherkey[row1][col1];
nc[k+1]=cipherkey[row2][col2];
}
if(col1==col2)
{
row1=(row1-1)%5;
row2=(row2-1)%5;
if(row1<0)
{
row1=5+row1;
}
if(row2<0)
{
row2=5+row2;
}
nc[k]=cipherkey[row1][col1];
nc[k+1]=cipherkey[row2][col2];
}
if(row1!=row2&&col1!=col2)
{
nc[k]=cipherkey[row1][col2];
nc[k+1]=cipherkey[row2][col1];
}
}
printf("\nCipher Text is\n");
for(i=0;i<lenns;i++)
{
if((nc[i]+'A')!='X')

```

```

        printf("%c",nc[i]+'A');
    }
    printf("\n");
}
int remrep(int size,int a[])
{
    int i,j,k;
    for(i=0;i<size;i++)
    {
        for(j=i+1;j<size;)
        {
            if(a[i]==a[j])
            {
                for(k=j;k<size;k++)
                {
                    a[k]=a[k+1];
                }
                size--;
            }
            else
            {
                j++;
            }
        }
    }
    return(size);
}
int positioning(int position,int a[],int size)
{
    int i,insitem=23,temp[size+1];
    for(i=0;i<=size;i++)
    {
        if(i<position)
        {
            temp[i]=a[i];
        }
        if(i>position)
        {
            temp[i]=a[i-1];
        }
        if(i==position)
        {
            temp[i]=insitem;
        }
    }
    for(i=0;i<=size;i++)
    {
        a[i]=temp[i];
    }
}

```

## OUTPUT:

```
Enter the message
cryptography
Entered String is CRYPTOGRAPHY
Enter the key
hello
HELLO
Play Fair Cipher rule
HEL0ABCDEFGHIJKMNPQRSTUVWXYZ
Arranged key
H E L O A
B C D F G
I J K M N
P Q R S T
U V W X Y
Entered Message After Processing according to Play fair cipher rule
CRYPTOGRAPHY
Cipher Text is
DQUTSADTHTAU
```

### **EXPERIMENT-3:IMPLEMENTATION OF SUBSTITUTION ALGORITHM(HILLCIPHER)**

**AIM:** To write a C program to implement the hill cipher substitution techniques.

#### **DESCRIPTION:**

Each letter is represented by a number modulo 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 is multiplied by an invertible  $n \times n$  matrix, against modulus 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, and it should be chosen randomly from the set of invertible  $n \times n$  matrices (modulo 26).

#### **EXAMPLE:**

$$\begin{bmatrix} 2 & 4 & 5 \\ 9 & 2 & 1 \\ 3 & 17 & 7 \end{bmatrix} \begin{bmatrix} 0 \\ 19 \\ 19 \end{bmatrix} = \begin{bmatrix} 171 \\ 57 \\ 456 \end{bmatrix} \pmod{26} = \begin{bmatrix} 15 \\ 5 \\ 14 \end{bmatrix} = \text{'PFO'}$$

#### **PROGRAM:**

```
#include<stdio.h>
#include<conio.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, t=0;
    unsigned int c[20],d[20];
    char msg[20];
    clrscr();
    printf("Enter plain text\n ");
    scanf("%s",msg);
    for(i=0;i<strlen(msg);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;
    }
    printf("\nDecrypted Cipher Text :");
    for(i=0;i<3;i++)
    printf(" %c",c[i]+65);
    getch();
    return 0;
}

```



## OUTPUT



A screenshot of the Turbo C++ IDE window. The title bar is blue and reads "Turbo C++ IDE". The main window has a black background with white text. The text displayed is as follows:

```
Enter plain text
ACT
0 2 19
Encrypted Cipher Text : P O H
Decrypted Cipher Text : A C T
```

### **EXPERIMENT-3:IMPLEMENTATION OF TRANSPOSITION TECHNIQUES(RAILFENCE ALGORITHM)**

**AIM:** To write a C program to implement the rail fence transposition technique.

#### **DESCRIPTION:**

In the rail fence cipher, the plain text is written downwards and diagonally on successive "rails" of an imaginary fence, then moving up when we reach the bottom rail. When we reach the top rail, the message is written downwards again until the whole plaintext is written out. The message is then read off in rows.

#### **EXAMPLE**

	A	U	T	H	O	R
	1	6	5	2	3	4
W	E	A	R	E	D	
I	S		C	O	V	E
R	E	D	S	A	V	
E	Y	O	U	R	S	
E	L	F	A	B	C	

yields the cipher

WIREEROSUA E V A R B D E V S C A C D O F E S E Y L .

#### **PROGRAM:**

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()
{
    int i,j,k,l;
    char a[20],c[20],d[20];
    printf("\nRAIL FENCE TECHNIQUE");
    printf("\nEnter the input string : ");
```

```

scanf("%[^\\n]s",a);
l=strlen(a);
for(i=0,j=0;i<l;i++)
{
if(i%2==0)
    c[j++]=a[i];
}
for(i=0;i<l;i++)
{
if(i%2==1)
    c[j++]=a[i];
}
c[j]='\0';
printf("\nCipher text of rail fence :");
printf("\n%s",c);
if(l%2==0)
    k=l/2;
else
    k=(l/2)+1;
for(i=0,j=0;i<k;i++)
{
    d[j]=c[i];
    j=j+2;
}
for(i=k,j=1;i<l;i++)
{
    d[j]=c[i];
    j=j+2;
}

```

```
}  
d[l]='\0';  
printf("\nDecryption : ");  
printf("%s",d);  
}
```

## OUTPUT:

```
RAIL FENCE TECHNIQUE
Enter the input string : computer science

Cipher text of rail fence :
cmue cecoptrsine
Decryption : computer science

...Program finished with exit code 0
Press ENTER to exit console.█
```

### **EXPERIMENT-3:IMPLEMENTATION OF TRANSPOSITION TECHNIQUES(ROW & COLUMN ALGORITHM)**

**AIM:** To write a C program to implement the row & column transposition technique.

#### **DESCRIPTION:**

In this technique, the message is written in adjacent rows and columns are permuted. Suppose that message is "meet me at the toga party at nine pm". The characters in the message are written in successive rows, each row having fixed number of columns. Then columns are permuted and cipher is generated. Example: Suppose that given message is written in 6 columns

M E E T M E  
A T T H E T O  
G A P A R T Y  
A T N I N E P  
M X Y

Column permutation: 4 1 3 2 5 6

The column permutation and the number of rows is the key for decryption. Here x, y are filler characters to complete the matrix.

Cipher text is ETGYETHPTMETAAPMAOTNMEANXETRIY

Decryption:

Key is the number of rows (5) and the column permutation (4 1 3 2 5 6). First 5 characters in cipher is written as column 2 (number 1 is in second position of the permutation of columns), next 5 characters as column 4 (number 2 is in fourth position of the permutation of columns) and so on.

#### **PROGRAM:**

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stdbool.h>

void sort_letters(char *phrase) {

    int n = strlen(phrase);

    for (int i = 0; i < n-1; i++) {
```

```

        for (int j = i+1; j < n; j++) {
            if (phrase[i] > phrase[j]) {
                char temp = phrase[i];
                phrase[i] = phrase[j];
                phrase[j] = temp;
            }
        }
    }
}

char** gridStr(char *line, char *key)
{
    int n = strlen(line);
    int columns = strlen(key);
    int rows = (n + columns - 1) / columns;
    char **matrix = malloc(rows * sizeof(char *));
    for (int j = 0; j < rows; j++) {
        matrix[j] = malloc(columns * sizeof(char));
    }
    int i = 0;
    for (int k = 0; k < rows; k++) {
        for (int l = 0; l < columns; l++) {
            if (i < n) {
                matrix[k][l] = line[i++];
            } else {
                matrix[k][l] = '*';
            }
        }
    }
    return matrix;
}

void swap_matrix_columns(char **matrix, int rows, int columns, int i, int j) {
    for (int k = 0; k < rows; k++) {
        char tmp = matrix[k][i];
        matrix[k][i] = matrix[k][j];
        matrix[k][j] = tmp;
    }
}

void print_matrix(char **matrix, int rows, int columns) {

```

```

for (int i = 0; i < rows; i++){
    for (int j = 0; j < columns; j++) {
        putchar(matrix[i][j]);
    }
    putchar('\n');
}
}

void encrypt(char **matrix, int rows, int columns, char *key) {
    char sorted_key[strlen(key)];
    strcpy(sorted_key, key);
    sort_letters(sorted_key);
    printf("Sorted key: %s\n", sorted_key);
    bool is_used[columns];
    for (int i = 0; i < columns; i++) is_used[i] = false;
    for (int i = 0; i < columns; i++) {
        if (!is_used[i]) {
            int j;
            for (j = 0; j < columns; j++) {
                if (key[i] == sorted_key[j] && !is_used[j]) {
                    break;
                }
            }
            swap_matrix_columns(matrix, rows, columns, i, j);
            is_used[i] = true;
            is_used[j] = true;
        }
    }
}

int main(void) {
    char key[50];
    char line[256];
    printf("Enter your string: ");
    if (fgets(line, sizeof line, stdin) == NULL) {
        fprintf(stderr, "No line read\n");
        exit(EXIT_FAILURE);
    }
    printf("Enter your key: ");
    if (fgets(key, sizeof key, stdin) == NULL) {

```



