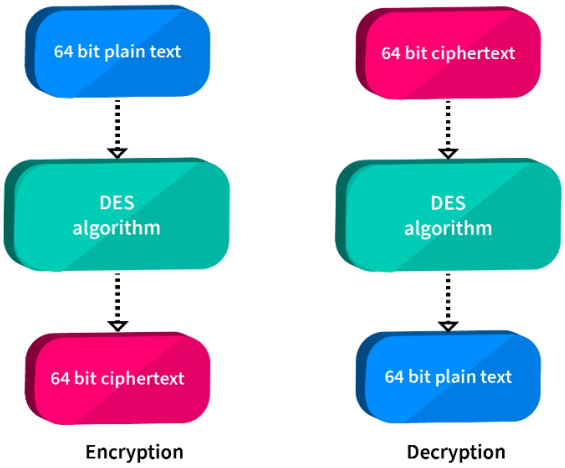
Description:

Perform encryption and decryption using Data Encryption Standard

The Data Encryption Standard algorithm is a block cipher algorithm that takes in 64-bit blocks of plaintext at a time as input and produces 64-bit blocks of cipher text at a time, using a 48-bit key for each input. In block cipher algorithms, the text to be encrypted is broken into 'blocks' of text, and each block is encrypted separately using the key.



Steps to perform Two rounds of DES

1. Take the 6 characters at a time from the file for encryption i.e. [8bitX6=64 bit block size]
2. Call a function that converts each character to ASCII and returns the binary of that character
3. Now we have 64 bit, call initial permutation function that changes the position of the bits.

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

4. Take 56 bit key and create two round key K1 and K2 by left shifting both Left half and Right half[consider only first 48 bit from 56 bit for round key]
5. Expansion Permutation (E-table), Right half is expanded from 32-bits to 48-bits

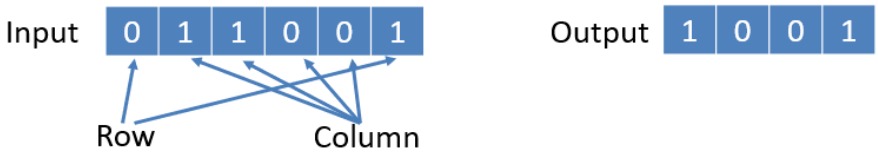
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	28
24	25	26	27	28	29
28	29	30	31	32	1

6. S-box Substitution: Accepts 48-bits from XOR operation and produces 32-bits using 8 substitution boxes (each S-box has a 6-bit i/p and 4-bit o/p).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

S-box 1

- Example:

