Date - 18-12-2020

SECURITY LAB-:IT-8761 UNIVERSITY LABORATORY PRACTICALS

Name: Jay Vishaal J
Register-No: 312217104065
Session: FN
Course: BE Computer Science and Engineering
College Code: 3122
College Name: SSN College of Engineering
Semester: VII

QUESTION:

Develop a java program to implement the DES decryption. Keys in numeric/ hexadecimal/ binary will be given. Input will be an encrypted message in hexadecimal and the output should be an intelligible English statement.

18-12-20

IT8761 Security Lab

AU Semester Practicals

Name : J Jay Vishaal

Reg no : 312217104065

Aim:

To develop a java program to implement DES decryption. keys in numeric / hexadecimal / binary will be given. Input will be an encrypted message in hexadecimal & the output should be an intelligible English statement.

Algorithm:

Key-generation:

Step 1: Initialize the permutation tables, left shift schedules.

Step 2: Read the 64 bit key.

Step 3: 64- bit key goes through a permutation (PC-1) which gives the output in 56 bits.

Step 4: These 56 bits are divided into 2 halves.

Step 5. Each half is rotated left by 1 (or) 2 bits depending on the left shift schedule initialized.

Step 6: Both halves go through PC-2 permutation which selects 24 from each left & right halves resulting a 48 bit round key.

Decryption:

Step 1: Use the input ie, the cipher text

Step 2: Permute block of 64 bits using Initial permutation.

Step 3: 64 bits are divided into 2 halves, left and right of 32 bits each.

Step 4: Right half goes through a function F which is the Feistel function.

Step 5: left half is XORed with output from F function.

Step 6: In last round, apply the inverse permutation on both the halves.

Step 7: The keys ki are used in reverse order i.e., k16 will be the key in the first round of decryption and so on.

Step 8: Display the plain text.

Fiestel Function F:

Step 1: 32 bits are expanded to 48 based on expansion table

Step 2: Round key is combined with 48 bits from the previous step by XOR operation.

Step 3: Substitution i.e., 8 s boxes are used for producing the 16 bit blocks

Step 4: Permutation is done based on a fixed table.