```

    fprintf(stderr, "No line read\n");
    exit(EXIT_FAILURE);
}
int len = strlen(line);
if (len && line[len - 1] == '\n') {
    line[--len] = '\0';
}
int len1 = strlen(key);
if (len1 && key[len1 - 1] == '\n') {
    key[--len1] = '\0';
}
char **matrix = gridStr(line, key);
int columns = len1;
int rows = (len + columns - 1) / columns;
print_matrix(matrix, rows, columns);
encrypt(matrix, rows, columns, key);
print_matrix(matrix, rows, columns);
printf("Encrypted String is ");
for(int i=0;i<rows;i++){
    for(int j=0;j<columns;j++){
        if(matrix[i][j]=='*'){
            break;
        }
        else{
            printf("%c",matrix[i][j]);
        }
    }
}
printf("\nDecrypted String is %s",line);
for (int row = 0; row < rows; row++) {
    free(matrix[row]);
}
free(matrix);
return EXIT_SUCCESS;
}

```

## OUTPUT:

```
Enter your string: cryptography
Enter your key: hello
crypt
ograp
hy***
Sorted key: ehlllo
rcrypt
gorap
yh***
Encrypted String is rcryptgorapyh
Decrypted String is cryptography
...Program finished with exit code 0
Press ENTER to exit console.□
```

## **EXPERIMENT-4:IMPLEMENTATION OF S-DES**

**AIM:** To implement the message encryption and decryption of 8-bit data using S-DES.

### **DESCRIPTION:**

Simplified Data Encryption Standard is a simple version of Data Encryption Standard having a 10-bit key and 8-bit plain text. It is much smaller than the DES algorithm as it takes only 8-bit plain text whereas DES takes 64-bit plain text. It was developed for educational purpose so that understanding DES can become easy. It is a block cipher algorithm and uses a symmetric key for its algorithm i.e. they use the same key for both encryption and decryption. It has 2 rounds for encryption which use two different keys.

Simplified Data Encryption Standard (S-DES) is equivalent to the DES algorithm. The SDES encryption algorithm produces an 8-bit block of plaintext (example: 10111101) and a 10-bit key as input and makes an 8-bit block of ciphertext as output. The S-DES decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key can develop that ciphertext as input and makes the initial 8-bit block of plaintext.

### **PROGRAM:**

```
#include<stdio.h>

#include<string.h>

int s_matrix0[4][4] = { { 1,0,3,2},
                        { 3,2,1,0},
                        { 0,2,1,3},
                        { 3,1,3,2} };

int s_matrix1[4][4] = { { 0,1,2,3},
                        { 2,0,1,3},
                        { 3,0,1,0},
                        { 2,1,0,3} };

int s0=0, s1=0;
```

```

int row=0, col=0;

int s0_binary[2], s1_binary[2];

int result[2];

int to_digit(int a, int b)
{
    int output;

    if(a==1 && b==1)

        output = 3;

    if(a==0 && b==1)

        output = 1;

    if(a==1 && b==0)

        output = 2;

    if(a==0 && b==0)

        output = 0;

    return output;
}

void to_binary(int num)
{
    int x;

    if(num == 3)

    {
        for(x=0; x<2; x++)

            result[x] = 1;

    }

    else if(num == 1)

    {

        result[0] = 0;

        result[1] = 1;

```

```

    }

    else if(num == 2)

    {

        result[0] = 1;

        result[1] = 0;

    }

    else if(num == 0)

    {

        for(x=0; x<2; x++)

            result[x] = 0;

    }

}

void main()

{

    int i,j,index;

    int k1[8], k2[8];

    int afterp10[11], ls1[10], ls2[10], afterip[8], afterrep[8], afterp4[4];

    int aftersboxesone[4], aftersboxestwo[4];

    int leftafterip[4], rightafterip[4];

    int leftafterrep[4], rightafterrep[4];

    int leftafterone[4], rightafterone[4];

    int afteripinverse[8];

    int afterone[8], aftertwo[8];

    int p10[10] = {3,5,2,7,4,10,1,9,8,6};

    int p8[8] = {6,3,7,4,8,5,10,9};

    int ip[8] = {2,6,3,1,4,8,5,7};

    int ep[8] = {4,1,2,3,2,3,4,1};

    int p4[4] = {2,4,3,1};

```

```

int ipinverse[8] = {4,1,3,5,7,2,8,6};

int key[11] = {1,0,1,0,0,0,0,1,0};

int plain[8] = {0,1,1,0,1,1,0,1};

printf("S-DES Key Generation and Encryption.\n");

printf("\n-----KEY GENERATION-----\n");

printf("\nEntered the 10-bit key is: ");

for(i=0; i<10; i++)

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

printf("\np10 permutation is defined as: ");

for(i=0; i<=9; i++)

    printf("%d ",p10[i]);

for(i=0; i<=9; i++)

{

    index = p10[i];

    afterp10[i] = key[index - 1];

}

afterp10[i] = '\0';

printf("\n\nAfter p10 = ");

for(i=0; i<=9; i++)

    printf("%d ",afterp10[i]);

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

{

    if(i == 4)

        ls1[i]=afterp10[0];

    else

        ls1[i]=afterp10[i+1];

}

for(i=5; i<10; i++)

```

```

{
    if(i == 9)
        ls1[i]=afterp10[5];
    else
        ls1[i]=afterp10[i+1];
}

printf("\n\nAfter LS-1 = ");

for(i=0; i<10; i++)

    printf("%d ", ls1[i]);

printf("\np8 permutation is defined as: ");

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

    printf("%d ", p8[i]);

index=0;

for(i=0; i<=9; i++)

{

    index = p8[i];

    k1[i] = ls1[index - 1];

}

printf("\n\n--->Key-1 is: ");

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

    printf("%d ", k1[i]);

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

{

    ls2[i]=ls1[i+2];

}

ls2[3]=ls1[0];

ls2[4]=ls1[1];

```

```

for(i=5; i<8; i++)

{

    ls2[i]=ls1[i+2];

}

ls2[8]=ls1[5];

ls2[9]=ls1[6];

printf("\n\nAfter LS-2 = ");

for(i=0; i<10; i++)

    printf("%d ", ls2[i]);

printf("\np8 permutation is defined as: ");

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

    printf("%d ", p8[i]);

index=0;

for(i=0; i<=9; i++)

{

    index = p8[i];

    k2[i] = ls2[index - 1];

}

printf("\n\n--->Key-2 is: ");

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

    printf("%d ", k2[i]);

printf("\n\n-----S-DES ENCRYPTION-----\n");

printf("\n--->Entered the 8-bit plaintext is: ");

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

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

printf("\nInitial permutation is defined as: ");

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

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

```



```

for(i=0; i<8; i++)
{
    index = ip[i];
    afterip[i] = plain[index - 1];
}
afterip[i] = '\0';
printf("\nAfter ip = ");
for(i=0; i<8; i++)
    printf("%d ", afterip[i]);
printf("\n\nExpand permutation is defined as: ");
for(i=0; i<8; i++)
    printf("%d ", ep[i]);
for(j=0; j<4; j++)
    lefterafterip[j] = afterip[j];
for(j=0; j<4; j++)
    rightafterip[j] = afterip[j+4];
for(i=0; i<4; i++)
{
    index = ep[i];
    afterep[i] = rightafterip[index - 1];
}
for(i=4; i<8; i++)
{
    index = ep[i];
    afterep[i] = lefterafterip[index - 1];
}
afterep[i] = '\0';

```

```

printf("\nAfter ep = ");

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

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

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

    k1[i] = k1[i] ^ afterep[i];

printf("\n\nAfter XOR operation with 1st Key= ");

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

    printf("%d ", k1[i]);

row = to_digit(k1[0],k1[3]);

col = to_digit(k1[1],k1[2]);

s0 = s_matrix0[row][col];

to_binary(s0);

for(j=0; j<2; j++)

    s0_binary[j] = result[j];

row = to_digit(k1[4],k1[7]);

col = to_digit(k1[5],k1[6]);

s1 = s_matrix1[row][col];

to_binary(s1);

for(j=0; j<2; j++)

    s1_binary[j] = result[j];

for(j=0; j<2; j++)

    aftersboxesone[j] = s0_binary[j];

for(i=0,j=2; i<2,j<4; i++,j++)

    aftersboxesone[j] = s1_binary[i];

printf("\n\nAfter first S-Boxes= ");

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

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

```

```

printf("\n\nP4 is defined as: ");

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

    printf("%d ",p4[i]);

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

{

    index = p4[i];

    afterp4[i] = aftersboxesone[index - 1];

}

afterp4[i] = '\0';

printf("\n\nAfter P4= ");

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

    printf("%d ", afterp4[i]);

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

    afterp4[i] = afterp4[i] ^ leftafterip[i];

printf("\n\nAfter XOR operation with left nibble of after ip= ");

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

    printf("%d ", afterp4[i]);

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

    afterone[i] = rightafterip[i];

for(i=0,j=4; i<4,j<8; i++,j++)

    afterone[j] = afterp4[i];

afterone[j] = '\0';

printf("\n\nAfter first part= ");

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

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

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

    leftafterone[j] = afterone[j];

```

```

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

    rightafterone[j] = afterone[j+4];

printf("\n\nExpand permutation is defined as: ");

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

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

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

{

    index = ep[i];

    afterep[i] = rightafterone[index - 1];

}

for(i=4; i<8; i++)

{

    index = ep[i];

    afterep[i] = rightafterone[index - 1];

}

afterep[i] = '\0';

printf("\n\nAfter second ep = ");

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

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

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

    k2[i] = k2[i] ^ afterep[i];

printf("\n\nAfter XOR operation with 2nd Key= ");