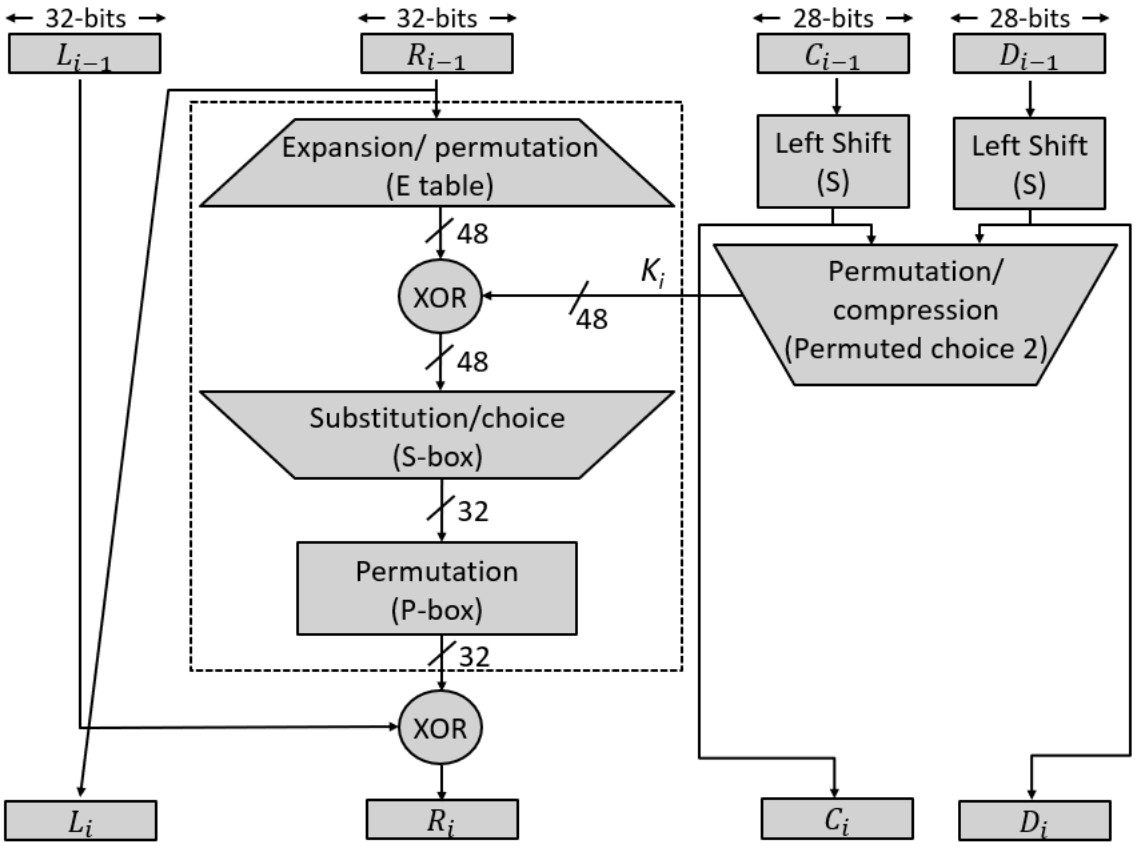


7. P-Box Permutation: rearrange the 32 bits in this fashion and read it again

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

8. XOR and Swap: Apply XOR Left portion and output of P-Box will be used as right portion for next round

Single round DES



METHODOLOGY FOLLOWED:

File1 : decryption.cpp

```
#include <iostream>
#include <bits/stdc++.h>
#include <string>
#include <fstream>
using namespace std;

string char_to_binaryString(int a)
{
    string st = "";
    while (a != 0)
    {
        st.push_back(a % 2 + '0');
        a /= 2;
    }

    int n = 8 - st.size();
    for (int i = 0; i < n; i++)
    {
        st.append("0");
    }
    reverse(st.begin(), st.end());
    return st;
}

// IP -> initial prmutation array;
int IP[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};

string text64_to_initial_permutation(string text64)
{
    string temp;
    for (int i = 0; i < 64; i++)
    {
        temp.push_back(text64[IP[i] - 1]);
    }
    return temp;
}
```

```
int E[48] = {
    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, 28,
    24, 25, 26, 27, 28, 29,
    28, 29, 30, 31, 32, 1};

string expantion_32_to_48(string st)
{
    string temp;
    for (int i = 0; i < 48; i++)
    {
        temp.push_back(st[E[i] - 1]);
    }
    return temp;
}

// binary string to int
int bsti(string st)
{
    int a = 0;
    int p = 1;
    for (int i = st.length() - 1; i >= 0; i--)
    {
        if (st[i] == '1')
        {
            a += p;
        }
        p *= 2;
    }

    return a;
}

void cheack_permutation(int a[])
{
    unordered_map<int, int> m;

    for (int i = 0; i < 64; i++)
        m[a[i]]++;

    for (int i = 0; i < 64; i++)
    {
        if (m[i] > 1)
```

```
        {
            cout << "duplicate";
            break;
        }
    }
}

int pbox[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};

string pboxPermutation(string st)
{
    string temp = "";
    for (int i = 0; i < 32; i++)
    {
        temp.push_back(st[pbox[i] - 1]);
    }
    return temp;
}

string XOR(string st1, string st2)
{

    string temp;
    for (int i = 0; i < st1.size(); i++)
    {
        if (st1[i] == st2[i])
        {
            temp.push_back('0');
        }
        else
        {
            temp.push_back('1');
        }
    }
    return temp;
}

int sbox[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, 10, 3, 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}};

string compress_48_to_32(string st)
{

    string cmp = "";
```

```
    for (int i = 0; i < 8; i++)
    {
        string stx = st.substr(6 * i, 6);

        int col = bsti(stx.substr(1, 4));
        int row = bsti(stx.substr(0, 1) + stx.substr(5, 1));