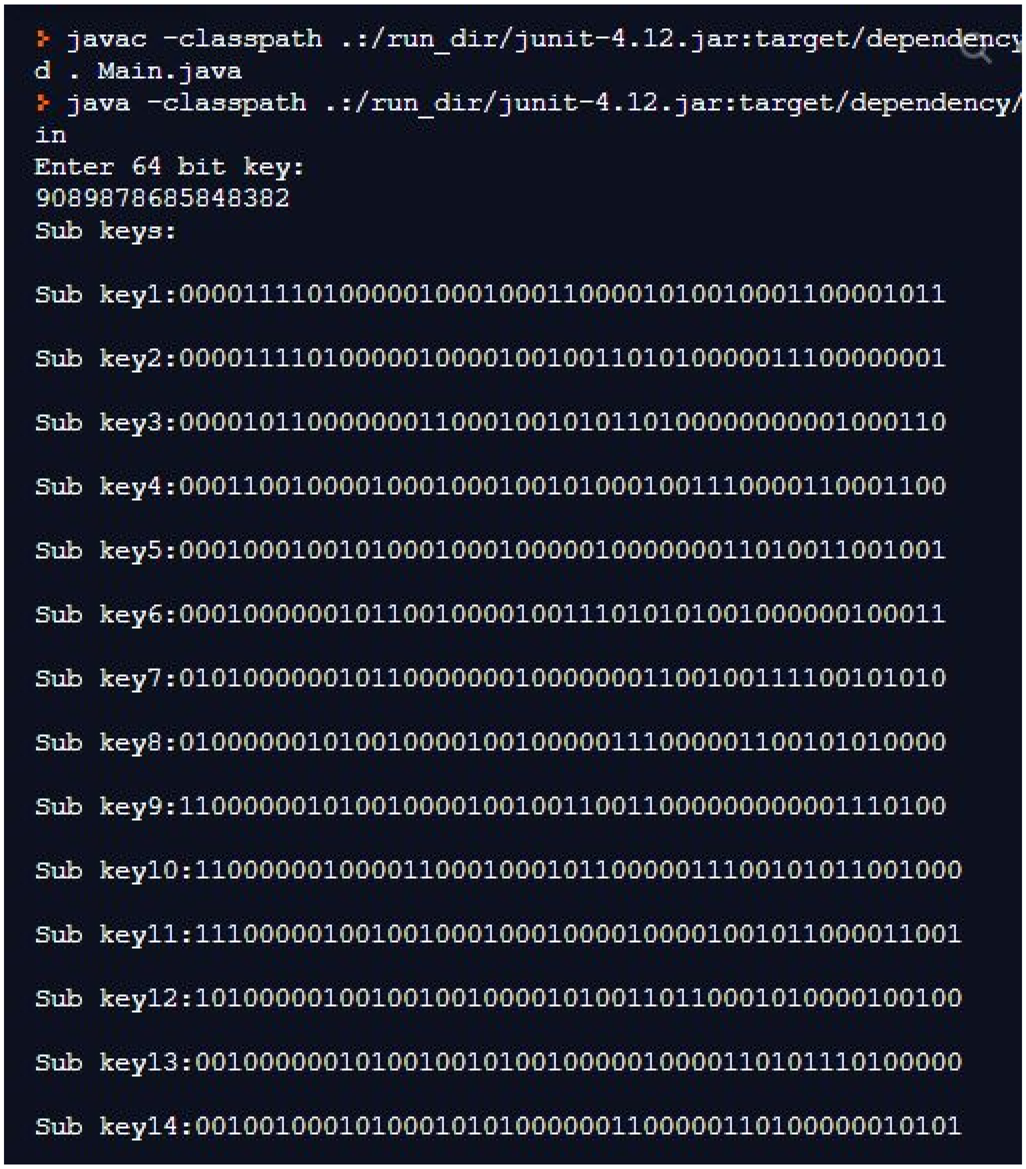
CODE :

```java
import java.lang.StringBuffer;
import java.math.*;
import java.util.*;

class Main {
   static int[] pc1 = { 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 };
   static int[] pc2 = { 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 };
   static int[] 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 };
   static int[] ip_inv = { 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 };
   static int[] expansion = { 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 };
   static int[] permutation_32 = { 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 };
   static int[][] s1 = { { 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 } };
   static int[][] s2 = { { 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 } };
    static int[][] s3 = { { 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 } };
    static int[][] s4 = { { 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 } };
    static int[][] s5 = { { 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 } };
    static int[][] s6 = { { 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 } };
    static int[][] s7 = { { 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 } };
    static int[][] s8 = { { 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 = { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };
    static String[] subkeys = new String[16];

    public static String leftShift(String key, int amount) {
        for (int i = 0; i < amount; i++) {
            char first = key.charAt(0);
            key = key.substring(1, key.length()) + first;
        }
        return key;
    }

    public static String generatePC1(String key) {
        char[] array = new char[pc1.length];
        int c = 0;
        for (int i : pc1) {
            array[c] = key.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);
    }

    public static String generatePC2(String key) {
        char[] array = new char[pc2.length];
        int c = 0;
        for (int i : pc2) {
            array[c] = key.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);
    }

    public static void keygen(String key) {
        String bin = new BigInteger(key, 16).toString(2);
        String left, right;
        while (bin.length() != 64) {
            bin = "0" + bin;
        }
        key = generatePC1(bin);
        left = key.substring(0, key.length() / 2);
        right = key.substring(key.length() / 2, key.length());
        for (int i = 1; i < 17; i++) {
            if (i == 1 || i == 2 || i == 9 || i == 16) {
                left = leftShift(left, 1);
                right = leftShift(right, 1);
            } else {
                left = leftShift(left, 2);
                right = leftShift(right, 2);
```