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

    printf("%d ", k2[i]);

row = to_digit(k2[0],k2[3]);

col = to_digit(k2[1],k2[2]);

s0 = s_matrix0[row][col];

to_binary(s0);

```

```

for(j=0; j<2; j++)

    s0_binary[j] = result[j];

row = to_digit(k2[4],k2[7]);

col = to_digit(k2[5],k2[6]);

s1 = s_matrix1[row][col];

to_binary(s1);

for(j=0; j<2; j++)

    s1_binary[j] = result[j];

for(j=0; j<2; j++)

    aftersboxestwo[j] = s0_binary[j];

for(i=0,j=2; i<2,j<4; i++,j++)

    aftersboxestwo[j] = s1_binary[i];

printf("\n\nAfter second S-Boxes= ");

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

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

printf("\n\nP4 is defined as: ");

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

    printf("%d ",p4[i]);

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

{

    index = p4[i];

    afterp4[i] = aftersboxestwo[index - 1];

}

afterp4[i] = '\0';

printf("\n\nAfter P4= ");

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

    printf("%d ", afterp4[i]);

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

```

```

    afterp4[i] = afterp4[i] ^ leftafterone[i];

printf("\n\nAfter XOR operation with left nibble of after first part= ");

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

    printf("%d ", afterp4[i]);

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

    aftertwo[i] = afterp4[i];

for(i=0,j=4; i<4,j<8; i++,j++)

    aftertwo[j] = rightafterone[i];

aftertwo[j] = '\0';

printf("\n\nAfter second part= ");

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

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

printf("\n\nInverse Initial permutation is defined as: ");

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

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

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

{

    index = ipinverse[i];

    afteripinverse[i] = aftertwo[index - 1];

}

afteripinverse[j] = '\0';

printf("\n\n-->8-bit Ciphertext will be= ");

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

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

}

```

## OUTPUT

```
input
S-DES Key Generation and Encryption.

-----KEY GENERATION-----
Entered the 10-bit key is: 1 0 1 0 0 0 0 1 0
p10 permutation is defined as: 3 5 2 7 4 10 1 9 8 6
After p10 = 1 0 0 0 0 1 1 0 0
After LS-1 = 0 0 0 0 1 1 1 0 0 0
p8 permutation is defined as: 6 3 7 4 8 5 10 9
---->Key-1 is: 1 0 1 0 0 1 0 0
After LS-2 = 0 0 1 0 0 0 0 1 1
p8 permutation is defined as: 6 3 7 4 8 5 10 9
---->Key-2 is: 0 1 0 0 0 1 1

-----S-DES ENCRYPTION-----
---->Entered the 8-bit plaintext is: 0 1 1 0 1 1 0 1
Initial permutation is defined as: 2 6 3 1 4 8 5 7
After ip = 1 1 1 0 0 1 1 0

Expand permutation is defined as: 4 1 2 3 2 3 4 1
After ep = 0 0 1 1 1 1 0 0

After XOR operation with 1st Key= 1 0 0 1 1 0 0 0
After first S-Boxes= 1 1 1 1
P4 is defined as: 2 4 3 1
After P4= 1 1 1 1

After XOR operation with left nibble of after ip= 0 0 0 1
After first part= 0 1 1 0 0 0 0 1

Expand permutation is defined as: 4 1 2 3 2 3 4 1
After second ep = 1 0 0 0 0 0 1 0

After XOR operation with 2nd Key= 1 1 0 0 0 0 0 1
After second S-Boxes= 0 1 1 0
P4 is defined as: 2 4 3 1
After P4= 1 0 1 0

After XOR operation with left nibble of after first part= 1 1 0 0
After second part= 1 1 0 0 0 0 0 1

Inverse Initial permutation is defined as: 4 1 3 5 7 2 8 6
--->8-bit Ciphertext will be= 0 1 0 0 0 1 1 0

...Program finished with exit code 0
Press ENTER to exit console.
```

## **EXPERIMENT-5: IMPLEMENTATION OF 64-BIT DATA USING DES ALGORITHM**

**AIM:** Implement the encryption and decryption of 64-bit data using DES Algorithm

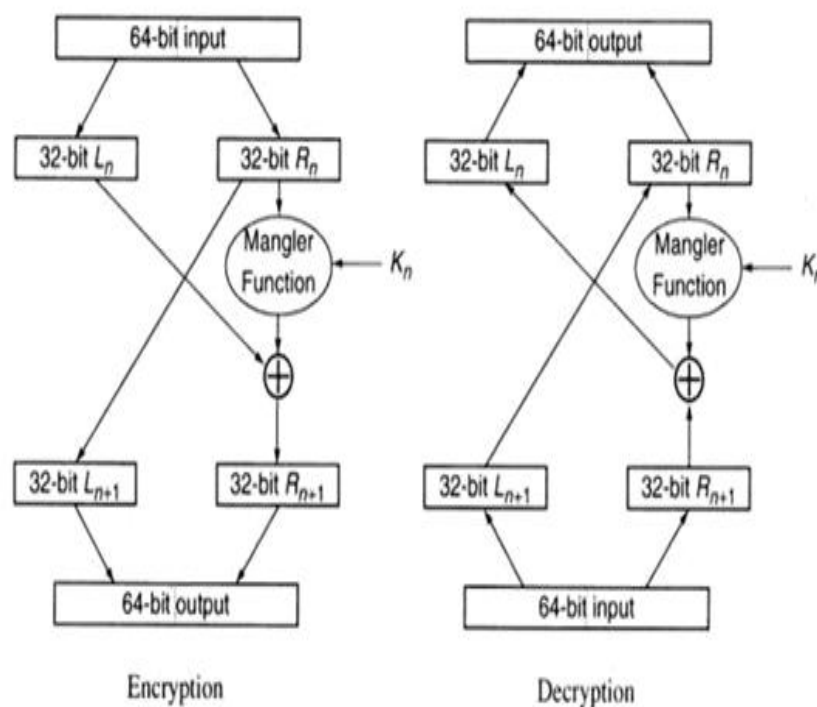
### **DESCRIPTION:**

DES is a symmetric encryption system that uses 64-bit blocks, 8 bits of which are used for parity checks. The key therefore has a "useful" length of 56 bits, which means that only 56 bits are actually used in the algorithm. The algorithm involves carrying out combinations, substitutions and permutations between the text to be encrypted and the key, while making sure the operations can be performed in both directions. The key is ciphered on 64 bits and made of 16 blocks of 4 bits, generally denoted  $k_1$  to  $k_{16}$ . Given that "only" 56 bits are actually used for encrypting, there can be  $2^{56}$  different keys.

The main parts of the algorithm are as follows:

1. Fractioning of the text into 64-bit blocks Initial permutation of blocks
2. Breakdown of the blocks into two parts: left and right, named L and R Permutation and substitution steps repeated 16 times
3. Re-joining of the left and right parts then inverse initial permutation.

### **EXAMPLE:**





## **PROGRAM:**

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

```
int Original_key [64] = {  
    0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0,  
    0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1,  
    1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0,  
    1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1  
};
```

```
int Permuted_Choice1[56] = {  
    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 Permuted_Choice2[48] = {  
    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 Initial_Permutation [64] = {  
    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 Final_Permutation[] =
```

```
{
```

```
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 P[] =
```

```
{
```

```
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 E[] =
```

```
{
```

```
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 S1[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
};
int S2[4][16]={
    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
};
int S3[4][16]={
    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
};
int S4[4][16]={
    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
};
int S5[4][16]={
    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
};
int S6[4][16]={
    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, 13, 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
};
int S7[4][16]={
    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
};
int S8[4][16]={
    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 shift_round[16] = { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };
int _56bit_key[56];
int _48bit_key[17][48];
int text_to_bits[99999], bits_size=0;
int Left32[17][32], Right32[17][32];
int et[48];
int XORtext[48];
int X[8][6];
int X2[32];
int R[32];
int chiper_text[64];
int encrypted_text[64];

```

```

int XOR(int a, int b) {
return (a ^ b);
}

void Dec_to_Binary(int n)
{
    int binaryNum[1000];
    int i = 0;
    while (n > 0) {
        binaryNum[i] = n % 2;
        n = n / 2;
        i++;
    }
    for (int j = i - 1; j >= 0; j--) {
        text_to_bits[bits_size++] = binaryNum[j];
    }
}

int F1(int i)
{
    int r, c, b[6];
    for (int j = 0; j < 6; j++)
        b[j] = X[i][j];
    r = b[0] * 2 + b[5];
    c = 8 * b[1] + 4 * b[2] + 2 * b[3] + b[4];
    if (i == 0)
        return S1[r][c];
    else if (i == 1)
        return S2[r][c];
    else if (i == 2)
        return S3[r][c];
    else if (i == 3)
        return S4[r][c];
    else if (i == 4)
        return S5[r][c];
}

```

```

else if (i == 5)
return S6[r][c];
else if (i == 6)
return S7[r][c];
else if (i == 7)
return S8[r][c];
}
int PBox(int pos, int bit)
{
    int i;
    for (i = 0; i < 32; i++)
        if (P[i] == pos + 1)
            break;
    R[i] = bit;
}
int ToBits(int value)
{
    int k, j, m;
    static int i;
    if (i % 32 == 0)
        i = 0;
    for (j = 3; j >= 0; j--)
    {
        m = 1 << j;
        k = value & m;
        if (k == 0)
            X2[3 - j + i] = '0' - 48;
        else
            X2[3 - j + i] = '1' - 48;
    }
    i = i + 4;
}

```

