ASSIGNMENT-4

NAME: AZIM RIZAN P

REG NO: 21BCE8448

DES Algorithm:

CODE:

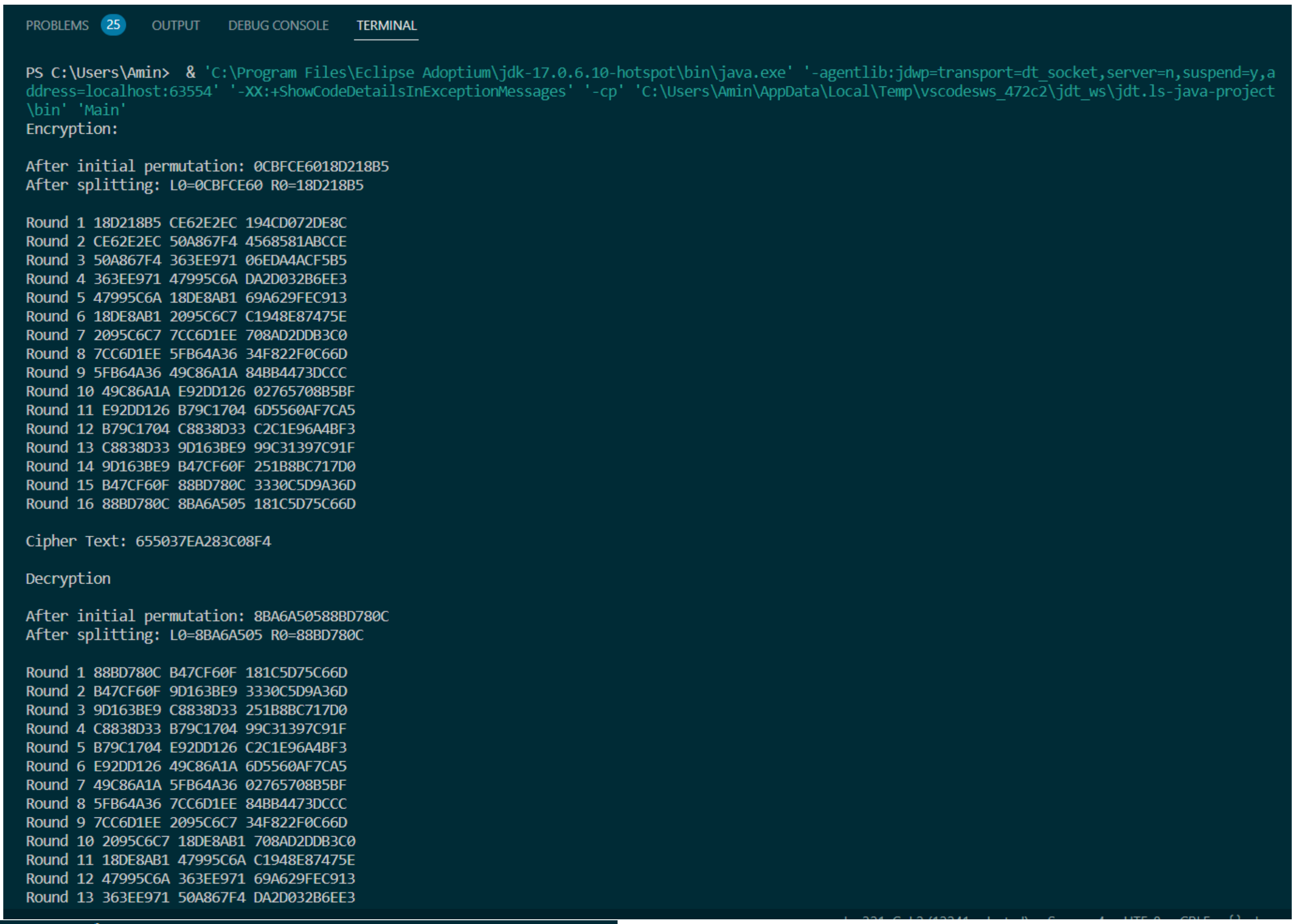
```
import java.util.*;

class Main {
    private static class DES {
        // Initial Permutation Table
        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 };

        // Inverse Initial Permutation Table
        int[] IP1
            = { 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 };

        // first key-hePermutation Table
        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 };
```

```
// second key-Permutation Table
       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 };

       // Expansion D-box Table
       int[] EP = { 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 };

       // Straight Permutation Table
       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 };

       // S-box Table
       int[][][] sbox
           = { { { 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 } } };
int[] shiftBits = { 1, 1, 2, 2, 2, 2, 2, 2,
                    1, 2, 2, 2, 2, 2, 2, 1 };

String hextoBin(String input)
{
    int n = input.length() * 4;
    input = Long.toBinaryString(
        Long.parseUnsignedLong(input, 16));
        while (input.length() < n)
            input = "0" + input;
        return input;
    }

    String binToHex(String input)
    {
        int n = (int)input.length() / 4;
        input = Long.toHexString(
            Long.parseUnsignedLong(input, 2));
        while (input.length() < n)
            input = "0" + input;
        return input;
    }

    String permutation(int[] sequence, String input)
    {
        String output = "";
        input = hextoBin(input);
        for (int i = 0; i < sequence.length; i++)
            output += input.charAt(sequence[i] - 1);
        output = binToHex(output);
        return output;
    }

    String xor(String a, String b)
    {
        // hexadecimal to decimal(base 10)
        long t_a = Long.parseUnsignedLong(a, 16);
        // hexadecimal to decimal(base 10)
        long t_b = Long.parseUnsignedLong(b, 16);
        // xor
        t_a = t_a ^ t_b;
        // decimal to hexadecimal
        a = Long.toHexString(t_a);
        // prepend 0's to maintain length
        while (a.length() < b.length())
            a = "0" + a;
        return a;
    }

    String leftCircularShift(String input, int numBits)
    {
        int n = input.length() * 4;
        int perm[] = new int[n];
        for (int i = 0; i < n - 1; i++)
            perm[i] = (i + 2);
        perm[n - 1] = 1;
        while (numBits-- > 0)
            input = permutation(perm, input);
        return input;
    }

    // preparing 16 keys for 16 rounds
    String[] getKeys(String key)
    {
        String keys[] = new String[16];
        // first key permutation
        key = permutation(PC1, key);
        for (int i = 0; i < 16; i++) {
            key = leftCircularShift(key.substring(0, 7),
                                    shiftBits[i])
                  + leftCircularShift(
                      key.substring(7, 14),
                      shiftBits[i]);
            // second key permutation
            keys[i] = permutation(PC2, key);
        }
        return keys;
    }

    // s-box lookup
    String sBox(String input)
    {
        String output = "";
        input = hextoBin(input);
        for (int i = 0; i < 48; i += 6) {
            String temp = input.substring(i, i + 6);
            int num = i / 6;
            int row = Integer.parseInt(
                temp.charAt(0) + "" + temp.charAt(5),
                2);
            int col = Integer.parseInt(
                temp.substring(1, 5), 2);
            output += Integer.toHexString(
                sbox[num][row][col]);
        }
        return output;
    }

    String round(String input, String key, int num)
    {
        // fk
        String left = input.substring(0, 8);
        String temp = input.substring(8, 16);
        String right = temp;
```

```
            // Expansion permutation
            temp = permutation(EP, temp);
            // xor temp and round key
            temp = xor(temp, key);
            // lookup in s-box table
            temp = sBox(temp);
            // Straight D-box
            temp = permutation(P, temp);
            // xor
            left = xor(left, temp);
            System.out.println("Round " + (num + 1) + " "
                               + right.toUpperCase() + " "
                               + left.toUpperCase() + " "
                               + key.toUpperCase());

            // swapper
            return right + left;
        }

        String encrypt(String plainText, String key)
        {
            int i;
            // get round keys
            String keys[] = getKeys(key);

            // initial permutation
            plainText = permutation(IP, plainText);
            System.out.println("After initial permutation: "
                               + plainText.toUpperCase());
            System.out.println(
                "After splitting: L0="
                + plainText.substring(0, 8).toUpperCase()
                + " R0="
                + plainText.substring(8, 16).toUpperCase()
                + "\n");

            // 16 rounds
            for (i = 0; i < 16; i++) {
                plainText = round(plainText, keys[i], i);
            }

            // 32-bit swap
            plainText = plainText.substring(8, 16)
                        + plainText.substring(0, 8);

            // final permutation
            plainText = permutation(IP1, plainText);
            return plainText;
```