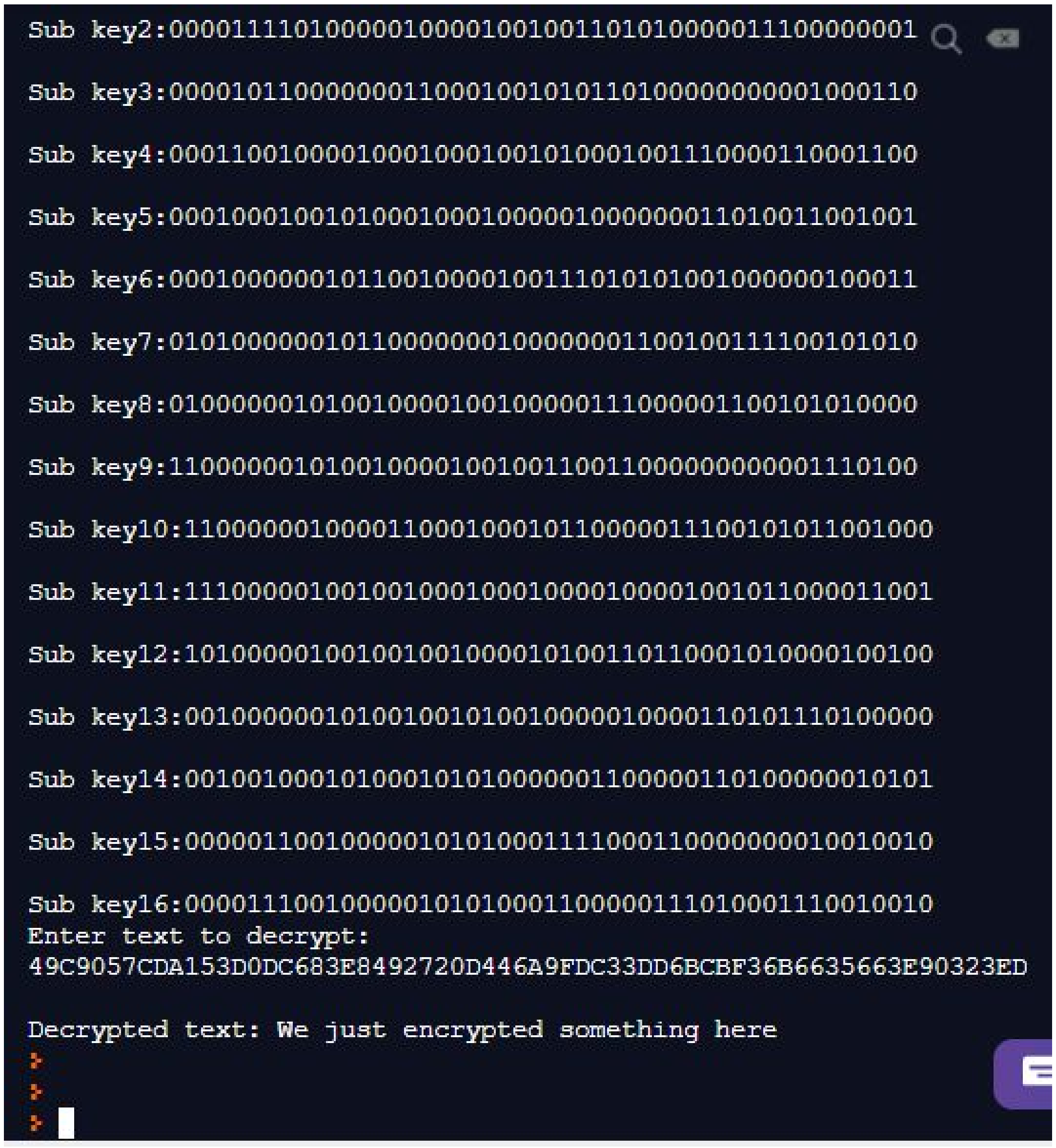
```java
            }
            key = "" + left + right;
            subkeys[i - 1] = generatePC2(key);
        }
    }

    public static String toHex(String input) {
        String hex = "";
        for (int i = 0; i < input.length(); i++) {
            hex += Integer.toHexString(input.charAt(i));
        }
        return hex;
    }

    public static String getIP(String input) {
        char[] array = new char[ip.length];
        int c = 0;
        for (int i : ip) {
            array[c] = input.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);
    }

    public static String expansion(String input) {
        char[] array = new char[expansion.length];
        int c = 0;
        for (int i : expansion) {
            array[c] = input.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);
    }

    public static String xorStrings(String s1, String s2) {
        String output = "";
        for (int i = 0; i < s1.length(); i++) {
            if (s1.charAt(i) == s2.charAt(i)) {
                output += "0";
            } else {
                output += "1";
            }
        }
        return output;
    }

    public static String applySbox(String input, int number) {
        String row, col, output = "";
        int r, c;
        row = "" + input.charAt(0) + input.charAt(input.length() - 1);
        col = input.substring(1, 5);
        r = Integer.parseInt(row, 2);
        c = Integer.parseInt(col, 2);
        switch (number) {
            case 0:
                output = Integer.toBinaryString(s1[r][c]);
                break;
            case 1:
                output = Integer.toBinaryString(s2[r][c]);
                break;
            case 2:
                output = Integer.toBinaryString(s3[r][c]);
                break;
            case 3:
                output = Integer.toBinaryString(s4[r][c]);
                break;
            case 4:
                output = Integer.toBinaryString(s5[r][c]);
                break;
            case 5:
                output = Integer.toBinaryString(s6[r][c]);
                break;
            case 6:
                output = Integer.toBinaryString(s7[r][c]);
                break;
            case 7:
                output = Integer.toBinaryString(s8[r][c]);
                break;
        }
        while (output.length() < 4) {
```

```java
            output = "0" + output;
        }
        return output;
    }

    public static String applySboxes(String input) {
        String output = "";
        for (int i = 0; i < 8; i++) {
            output += applySbox(input.substring(i * 6, i * 6 + 6), i);
        }
        return output;
    }

    public static String getPermutation32(String input) {
        char[] array = new char[permutation_32.length];
        int c = 0;
        for (int i : permutation_32) {
            array[c] = input.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);
    }

    public static String getIPinverse(String input) {
        char[] array = new char[ip_inv.length];
        int c = 0;
        for (int i : ip_inv) {
            array[c] = input.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);
    }

    public static String feistalRound(String right, String key) {
        right = expansion(right);
        String output = xorStrings(right, key);
        output = applySboxes(output);
        output = getPermutation32(output);
        return output;
    }
    public static String Decrypt(String text) {
        String output = "", bin = "";
        String t = "" + text;
        int iter = t.length() / 16;
        if (t.length() % 16 != 0) {
            iter++;
        }
        for (int z = 0; z < iter; z++) {
            if (z + 1 < iter) {
                bin = new BigInteger(t.substring(z * 16, z * 16 + 16),
16).toString(2);
            } else {
                bin = new BigInteger(t.substring(z * 16, t.length()),
16).toString(2);
            }
            while (bin.length() < 64) {
                bin = "0" + bin;
            }
            bin = getIP(bin);
            String temp, left, right;
            left = bin.substring(0, bin.length() / 2);
            right = bin.substring(bin.length() / 2, bin.length());
            for (int i = 0; i < 16; i++) {
                temp = left;
                left = right;
                right = xorStrings(temp, feistalRound(right, subkeys[15 -
i]));
            }
            text = right + left;
            text = getIPinverse(text);
            text = new BigInteger(text, 2).toString(16);
            while (text.length() < 16) {
                text = "0" + text;
            }
            output += text;
        }
        return output;
    }
    public static String hexToString(String str) {
        String result = new String();
        char[] charArray = str.toCharArray();
        for (int i = 0; i < charArray.length; i = i + 2) {
            String st = "" + charArray[i] + "" + charArray[i + 1];
            char ch = (char) Integer.parseInt(st, 16);
            result = result + ch;
        }
        return result;
    }

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        String text, key, strText;
        System.out.println("Enter 64 bit key:");
        key = s.nextLine();
        keygen(key);
        System.out.println("Sub keys:");
        for (int i = 0; i < 16; i++) {
            System.out.println("\nSub key" + String.valueOf(i + 1) + ":" +
subkeys[i]);
        }
        System.out.println("Enter text to decrypt:");
        String encrypted = s.nextLine();
        String decrypted = Decrypt(encrypted);
        System.out.println("\nDecrypted text: " + hexToString(decrypted));
    }
}
```