```

int SBox(int XORtext[])
{
    int k = 0;
    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 6; j++)
            X[i][j] = XORtext[k++];
    int value;
    for (int i = 0; i < 8; i++)
    {
        value = F1(i);
        ToBits(value);
    }
}

void expansion_function(int pos, int bit)
{
    for (int i = 0; i < 48; i++)
        if (E[i] == pos + 1)
            et[i] = bit;
}

void cipher(int Round, int mode)
{
    for (int i = 0; i < 32; i++)
        expansion_function(i, Right32[Round - 1][i]);
    for (int i = 0; i < 48; i++)
    {
        if (mode == 0)
            XORtext[i] = XOR(et[i], _48bit_key[Round][i]);
        else
            XORtext[i] = XOR(et[i], _48bit_key[17 - Round][i]);
    }
    SBox(XORtext);
    for (int i = 0; i < 32; i++)
        PBox(i, X2[i]);
}

```

```

for (int i = 0; i < 32; i++)
Right32[Round][i] = XOR(Left32[Round - 1][i], R[i]);
}
void finalPermutation(int pos, int bit)
{
int i;
for (i = 0; i < 64; i++)
if (Final_Permutation[i] == pos + 1)
break;
encrypted_text[i] = bit;
}
void Encrypt_each_64_bit (int plain_bits [])
{
int IP_result [64] , index=0;
for (int i = 0; i < 64; i++) {
IP_result [i] = plain_bits[ Initial_Permutation[i] ];
}
for (int i = 0; i < 32; i++)
Left32[0][i] = IP_result[i];
for (int i = 32; i < 64; i++)
Right32[0][i - 32] = IP_result[i];
for (int k = 1; k < 17; k++)
{
cipher(k, 0);
for (int i = 0; i < 32; i++)
Left32[k][i] = Right32[k - 1][i];
}
for (int i = 0; i < 64; i++)
{
if (i < 32)
cipher_text[i] = Right32[16][i];
else
cipher_text[i] = Left32[16][i - 32];
}
}

```



```

finalPermutation(i, chiper_text[i]);
}
for (int i = 0; i < 64; i++)
printf("%d", encrypted_text[i]);
}
void convert_Text_to_bits(char *plain_text){
for(int i=0;plain_text[i];i++){
int asci = plain_text[i];
Dec_to_Binary(asci);
}
}
void key56to48(int round, int pos, int bit)
{
int i;
for (i = 0; i < 56; i++)
if (Permutated_Choice2[i] == pos + 1)
break;
_48bit_key[round][i] = bit;
}
int key64to56(int pos, int bit)
{
int i;
for (i = 0; i < 56; i++)
if (Permutated_Choice1[i] == pos + 1)
break;
_56bit_key[i] = bit;
}
void key64to48(int key[])
{
int k, backup[17][2];
int CD[17][56];
int C[17][28], D[17][28];

```

```

for (int i = 0; i < 64; i++)
key64to56(i, key[i]);
for (int i = 0; i < 56; i++)
if (i < 28)
C[0][i] = _56bit_key[i];
else
D[0][i - 28] = _56bit_key[i];
for (int x = 1; x < 17; x++)
{
int shift = shift_round[x - 1];
for (int i = 0; i < shift; i++)
backup[x - 1][i] = C[x - 1][i];
for (int i = 0; i < (28 - shift); i++)
C[x][i] = C[x - 1][i + shift];
k = 0;
for (int i = 28 - shift; i < 28; i++)
C[x][i] = backup[x - 1][k++];
for (int i = 0; i < shift; i++)
backup[x - 1][i] = D[x - 1][i];
for (int i = 0; i < (28 - shift); i++)
D[x][i] = D[x - 1][i + shift];
k = 0;
for (int i = 28 - shift; i < 28; i++)
D[x][i] = backup[x - 1][k++];
}
for (int j = 0; j < 17; j++)
{
for (int i = 0; i < 28; i++)
CD[j][i] = C[j][i];
for (int i = 28; i < 56; i++)
CD[j][i] = D[j][i - 28];
}

```

```

for (int j = 1; j < 17; j++)
for (int i = 0; i < 56; i++)
key56to48(j, i, CD[j][i]);
}

int main(){
char plain_text[100];
printf("Enter the text :- ");
scanf("%[^\n]s",plain_text);
convert_Text_to_bits(plain_text);
key64to48(Original_key);
int _64bit_sets = bits_size/64;
printf("Decrypted output is\n");
for(int i=0;i<= _64bit_sets ;i++) {
Encrypt_each_64_bit (text_to_bits + 64*i);
}
printf("\n");
printf("%s",plain_text);
return 0;
}

```

### OUTPUT:

```
Enter the text :- hello
Decrypted output is
1010001011100111101010011110001110111000000001110100110010101100
hello

...Program finished with exit code 0
Press ENTER to exit console.[]
```

## EXPERIMENT-6: IMPLEMENTATION OF BLOCK CIPHER

**AIM:** To write a C program to implement block cipher principles of ECB and CBC

### DESCRIPTION:

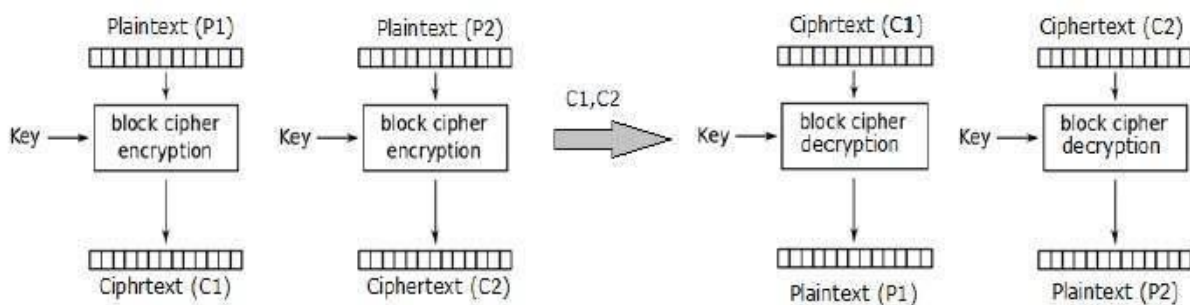
Electronic code book is the easiest block cipher mode of functioning. It is easier because of direct encryption of each block of input plaintext and output is in form of blocks of encrypted ciphertext. Generally, if a message is larger than  $b$  bits in size, it can be broken down into a bunch of blocks and the procedure is repeated.

#### Operation

- The user takes the first block of plaintext and encrypts it with the key to produce the first block of ciphertext.
- He then takes the second block of plaintext and follows the same process with same key and so on so forth.

The ECB mode is **deterministic**, that is, if plaintext block  $P_1, P_2, \dots, P_m$  are encrypted twice under the same key, the output ciphertext blocks will be the same.

In fact, for a given key technically we can create a codebook of ciphertexts for all possible plaintext blocks. Encryption would then entail only looking up for required plaintext and select the corresponding ciphertext. Thus, the operation is analogous to the assignment of code words in a codebook, and hence gets an official name – Electronic Codebook mode of operation (ECB).



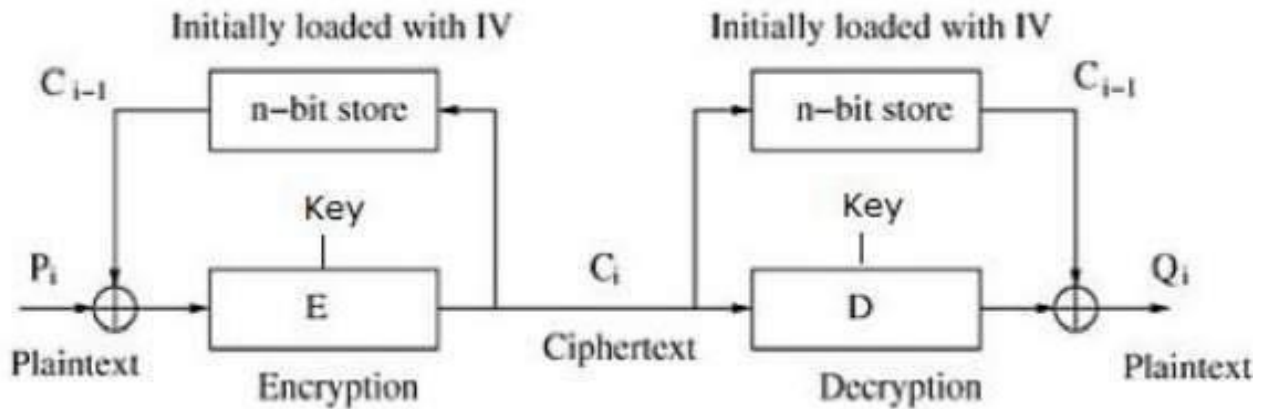
Cipher block chaining or CBC is an advancement made on ECB since ECB compromises some security requirements. In CBC, the previous cipher block is given as input to the next encryption algorithm after XOR with the original plaintext block. In a nutshell here, a cipher block is produced by encrypting an XOR output of the previous cipher block and present plaintext block.

#### Operation

The operation of CBC mode is depicted in the following illustration. The steps are as follows –

- Load the  $n$ -bit Initialization Vector (IV) in the top register.
- XOR the  $n$ -bit plaintext block with data value in top register.
- Encrypt the result of XOR operation with underlying block cipher with key  $K$ .

- Feed ciphertext block into top register and continue the operation till all plaintext blocks are processed.
- For decryption, IV data is XORed with first ciphertext block decrypted. The first ciphertext block is also fed into to register replacing IV for decrypting next ciphertext block.



### **PROGRAM:**

```
import java.util.*;

public class Main{

    private static byte[] P_key = {
        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
    };

    private static byte[] L_shift = {
        1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1
    };

    private static byte[] second_p = {
        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
};

private static byte[] IP = {
    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
};

// E selection table
private static byte[] sel_table = {
    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
};

private static byte[][] S = {
    {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, 13, 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}
};

private static byte[] P = {
    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
};

