PRACTICAL-6

AIM: Write a C program to implement simple DES.

INTRODUCTION:

- The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).
- DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DES has an effective key length of 56 bits, since 8 of the 64 bits of the key are not used by the encryption algorithm (function as check bits only).
- Since DES is based on the Feistel Cipher, all that is required to specify DES is :
  - Round function
  - Key schedule
  - Any additional processing − Initial and final permutation
- The DES satisfies both the desired properties of block cipher. These two properties make cipher very strong.
  - **Avalanche effect** − A small change in plaintext results in the very great change in the ciphertext.
  - **Completeness** − Each bit of ciphertext depends on many bits of plaintext.
- During the last few years, cryptanalysis have found some weaknesses in DES when key selected are weak keys. These keys shall be avoided.
- DES has proved to be a very well designed block cipher. There have been no significant cryptanalytic attacks on DES other than exhaustive key search.

CODE:

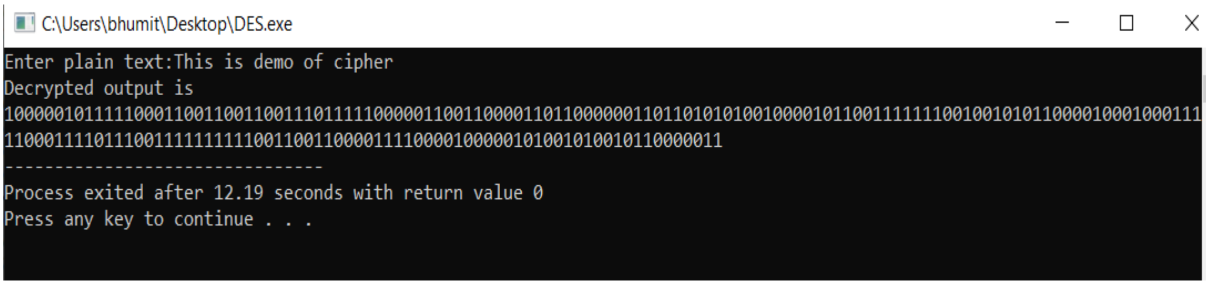
```
#include <stdio.h>

 int Original_key [64] = { // you can change key if required

            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 Permutated_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 Permutated_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 Iintial_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 shifts_for_each_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 EXPtext[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)
                    EXPtext[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(EXPtext[i], _48bit_key[Round][i]);
                else
                    XORtext[i] = XOR(EXPtext[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[ Iintial_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++)
            { // processing through all 16 rounds
                cipher(k, 0);
                for (int i = 0; i < 32; i++)
                    Left32[k][i] = Right32[k - 1][i]; // right part comes as it is to next
round left part}
            for (int i = 0; i < 64; i++)
            { // 32bit swap as well as Final Inverse Permutation
                if (i < 32)
                    chiper_text[i] = Right32[16][i];
                else
                    chiper_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 = shifts_for_each_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 plain text:");gets(plain_text);
        convert_Text_to_bits(plain_text);
        key64to48(Original_key); // it creates all keys for all rounds
        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);}
        return 0;  }
```

OUTPUT:



```
Enter plain text:This is demo of cipher
Decrypted output is
1000001011111000110011001100111011111100000110011000011011000000110110101010010000101100111111100100101011000010001000111
110001111011100111111111100110011000011110000100000101001010010110000011
--------------------------------
Process exited after 12.19 seconds with return value 0
Press any key to continue . . .
```