        // char_to_binaryString(sbox[row][col])->it gives 8 bit binary string
like 9-> 00001001.
        // we wants only 4bit (0-15) hence we

        cmp.append(char_to_binaryString(sbox[row][col]).substr(4, 4));
    }

    return cmp;
}

string function1(string text32, string key)
{
    // expantion

    string Etext48 = expantion_32_to_48(text32);

    // xor whith key

    string xorSt = XOR(Etext48, key);

    cout << "\n       " << Etext48 << "\n";
    cout << "       " << key << "\n";
    cout << "xorSt: " << xorSt << "\n";

    //  compretion / sunstitution /choise S-box
    string cmp = compress_48_to_32(xorSt);
    return pboxPermutation(cmp);
}
int IIP[64] = {
    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};

string inverse_initial_permutation(string st)
{
    string temp;
```

```
    for (int i = 0; i < 64; i++)
    {
        temp.push_back(st[IIP[i] - 1]);
    }
    return temp;
}

ifstream fin2;

string Decryption(string text64)
{

    // initial permutation

    string initial_permutation = text64_to_initial_permutation(text64);
    cout << "\nIP: " << initial_permutation << " ";

    // left - right

    string L1 = initial_permutation.substr(0, 32);

    string R1 = initial_permutation.substr(32, 32);

    string key1;
    getline(fin2, key1);

    string key2;
    getline(fin2, key2);

    string L2 = R1;
    string fR1 = function1(R1, key2);
    string R2 = XOR(L1, fR1);

    // for round2

    string L3 = R2;
    string fR2 = function1(R2, key1);
    string R3 = XOR(L2, fR2);

    //  L3+R3 (-SWAP->) R3+L3  , HENCE WE RETURN R3+L3

    return R3 + L3;
}

int main()
{

    ifstream fin;
```

```
    fin.open("output.txt");

    fin2.open("key.txt");

    ofstream fout;
    fout.open("doutput.txt");

    string st;

    unordered_map<string, char> m;
    unordered_map<char, string> binary;

    for (int i = 0; i < 256; i++)
    {
        char ch = i;
        binary[ch] = char_to_binaryString(i);
        m[binary[ch]] = ch;
    }
    char ch;

    string plainText64 = "";

    int c = 0;

    while (fin.get(ch))
    {

        plainText64.append(binary[ch]);

        c++;
        if (c == 8)
        {
            string ciphertext = "";
            string str = Decryption(plainText64);

            string binary_ciphetext = inverse_initial_permutation(str);

            for (int i = 0; i < 8; i++)
            {
                int a = i * 8;
                char ch = m[binary_ciphetext.substr(i * 8, 8)];
                ciphertext.push_back(ch);
            }

            fout << ciphertext;
            c = 0;
            plainText64 = "";
        }
```

```
    }

    fin.close();
    fout.close();
    fin2.close();
}
```