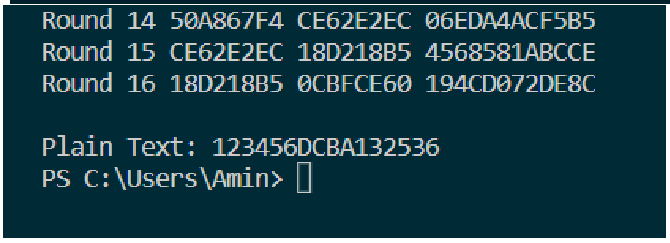
```java
        }

        String decrypt(String plainText, String key)
        {
            int i;
            // get round keys
            String keys[] = getKeys(key);

            // initial permutation
            plainText = permutation(IP, plainText);
            System.out.println("After initial permutation: "
                                   + plainText.toUpperCase());
            System.out.println(
                "After splitting: L0="
                + plainText.substring(0, 8).toUpperCase()
                + " R0="
                + plainText.substring(8, 16).toUpperCase()
                + "\n");

            // 16-rounds
            for (i = 15; i > -1; i--) {
                plainText
                    = round(plainText, keys[i], 15 - i);
            }

            // 32-bit swap
            plainText = plainText.substring(8, 16)
                        + plainText.substring(0, 8);
            plainText = permutation(IP1, plainText);
            return plainText;
        }
    }

    // Driver code
    public static void main(String args[])
    {
        String text = "123456DCBA132536";
        String key = "AABB09182736CCDD";

        DES cipher = new DES();
        System.out.println("Encryption:\n");
        text = cipher.encrypt(text, key);
        System.out.println(
            "\nCipher Text: " + text.toUpperCase() + "\n");
        System.out.println("Decryption\n");
        text = cipher.decrypt(text, key);
        System.out.println("\nPlain Text: "
                           + text.toUpperCase());
        }
}
```