//final permutation
private static byte[] FP = {
    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
};

private static int[] C = new int[28];
private static int[] D = new int[28];
private static int[] L = new int[32];
private static int[] R = new int[32];
public static void main(String args[]){
    Scanner scan = new Scanner(System.in);
    System.out.println("");
    System.out.println("*****");
    System.out.println("***** DES *****");
    System.out.println("*****");
    System.out.println("");
    System.out.println("enter 64 bits plaintext: ");
    String input = scan.nextLine();
    StringTokenizer strToken = new StringTokenizer(input);
    int count = strToken.countTokens();
    System.out.println("count: " + count);
}

```

```

int[] plaintext = new int[count];
for(int i=0; i < count; i++){
    plaintext[i] = Integer.parseInt((String)strToken.nextElement());
}
System.out.print("{ ");
for(int j = 0; j < plaintext.length; j++){
    System.out.print("" + plaintext[j] + ",");
}
// end for
System.out.println("}");
int[] key = new int[count];
System.out.println("");
System.out.println("enter 64 bits key: ");
String key_in = scan.nextLine();
StringTokenizer strToken1 = new StringTokenizer(key_in);
int count1 = strToken1.countTokens();
System.out.println("count: " + count1);
for(int i=0; i < count1; i++){
    key[i] = Integer.parseInt((String)strToken1.nextElement());
}
System.out.print("{ ");
for(int j = 0; j < key.length; j++){
    System.out.print("" + key[j] + ",");
}
// end for
System.out.println("}");
System.out.println("Processing DES...");
System.out.println("...");
System.out.println("...");
System.out.println("");
int ciphertext[] = DES(plaintext,key);
System.out.println("");
System.out.println("*****");
System.out.println("***** ECB *****");
System.out.println("*****");

```

```

System.out.println("");
String s1 = "I LOVE SECURITY";
String s2 = "ABCDEFGH";
String IV = "ABCDEFGH";
int ecb[] = ECB(s1,s2);
System.out.println("");
System.out.println("*****");
System.out.println("***** CBC *****");
System.out.println("*****");
System.out.println("");
int cbc[] = CBC(s1,s2,IV);
System.out.println("");
}

//*****

//***** DES *****

//*****

private static int[] DES(int[] plaintext, int[] key){
    if(plaintext.length == 64 && key.length == 64)
    for(int i = 0; i < 28; i++){
        C[i] = key[P_key[i] - 1];
    }
    // System.out.println("Processing DES...1");
    for(int i = 28; i < 56; i++){
        D[i-28] = key[P_key[i] -1];
    }
    // System.out.println("Processing DES...2");
    int[] pre_plaintext = new int[plaintext.length];
    for(int i=0; i < plaintext.length; i++){
        pre_plaintext[i] = plaintext[IP[i] - 1];
    }
    // System.out.println("Processing DES...3");
    System.arraycopy(pre_plaintext, 0, L, 0, 32);
    System.arraycopy(pre_plaintext, 32, R, 0, 32);

```

```

// System.out.println("Processing DES...4");
for(int i = 0; i < 16; i++){
    // System.out.println("Processing DES...5");
    int[] expand = new int[48];
    for(int j = 0; j < 48; j++){
        expand[j] = R[sel_table[j] - 1];
    }
    // System.out.println("Processing DES...6");
    int[] K = new int[48];
    K = key_cal(key, i);
    int E_xor[] = xor(expand,K);
    //System.out.println("Processing DES...7");
    int cal[] = S_box(E_xor);
    // System.out.println("Processing DES...8");
    int[] new_R = new int[0];
    new_R = xor(L, cal);
    //System.out.println("Processing DES...9");
    L = R;
    R = new_R;
}
int plaintext_final[] = new int[64];
System.arraycopy(R,0,plaintext_final,0,R.length);
System.arraycopy(L,0,plaintext_final,32,L.length);
int ciphertext1[] = new int[64];
for(int i = 0; i < plaintext_final.length; i++){
    ciphertext1[i] = plaintext_final[FP[i] - 1];
}
//displayBits(ciphertext);
System.out.println("");
System.out.println("Ciphertext: ");
System.out.print(" { ");
for(int i = 0; i < ciphertext1.length; i++){
    System.out.print("'" + ciphertext1[i] + ",");
}

```

```

    }

    System.out.println("{}");

    System.out.println("");

    return ciphertext1;

}

else

{

    throw new ArrayIndexOutOfBoundsException("Array size wrong");

}

};//end DES

private static int[] key_cal (int key[], int iter){

    //C and D left shift for 16 rounds

    //CD[] combine from C0D0 ~ C15D15

    int CD[] = new int[56];

    int tem_C[] = new int[28];

    int tem_D[] = new int[28];

    int i = (int)L_shift[iter];

    tem_C = left_shift(C, i);

    tem_D = left_shift(D, i);

    //CD store the combine of temC and temD

    System.arraycopy(tem_C,0,CD,0,tem_C.length);

    System.arraycopy(tem_D,0,CD,28,tem_D.length);

    //use Second_p table to reduce to 48bits

    int tem_K[] = new int[48];

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

        tem_K[j] = CD[second_p[j] - 1];

    }

    C = tem_C;

    D = tem_D;

    return tem_K;

}//end key_cal

private static int[] S_box (int[] input){

    int[] output = new int[32];

```

```

for(int i = 0; i < 8; i++){
    int row[] = new int[2];
    int column[] = new int[4];
    // the first and the last of every 6 bits block are the Row locations
    row[0] = input[6*i];
    row[1] = input[(6*i) + 5];
    String s1 = row[0] + "" + row[1];
    // the 2nd to the 4th of every 6 bits block are the Column locations
    column[0] = input[(6*i)+1];
    column[1] = input[(6*i)+2];
    column[2] = input[(6*i)+3];
    column[3] = input[(6*i)+4];
    String s2 = column[0] + "" + column[1] + "" + column[2] + "" + column[3];
    int x = Integer.parseInt(s1, 2);
    int y = Integer.parseInt(s2, 2);
    int z = S[i][(x*16) + y];
    String b = Integer.toBinaryString(z);
    while(b.length() < 4){
        b = "0" + b;
    }
    for(int j = 0; j < 4; j++){
        output[(i*4) + j] = Integer.parseInt(b.charAt(j) + "");
    }
}
int[] result = new int[32];
for(int i = 0; i < P.length; i++){
    result[i] = output[P[i] - 1];
}
return result;
} //end S_box

private static int[] xor (int[] input1, int[] input2){
    int[] output = new int[input1.length];
    for(int i = 0; i < input1.length; i++){

```

```

        output[i] = (input1[i]+input2[i]) % 2;
    }
    return output;
} // end xor

private static int[] left_shift(int[] input, int n){
    int temp[] = new int[input.length];
    if(n == 1){
        temp[input.length - 1] = input[0];
        for(int i = 0; i < input.length - 1; i++){
            temp[i] = input[i+1];
        }
    }
    else
    {
        temp[input.length - 2] = input[0];
        temp[input.length - 1] = input[1];
        for(int i = 0; i < input.length - 2; i++){
            temp[i] = input[i+2];
        }
    }
    return temp;
} //end left Shift

/*****

/*****      ECB      *****/

/*****/

private static int[] ECB(String plaintext, String key){
    char[] plaintext_array = plaintext.toCharArray();
    char[] key_array = key.toCharArray();
    int[] num = new int[plaintext_array.length];
    int[] num2 = new int[key_array.length];
    //convert plaintext to int[] num
    for(int i = 0; i < plaintext_array.length; i++){
        num[i] = plaintext_array[i];
    }

```

```

    }
    //convert key to int[] num2
    for(int i = 0; i < key_array.length; i++){
        num2[i] = key_array[i];
    }
    String s = "";
    String temps;
    String s_key = "";
    String temps_key;
    for(int i = 0; i < num.length; i++){
        int tem = num[i];
        temps = Integer.toBinaryString(tem);
        while(temps.length() < 8){
            temps = "0" + temps;
            if(temps.length() < 8)
                temps = "0" + temps;
        }
        s = s + temps;
    }
    System.out.println("");
    System.out.println("Plaintext convert to binary: " + s);
    //expand each character in key to 8 bits ASCII code
    for(int i = 0; i < num2.length; i++){
        int tem = num2[i];
        temps_key = Integer.toBinaryString(tem);
        while(temps_key.length() < 8){
            temps_key = "0" + temps_key;
            if(temps_key.length() < 8)
                temps_key = "0" + temps_key;
        }
        s_key = s_key + temps_key;
    }
    if( s.length() < 64){

```



```

        throw new ArrayIndexOutOfBoundsException("Array size wrong");
    }
    else
    {
        int numofblock = ((s.length() - (s.length() % 64)) / 64);
        int n = 64 - (s.length() - 64 * numofblock);
        for(int i = 0; i < n; i++){
            s = s + "0";
        }
        int[] text_result = new int[s.length()];
        for(int i = 0; i < s.length(); i++){
            text_result[i] = Integer.parseInt(String.valueOf(s.charAt(i)));
        }
        int[] key_result = new int[s_key.length()];
        for(int i = 0; i < s_key.length(); i++){
            key_result[i] = Integer.parseInt(String.valueOf(s_key.charAt(i)));
        }
        System.out.println("");
        System.out.println("The original plaintext: ");
        System.out.println("{");
        for(int i = 0; i < text_result.length; i++){
            System.out.print("" + text_result[i] + ",");
        }
        System.out.println("}" + " ---- Count : " + text_result.length);
        int[] n1 = new int[64];
        int[] n2 = new int[64];
        for(int i = 0 ; i < 64; i++){
            n1[i] = text_result[i];
        }
        for(int i = 0; i < 64; i++){
            n2[i] = text_result[i+64];
        }
        System.out.println("");
    }

```