File2: DES.cpp

```
#include <iostream>
#include <bits/stdc++.h>
#include <string>
#include <fstream>
using namespace std;

string char_to_binaryString(int a)
{
    string st = "";
    while (a != 0)
    {
        st.push_back(a % 2 + '0');
        a /= 2;
    }

    int n = 8 - st.size();
    for (int i = 0; i < n; i++)
    {
        st.append("0");
    }
    reverse(st.begin(), st.end());
    return st;
}

// IP -> initial prmutation array;
int IP[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};

string text64_to_initial_permutation(string text64)
{
    string temp;
    for (int i = 0; i < 64; i++)
```

```
    {
        temp.push_back(text64[IP[i] - 1]);
    }
    return temp;
}

int E[48] = {
    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, 28,
    24, 25, 26, 27, 28, 29,
    28, 29, 30, 31, 32, 1};

string expantion_32_to_48(string st)
{
    string temp;
    for (int i = 0; i < 48; i++)
    {
        temp.push_back(st[E[i] - 1]);
    }
    return temp;
}

// binary string to int
int bsti(string st)
{
    int a = 0;
    int p = 1;
    for (int i = st.length() - 1; i >= 0; i--)
    {
        if (st[i] == '1')
        {
            a += p;
        }
        p *= 2;
    }

    return a;
}

void cheack_permutation(int a[])
{
    unordered_map<int, int> m;

    for (int i = 0; i < 64; i++)
```

```
        m[a[i]]++;

    for (int i = 0; i < 64; i++)
    {
        if (m[i] > 1)
        {
            cout << "duplicate";
            break;
        }
    }
}

int pbox[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};

string pboxPermutation(string st)
{
    string temp = "";
    for (int i = 0; i < 32; i++)
    {
        temp.push_back(st[pbox[i] - 1]);
    }
    return temp;
}

string XOR(string st1, string st2)
{

    string temp;
    for (int i = 0; i < st1.size(); i++)
    {
        if (st1[i] == st2[i])
        {
            temp.push_back('0');
        }
        else
        {
            temp.push_back('1');
        }
    }
    return temp;
}

int sbox[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, 10, 3, 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}};
```

```
string compress_48_to_32(string st)
{

    string cmp = "";
    for (int i = 0; i < 8; i++)
    {
        string stx = st.substr(6 * i, 6);

        int col = bsti(stx.substr(1, 4));
        int row = bsti(stx.substr(0, 1) + stx.substr(5, 1));
        // char_to_binaryString(sbox[row][col])->it gives 8 bit binary string
like 9-> 00001001.
        // we wants only 4bit (0-15) hence we

        cmp.append(char_to_binaryString(sbox[row][col]).substr(4, 4));
    }

    return cmp;
}

string function1(string text32, string key)
{
    // expantion

    string Etext48 = expantion_32_to_48(text32);

    // xor whith key

    string xorSt = XOR(Etext48, key);

    //  compretion / sunstitution /choise S-box
    string cmp = compress_48_to_32(xorSt);
    return pboxPermutation(cmp);
}
int IIP[64] = {
    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};
string inverse_initial_permutation(string st)
{
    string temp;
```

```
    for (int i = 0; i < 64; i++)
    {
        temp.push_back(st[IIP[i] - 1]);
    }
    return temp;
}

ofstream key_file_out;

string Encryption(string text64)
{

    // initial permutation

    string initial_permutation = text64_to_initial_permutation(text64);

    // left - right

    string L1 = initial_permutation.substr(0, 32);

    string R1 = initial_permutation.substr(32, 32);

    // key_permutation_64;

    string key_56 = "";
    for (int i = 0; i < 56; i++)
    {
        char ch = rand() % 2 + '0';
        key_56.push_back(ch);
    }
    //  cout << "\n56 key permutation: " << key_56 << "\n";

    // initial 56 bit permutation into right  & left part

    string lkey_0 = key_56.substr(0, 28);  // left 28 bit of 56 bit
permutation1
    string rkey_0 = key_56.substr(28, 28); // rigt 28 bit of 56 bit
permutation1

    // for round1

    string lkey_1 = lkey_0.substr(1, 28) + lkey_0.substr(0, 1); // circular
left shift on lkey
    string rkey_1 = rkey_0.substr(1, 28) + rkey_0.substr(0, 1); // circular
left shift on rkey
    string key1 = (lkey_1 + rkey_1).substr(0, 48);              // first 48
bits form 56 bit    lkey_1 + rkey_2
```

```
    string L2 = R1;
    string fR1 = function1(R1, key1);
    string R2 = XOR(L1, fR1);

    key_file_out << key1 << "\n";

    // cout << "L2 : " << L2 << " R2: " << R2 << "\n";

    // for round2

    string lkey_2 = lkey_1.substr(1, 28) + lkey_1.substr(0, 1); // circular
left shift on lkey
    string rkey_2 = rkey_1.substr(1, 28) + rkey_1.substr(0, 1); // circular
left shift on rkey

    string key2 = (lkey_2 + rkey_2).substr(0, 48); // first 48 bits form 56
bit    lkey_2 + rkey_2

    key_file_out << key2 << "\n";

    string L3 = R2;
    string fR2 = function1(R2, key2);
    string R3 = XOR(L2, fR2);

    //  cout << "L3 : " << L3 << " R3: " << R3 << "\n";

    //  L3+R3 (-SWAP->) R3+L3  , HENCE WE RETURN R3+L3

    return R3 + L3;
}

int main()
{
    ifstream fin;
    fin.open("input.txt");

    ofstream fout;
    fout.open("output.txt");

    key_file_out.open("key.txt");

    string st;

    unordered_map<string, char> m;
    unordered_map<char, string> binary;

    for (int i = 0; i < 256; i++)
    {
```

```
        char ch = i;
        string s = char_to_binaryString(i);
        binary[ch] = s;
        m[s] = ch;
    }

    char ch;

    int c = 0;
    string plainText64 = "";

    int cnt = 0;
    while (fin.get(ch))
    {
        cnt++;

        plainText64.append(binary[ch]);

        c++;
        if (c == 8)
        {
            string ciphertext = "";

            string str = Encryption(plainText64);
            string binary_ciphetext = inverse_initial_permutation(str);

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

                char ch = m[binary_ciphetext.substr(i * 8, 8)];
                ciphertext.push_back(ch);
            }

            fout << ciphertext;
            c = 0;
            plainText64 = "";
        }
    }

    cout << "\n************" << cnt << "\n";

    if (c != 0)
    {
        string chiphertext = "";
        int sz = plainText64.length();

        for (int i = sz / 8; i < 8; i++)
        {
```

```cpp
            plainText64.append(binary[' ']);
        }

        string str = Encryption(plainText64);
        string binary_ciphetext = inverse_initial_permutation(str);

        for (int i = 0; i < 8; i++)
        {
            char ch = m[binary_ciphetext.substr(i * 8, 8)];

            chiphertext.push_back(ch);
        }
        fout << chiphertext;
    }

    fin.close();
    fout.close();
    key_file_out.close();
}
```