OUTPUT:



PROBLEMS 25 OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Amin>  & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.6.10-hotspot\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,a
ddress=localhost:63554' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Amin\AppData\Local\Temp\vscodesws_472c2\jdt_ws\jdt.ls-java-project
\bin' 'Main'
Encryption:

After initial permutation: 0CBFCE6018D218B5
After splitting: L0=0CBFCE60 R0=18D218B5

Round 1 18D218B5 CE62E2EC 194CD072DE8C
Round 2 CE62E2EC 50A867F4 4568581ABCCE
Round 3 50A867F4 363EE971 06EDA4ACF5B5
Round 4 363EE971 47995C6A DA2D032B6EE3
Round 5 47995C6A 18DE8AB1 69A629FEC913
Round 6 18DE8AB1 2095C6C7 C1948E87475E
Round 7 2095C6C7 7CC6D1EE 708AD2DDB3C0
Round 8 7CC6D1EE 5FB64A36 34F822F0C66D
Round 9 5FB64A36 49C86A1A 84BB4473DCCC
Round 10 49C86A1A E92DD126 02765708B5BF
Round 11 E92DD126 B79C1704 6D5560AF7CA5
Round 12 B79C1704 C8838D33 C2C1E96A4BF3
Round 13 C8838D33 9D163BE9 99C31397C91F
Round 14 9D163BE9 B47CF60F 251B8BC717D0
Round 15 B47CF60F 88BD780C 3330C5D9A36D
Round 16 88BD780C 8BA6A505 181C5D75C66D

Cipher Text: 655037EA283C08F4

Decryption

After initial permutation: 8BA6A50588BD780C
After splitting: L0=8BA6A505 R0=88BD780C

Round 1 88BD780C B47CF60F 181C5D75C66D
Round 2 B47CF60F 9D163BE9 3330C5D9A36D
Round 3 9D163BE9 C8838D33 251B8BC717D0
Round 4 C8838D33 B79C1704 99C31397C91F
Round 5 B79C1704 E92DD126 C2C1E96A4BF3
Round 6 E92DD126 49C86A1A 6D5560AF7CA5
Round 7 49C86A1A 5FB64A36 02765708B5BF
Round 8 5FB64A36 7CC6D1EE 84BB4473DCCC
Round 9 7CC6D1EE 2095C6C7 34F822F0C66D
Round 10 2095C6C7 18DE8AB1 708AD2DDB3C0
Round 11 18DE8AB1 47995C6A C1948E87475E
Round 12 47995C6A 363EE971 69A629FEC913
Round 13 363EE971 50A867F4 DA2D032B6EE3
```



```
Round 14 50A867F4 CE62E2EC 06EDA4ACF5B5
Round 15 CE62E2EC 18D218B5 4568581ABCCE
Round 16 18D218B5 0CBFCE60 194CD072DE8C

Plain Text: 123456DCBA132536
PS C:\Users\Amin> 
```