```

System.out.println(" 1st block cipher: ");
int[] ciphertext1 = DES(n1,key_result);
System.out.println("");
System.out.println(" 2nd block cipher: ");
int[] ciphertext2 = DES(n2,key_result);
int[] finaltext = new int[ciphertext1.length + ciphertext2.length];
for(int i = 0; i < ciphertext1.length; i++){
    finaltext[i] = ciphertext1[i];
}
for(int i = 0; i < ciphertext2.length; i++){
    finaltext[i + 64] = ciphertext2[i];
}
int numofdex = finaltext.length / 8;
String binarystring = "";
int[] final_output = new int[numofdex];
for(int i = 0; i < numofdex; i++){
    for(int j = i*8; j <= i*8 +7; j++){
        binarystring = binarystring + finaltext[j];
    }
    int numtemp;
    numtemp = Integer.parseInt(binarystring,2);
    final_output[i] = numtemp;
    binarystring = "";
}
System.out.println("");
System.out.println("Processing ECB.....");
System.out.println(".....");
System.out.println(".....");
System.out.println("Ciphertext of ECB: ");
System.out.print("{");
for(int i = 0; i < final_output.length; i++){
    System.out.print("" + final_output[i] + ",");
}

```

```

        System.out.println("}");

        System.out.println("");

        return final_output;

    }

}

//*****

//*****      CBC      *****

//*****

private static int[] CBC(String plaintext, String key, String IV){

    char[] plaintext_array = plaintext.toCharArray();
    char[] key_array = key.toCharArray();
    char[] iv_array = IV.toCharArray();
    int[] num = new int[plaintext_array.length];
    int[] num2 = new int[key_array.length];
    int[] num3 = new int[iv_array.length];
    for(int i = 0; i < plaintext_array.length; i++){
        num[i] = plaintext_array[i];
    }
    for(int i = 0; i < key_array.length; i++){
        num2[i] = key_array[i];
    }
    for(int i = 0; i < iv_array.length; i++){
        num3[i] = iv_array[i];
    }
    String stext = "";
    String temps;
    String skey = "";
    String temps_key;
    String siv = "";
    String temps_iv;

    //expand each character in plaintext to 8 bits ASCII code
    for(int i = 0; i < num.length; i++){
        int tem = num[i];

```

```

    temps = Integer.toBinaryString(tem);
    while(temps.length() < 8){
        temps = "0" + temps;
        if(temps.length() < 8)
            temps = "0" + temps;
    }
    stext = stext + temps;
}
System.out.println("Plaintext convert to binary: " + stext);
System.out.println("");
for(int i = 0; i < num2.length; i++){
    int tem = num2[i];
    temps_key = Integer.toBinaryString(tem);
    while(temps_key.length() < 8){
        temps_key = "0" + temps_key;
        if(temps_key.length() < 8)
            temps_key = "0" + temps_key;
    }
    skey = skey + temps_key;
}
System.out.println("key convert to binary: " + skey);
System.out.println("");
for(int i = 0; i < num3.length; i++){
    int tem = num3[i];
    temps_iv = Integer.toBinaryString(tem);
    while(temps_iv.length() < 8){
        temps_iv = "0" + temps_iv;
        if(temps_iv.length() < 8)
            temps_iv = "0" + temps_iv;
    }
    siv = siv + temps_iv;
}
System.out.println("IV convert to binary: " + siv);

```

```

System.out.println("");
if( stext.length() < 64){
    throw new ArrayIndexOutOfBoundsException("Array size wrong");
}
else
{
    int numofblock = ((stext.length() - (stext.length() % 64)) / 64);
    int n = 64 - (stext.length() - 64 * numofblock);
    for(int i = 0; i < n; i++){
        stext = stext + "0";
    }
    int[] text_result = new int[stext.length()];

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

        text_result[i] = Integer.parseInt(String.valueOf(stext.charAt(i)));

    }
    int[] key_result = new int[skey.length()];
    for(int i = 0; i < skey.length(); i++){

        key_result[i] = Integer.parseInt(String.valueOf(skey.charAt(i)));
    }
    int[] iv_result = new int[siv.length()];
    for(int i = 0; i < siv.length(); i++){

        iv_result[i] = Integer.parseInt(String.valueOf(siv.charAt(i)));

    }
    int[] n1 = new int[64];
    int[] n2 = new int[64];
    for(int i = 0 ; i < 64; i++){
        n1[i] = text_result[i];

```

```

    }
    for(int i = 0; i < 64; i++){
        n2[i] = text_result[i+64];
    }
    System.out.println("");
    System.out.println(" 1st block cipher: ");
    int[] first = xor(n1, iv_result);
    int[] c1 = DES(first,key_result);
    System.out.println("");
    System.out.println(" 2nd block cipher: ");
    int[] second = xor(n2,c1);
    int[] c2 = DES(second,key_result);
    System.out.println("");
    int[] finaltext = new int[c1.length + c2.length];
    for(int i = 0; i < c1.length; i++){
        finaltext[i] = c1[i];
    }
    for(int i = 0; i < c2.length; i++){
        finaltext[i + 64] = c2[i];
    }

    int numofdex = finaltext.length / 8;
    String binarystring = "";
    int[] final_output = new int[numofdex];

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

        for(int j = i*8; j <= i*8 +7; j++){

            binarystring = binarystring + finaltext[j];
        }
        int numtemp;
        numtemp = Integer.parseInt(binarystring,2);

```

```
        final_output[i] = numtemp;
        binarystring = "";
    }
    System.out.println("Processing CBC....");
    System.out.println(".....");
    System.out.println("Ciphertext of CBC: ");
    System.out.print("{");
    for(int i = 0; i < final_output.length; i++){
        System.out.print("'" + final_output[i] + ",");
    }
    System.out.print("}");
    System.out.println("");
    return final_output;
}
};
}
```

[illegible]



## **EXPERIMENT-7: IMPLEMENTATION OF RSA ALGORITHM**

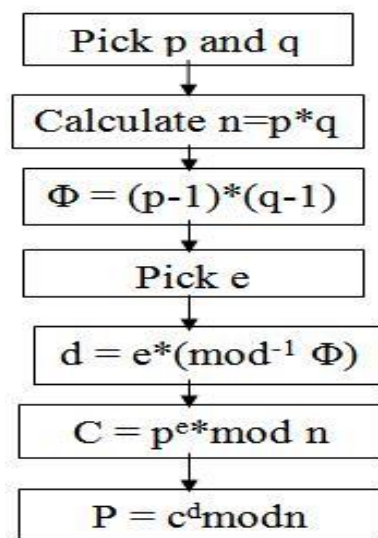
**AIM:** To write a C program to implement the RSA Algorithm for encryption and decryption.

### **DESCRIPTION:**

RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of them can be given to everyone. A basic principle behind RSA is the observation that it is practical to find three very large positive integers  $e$ ,  $d$  and  $n$  such that with modular exponentiation for all integer  $m$ :  $(m^e)^d = m \pmod{n}$

The public key is represented by the integers  $n$  and  $e$ ; and, the private key, by the integer  $d$ .  $m$  represents the message. RSA involves a public key and a private key. The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key.

### **EXAMPLE:**



### **PROGRAM:**

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

long int p,q,n,t,flag,e[100],d[100],temp[100],j,m[100],en[100],i;
char msg[100];

int prime(long int);
```

```

void ce();
long int cd(long int);
void encrypt();
void decrypt();
int main()
{
printf("\nEnter FIRST PRIME NUMBER\n");
scanf("%ld",&p);
flag=prime(p);
if(flag==0)
{
printf("\nWRONG INPUT\n");
exit(1);
}
printf("\nEnter ANOTHER PRIME NUMBER\n");
scanf("%ld",&q);
flag=prime(q);
if(flag==0||p==q)
{
printf("\nWRONG INPUT\n");
exit(1);
}
printf("\nEnter MESSAGE\n");
fflush(stdin);
scanf("%s",msg);
for(i=0;msg[i]!=NULL;i++)
m[i]=msg[i];
n=p*q;
t=(p-1)*(q-1);
ce();
printf("\nPOSSIBLE VALUES OF e AND d ARE\n");
for(i=0;i<t-1;i++)
printf("\n%ld\t%ld",e[i],d[i]);
encrypt();

```

```

decrypt();
return 0;
}
int prime(long int pr)
{
int i;
j=sqrt(pr);
for(i=2;i<=j;i++)
{
if(pr%i==0)
return 0;
}
return 1;
}
void ce()
{
int k;
k=0;
for(i=2;i<t;i++)
{
if(t%i==0)
continue;
flag=prime(i);
if(flag==1 && i!=p && i!=q)
{
e[k]=i; flag=cd(e[k]);
if(flag>0)
{
d[k]=flag;
k++;
}
if(k==99)
break;
}
}
}

```

```

    }
}
long int cd(long int x)
{
    long int k=1;
    while(1)
    {
        k=k+t;
        if(k%x==0)
            return(k/x);
    }
}
void encrypt()
{
    long int pt,ct,key=e[0],k,len;
    i=0;
    len=strlen(msg);
    while(i!=len)
    {
        pt=m[i];
        pt=pt-96;
        k=1;
        for(j=0;j<key;j++)
        {
            k=k*pt;
            k=k%n;
        }
        temp[i]=k;
        ct=k+96;
        en[i]=ct;
        i++;
    }
    en[i]=-1;
    printf("\nTHE ENCRYPTED MESSAGE IS\n");