OUTPUT:



```
› javac -classpath .:/run_dir/junit-4.12.jar:target/dependency
d . Main.java
› java -classpath .:/run_dir/junit-4.12.jar:target/dependency/
in
Enter 64 bit key:
9089878685848382
Sub keys:

Sub key1:000011110100000010001000110000101001000110000101 1

Sub key2:000011110100000010000100100110101000001110000000 1

Sub key3:000010110000000011000100101011010000000000100011 0

Sub key4:000110010000100010001001010001001110000110001100

Sub key5:000100010010100010001000001000000001101001100100 1

Sub key6:000100000010110010000100111010101001000000010001 1

Sub key7:010100000010110000000010000000110010011110010101 0

Sub key8:010000001010010000100100000111000000110010101000 0

Sub key9:110000001010010000100100110011000000000000111010 0

Sub key10:110000001000011000100010110000011100101011001000

Sub key11:111000001001001000100010000100001001011000011001

Sub key12:101000001001001001000010100110110001010000100100

Sub key13:001000000101001001010010000010000110101110100000

Sub key14:001001000101000101010000000110000011010000001010 1
```



```
Sub key2:000011110100000100001001001101010000011100000001

Sub key3:000010110000000110001001010110100000000001000110

Sub key4:000110010000100010001001010001001110000110001100

Sub key5:000100010010100010001000001000000011010011001001

Sub key6:000100000010110010000100111010101001000000100011

Sub key7:0101000000101100000000100000001100100111100101010

Sub key8:0100000001010010000100100000111000001100101010000

Sub key9:1100000010100100001001001100110000000000001110100

Sub key10:110000001000011000100010110000011100101011001000

Sub key11:111000001001001000100010000100001001011000011001

Sub key12:101000001001001001000010100110110001010000100100

Sub key13:001000000101001001010010000010000110101110100000

Sub key14:0010010001010001010100000001100000110100000010101

Sub key15:00000110010000001010100011110001100000000010010010

Sub key16:0000111001000000101010001100000111010001110010010
Enter text to decrypt:
49C9057CDA153D0DC683E8492720D446A9FDC33DD6BCBF36B6635663E90323ED

Decrypted text: We just encrypted something here
>
>
>
```

RESULT:

The DES decryption was successfully implemented and executed in JAVA Programming Language.