File3: input.txt

hello how jjbkjbjkbbbbbbjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj

kkkkkkkkkkkkkkkkkkkkk nirma kkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk

llllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllll

hello how jjbkjbjkbbbbbbjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj

kkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk

llllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllll

hello how jjbkjbjkbbbbbbjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj

kkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk

llllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllll

File4: key.txt (these are keys through which we encrypted the message. these keys are generated during performing encryption. we store these keys in key.txt file for decrypt the message.)

100100000111111101010010010111010101110110110111

0010000001111111010100100101110101011101101101110

110010000000000101000110110010001001011000111100

100100000000010100011011001100100101100011111000

0011110001011101000100011110111110100000100101 0

0111100010111010001000111100111110100000100101 01

0100001010010110000110101110110101101100100011 01

1000010100101100001101011100101011011001000110 11

0000001101100000101011001000100001110001001111 00

0000011011000001010110010000000011100010011110 01

0111101010011001011010011011010011110111101111 00

1111010100110010110100110110100111101111011110 01

1011111000110100010001110100101100011010000110 10

0111110001101000100011101001011000110100001101 01

1101101101001001101111010110011110000001010001 11

1011011010010011011110101101111100000010100011 10

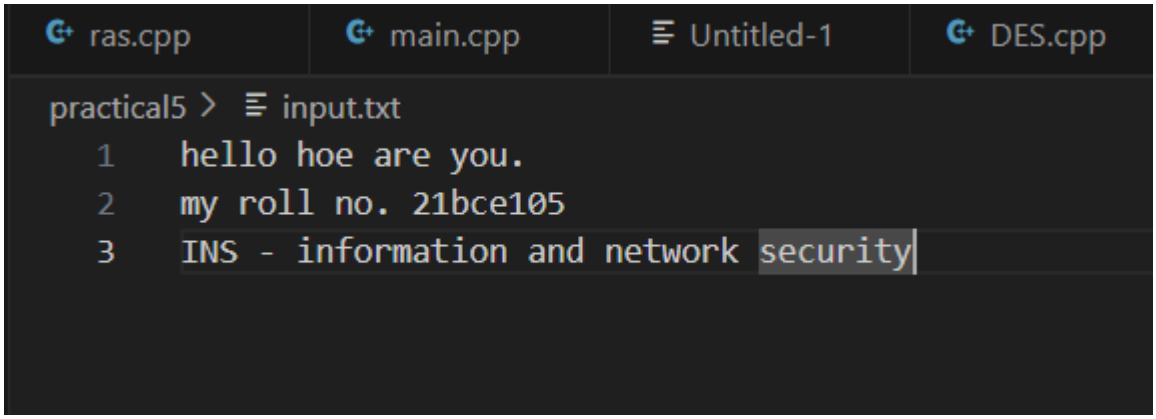
0110000100110001001000101011001001110111010001 01

1100001001100010010001010110010011101110100010 11

0111110000101010100111001100010111000101010100 01

1111100001010101001110011000101110001010101000 11

File5: input.txt



File6: output.txt

ras.cpp | main.cpp | Untitled-1 | DES.cpp

practical5 > output.txt

```
1  ?bIENQ+8?1?e¥c!?????????SOH???&
2  ?ag+u??_ECAN2?*EM&@SYN?ACKDLE7??Vj?jNAKSOH?
3  ??v!O?ZDC1'Q6????BEL?Z?ȯ
```

File7: doutput.txt

ras.cpp | main.cpp | Untitled-1 | DES.cpp

practical5 > doutput.txt

```
1  hello hoe are you.
2  my roll no. 21bce105
3  INS - information and network security
```