```

```

for(i=0;en[i]!=-1;i++)
printf("%c",en[i]);
}
void decrypt()
{
long int pt,ct,key=d[0],k;
i=0;
while(en[i]!=-1)
{
ct=temp[i];
k=1;
for(j=0;j<key;j++)
{
k=k*ct;
k=k%n;
}
pt=k+96;
m[i]=pt;
i++;
}
m[i]=-1;
printf("\nTHE DECRYPTED MESSAGE IS\n");
for(i=0;m[i]!=-1;i++)
printf("%c",m[i]);
}

```

## OUTPUT

```
ENTER FIRST PRIME NUMBER
3

ENTER ANOTHER PRIME NUMBER
5

ENTER MESSAGE
cryptography

POSSIBLE VALUES OF e AND d ARE
7      7
THE ENCRYPTED MESSAGE IS
lljae`mlaabj
THE DECRYPTED MESSAGE IS
ccjae`gcaahj

...Program finished with exit code 0
Press ENTER to exit console.
```

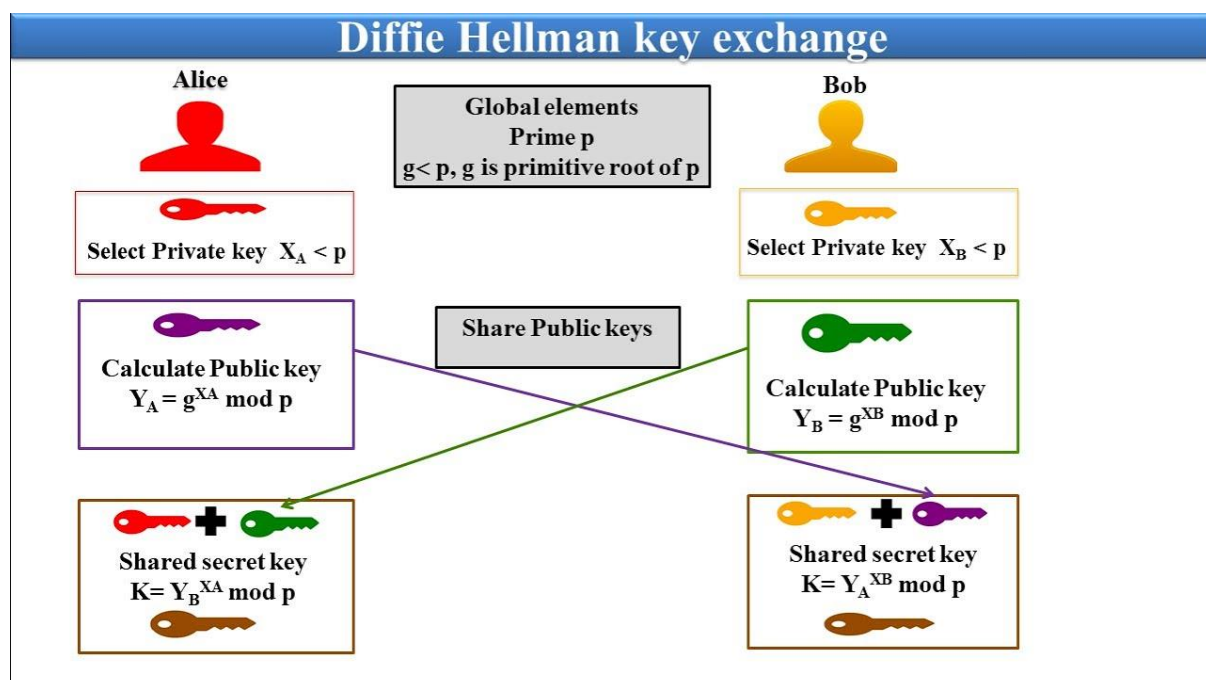
## EXPERIMENT-8: IMPLEMENTATION OF DIFFIE-HELLMAN

**AIM:** To implement the Diffie-Hellman Key Exchange mechanism using HTML and JavaScript

### **DESCRIPTION:**

Diffie–Hellman Key Exchange establishes a shared secret between two parties that can be used for secret communication for exchanging data over a public network. It is primarily used as a method of exchanging cryptography keys for use in symmetric encryption algorithms like AES. The algorithm in itself is very simple. The process begins by having the two parties, Alice and Bob. Let's assume that Alice wants to establish a shared secret with Bob.

### EXAMPLE:

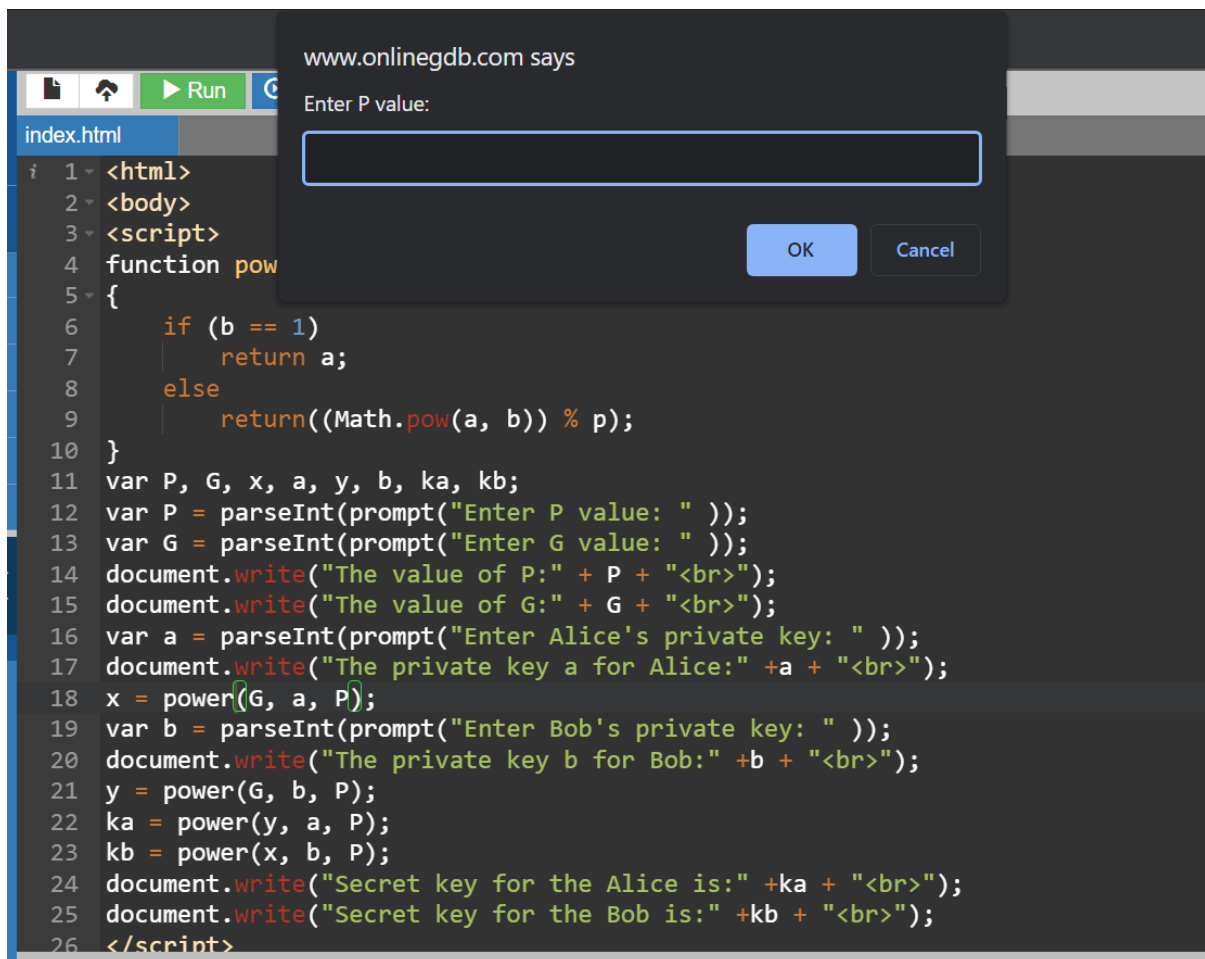


### **PROGRAM:**

```
<html>
<body>
<script>
function power(a, b, p)
{
    if (b == 1)
        return a;
    else
        return((Math.pow(a, b)) % p);
}
var P, G, x, a, y, b, ka, kb;
var P = parseInt(prompt("Enter P value: " ));
var G = parseInt(prompt("Enter G value: " ));
document.write("The value of P:" + P + "<br>");
document.write("The value of G:" + G + "<br>");
var a = parseInt(prompt("Enter Alice's private key: " ));
document.write("The private key a for Alice:" +a + "<br>");
x = power(G, a, P);
var b = parseInt(prompt("Enter Bob's private key: " ));
document.write("The private key b for Bob:" +b + "<br>");
y = power(G, b, P);
ka = power(y, a, P);
kb = power(x, b, P);
document.write("Secret key for the Alice is:" +ka + "<br>");
document.write("Secret key for the Bob is:" +kb + "<br>");
</script>
</body>
</html>
```



## OUTPUT



The screenshot shows a web browser window with a dark theme. A prompt dialog box is open, titled "www.onlinegdb.com says", with the text "Enter P value:" and an input field. Below the input field are "OK" and "Cancel" buttons. In the background, a code editor shows the following JavaScript code:

```
index.html
1 <html>
2 <body>
3 <script>
4 function pow
5 {
6     if (b == 1)
7         return a;
8     else
9         return((Math.pow(a, b)) % p);
10 }
11 var P, G, x, a, y, b, ka, kb;
12 var P = parseInt(prompt("Enter P value: " ));
13 var G = parseInt(prompt("Enter G value: " ));
14 document.write("The value of P:" + P + "<br>");
15 document.write("The value of G:" + G + "<br>");
16 var a = parseInt(prompt("Enter Alice's private key: " ));
17 document.write("The private key a for Alice:" +a + "<br>");
18 x = power(G, a, P);
19 var b = parseInt(prompt("Enter Bob's private key: " ));
20 document.write("The private key b for Bob:" +b + "<br>");
21 y = power(G, b, P);
22 ka = power(y, a, P);
23 kb = power(x, b, P);
24 document.write("Secret key for the Alice is:" +ka + "<br>");
25 document.write("Secret key for the Bob is:" +kb + "<br>");
26 </script>
```

**The value of P:23**  
**The value of G:9**  
**The private key a for Alice:4**  
**The private key b for Bob:3**  
**Secret key for the Alice is:9**  
**Secret key for the Bob is:9**

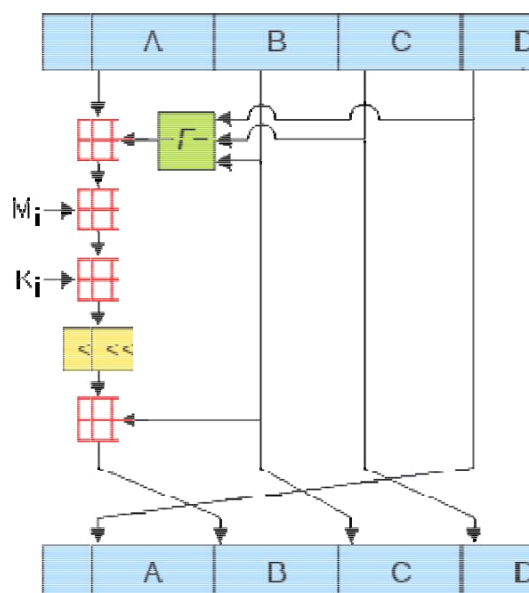
## **EXPERIMENT-9: IMPLEMENTATION OF MD5**

**AIM:** To write a java program to calculate the message digest of a text using the MD5 hashing technique.

### **DESCRIPTION:**

MD5 processes a variable-length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit blocks. The message is **padding** so that its length is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits less than a multiple of 512. The remaining bits are filled up with 64 bits representing the length of the original message, modulo  $2^{64}$ . The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted A, B, C, and D. These are initialized to certain fixed constants. The main algorithm then uses each 512-bit message block in turn to modify the state.

### **EXAMPLE:**



**PROGRAM:**

```
import java.security.*;
import java.util.*;
public class Main {
public static void main(String[] a) {
    try {
        MessageDigest md = MessageDigest.getInstance("MD5");
        System.out.println("Message digest object info: ");
        System.out.println(" Algorithm = " +md.getAlgorithm());
        System.out.println(" Provider = " +md.getProvider());
        System.out.println(" ToString = " +md.toString());
        String input = "";
        md.update(input.getBytes());
        byte[] output = md.digest();
        System.out.println();
        System.out.println("MD5(\""+input+"\") = " +bytesToHex(output));
        input = "abc";
        md.update(input.getBytes());
        output = md.digest();
        System.out.println();
        System.out.println("MD5(\""+input+"\") = " +bytesToHex(output));
        input = "abcdefghijklmnopqrstuvwxyz";
        md.update(input.getBytes());
        output = md.digest();
        System.out.println();
        System.out.println("MD5(\""+input+"\") = " +bytesToHex(output));
        System.out.println("");
    }
    catch (Exception e) {
        System.out.println("Exception: " +e); }
}
```

```
}  
  
public static String bytesToHex(byte[] b) {  
    char hexDigit[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};  
    StringBuffer buf = new StringBuffer();  
    for (int j=0; j<b.length; j++) {  
        buf.append(hexDigit[(b[j] >> 4) & 0x0f]);  
        buf.append(hexDigit[b[j] & 0x0f]);  
    }  
    return buf.toString();  
}  
}
```

## OUTPUT:

```
Message digest object info:
```

```
Algorithm = MD5
```

```
Provider = SUN version 11
```

```
ToString = MD5 Message Digest from SUN, <initialized>
```

```
MD5("") = D41D8CD98F00B204E9800998ECF8427E
```

```
MD5("abc") = 900150983CD24FB0D6963F7D28E17F72
```

```
MD5("abcdefghijklmnopqrstuvwxyz") = C3FCD3D76192E4007DFB496CCA67E13B
```

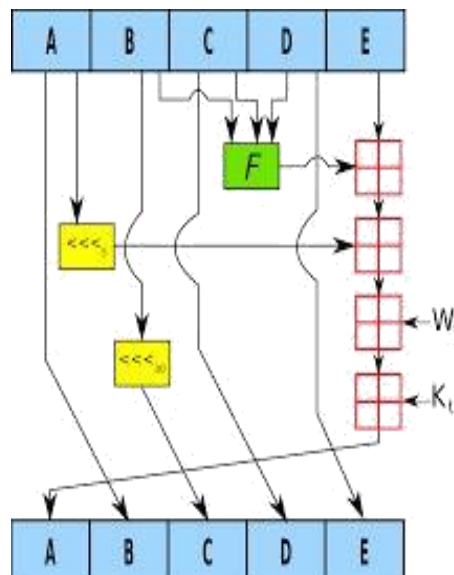
## **EXPERIMENT-10: IMPLEMENTATION OF SHA-1**

**AIM-**To calculate the message digest of a text using the SHA-1 algorithm in Java Language

### **DESCRIPTION**

In cryptography, SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function. SHA-1 produces a 160-bit hash value known as a message digest. The way this algorithm works is that for a message of size  $< 264$  bits it computes a 160-bit condensed output called a message digest. The SHA-1 algorithm is designed so that it is practically infeasible to find two input messages that hash to the same output message. A hash function such as SHA-1 is used to calculate an alphanumeric string that serves as the cryptographic representation of a file or a piece of data. This is called a digest and can serve as a digital signature. It is supposed to be unique and non-reversible.

### **EXAMPLE:**



**PROGRAM:**

```
import java.security.*;

import java.util.*;

public class Main{

    public static void main(String[] a) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA1");
            System.out.println("Message digest object info: ");
            System.out.println(" Algorithm = " +md.getAlgorithm());
            System.out.println(" Provider = " +md.getProvider());
            System.out.println(" ToString = " +md.toString());
            String input = "";
            md.update(input.getBytes());
            byte[] output = md.digest();
            System.out.println();
            System.out.println("SHA1(\""+input+"") = " +bytesToHex(output));
            input = "abc";
            md.update(input.getBytes());
            output = md.digest();
            System.out.println();
            System.out.println("SHA1(\""+input+"") = " +bytesToHex(output));
            input = "abcdefghijklmnopqrstuvwxyz";
            md.update(input.getBytes());
            output = md.digest();
            System.out.println();
            System.out.println("SHA1(\""+input+"") = " +bytesToHex(output));
            System.out.println(""); }
        catch (Exception e) {
            System.out.println("Exception: " +e);
```

```
    }  
}  
  
public static String bytesToHex(byte[] b) {  
    char hexDigit[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};  
    StringBuffer buf = new StringBuffer();  
    for (int j=0; j<b.length; j++) {  
        buf.append(hexDigit[(b[j] >> 4) & 0x0f]);  
        buf.append(hexDigit[b[j] & 0x0f]); }  
    return buf.toString();  
}  
}
```



## OUTPUT

```
Message digest object info:
Algorithm = SHA1
Provider = SUN version 11
ToString = SHA1 Message Digest from SUN, <initialized>

SHA1("") = DA39A3EE5E6B4B0D3255BFEF95601890AFD80709

SHA1("abc") = A9993E364706816ABA3E25717850C26C9CD0D89D

SHA1("abcdefghijklmnopqrstuvwxyz") = 32D10C7B8CF96570CA04CE37F2A19D84240D3A89

...Program finished with exit code 0
Press ENTER to exit console.█
```

## **EXPERIMENT-11: STUDY OF INTRUSION DETECTION SYSTEM**

**AIM:** To study the Intrusion Detection System(snort IDS)

### **DESCRIPTION:**

An Intrusion Detection System (IDS) is a system that monitors network traffic for suspicious activity and issues alerts when such activity is discovered. It is a software application that scans a network or a system for the harmful activity or policy breaching. Any malicious venture or violation is normally reported either to an administrator or collected centrally using a security information and event management (SIEM) system. A SIEM system integrates outputs from multiple sources and uses alarm filtering techniques to differentiate malicious activity from false alarms.

Although intrusion detection systems monitor networks for potentially malicious activity, they are also disposed to false alarms. Hence, organizations need to fine-tune their IDS products when they first install them. It means properly setting up the intrusion detection systems to recognize what normal traffic on the network looks like as compared to malicious activity. Intrusion prevention systems also monitor network packets inbound the system to check the malicious activities involved in it and at once send the warning notifications.

SNORT is a network based intrusion detection system which is written in C programming language. It was developed in 1998 by Martin Roesch. Now it is developed by Cisco. It is free open-source software. It can also be used as a packet sniffer to monitor the system in real time. The network admin can use it to watch all the incoming packets and find the ones which are dangerous to the system. It is based on library packet capture tool. The rules are fairly easy to create and implement and it can be deployed in any kind of operating system and any kind of network environment. The main reason of the popularity of this IDS over others is that it is a free-to-use software and also open source because of which any user can be able to use it as the way he wants.

In Windows:

- **Step-1:** Download SNORT installer from [https://www.snort.org/downloads/snort/Snort\\_2\\_9\\_15\\_Installer.exe](https://www.snort.org/downloads/snort/Snort_2_9_15_Installer.exe)
- **Step-2:** Execute the Snort\_2\_9\_15\_Installer.exe