**Lab-10**

**Name : Divan MunafSha Salimsha**

**Roll No : CE035**

**Subject : NIS**

**Student ID: 19CEUBG006**

**Aim: Write a program to implement DES cipher.**

* **Round Function Implementation**

**Source Code:**

**Programming Language : C++**

**Ans:**

#include "../functions.h"

string bin2hex(string str)

{

    unordered\_map<string, char> bin;

    bin.insert({string("0000"), '0'});

    bin.insert({string("0001"), '1'});

    bin.insert({string("0010"), '2'});

    bin.insert({string("0011"), '3'});

    bin.insert({string("0100"), '4'});

    bin.insert({string("0101"), '5'});

    bin.insert({string("0110"), '6'});

    bin.insert({string("0111"), '7'});

    bin.insert({string("1000"), '8'});

    bin.insert({string("1001"), '9'});

    bin.insert({string("1010"), 'A'});

    bin.insert({string("1011"), 'B'});

    bin.insert({string("1100"), 'C'});

    bin.insert({string("1101"), 'D'});

    bin.insert({string("1110"), 'E'});

bin.insert({string("1111"), 'F'});

    string hex = "";

    for (int i = 0; i <= (str.length() - 3); i += 4)

    {

        string temp = str.substr(i, 4);

        hex += bin[temp];

    }

    return hex;

}

string hex2bin(string str)

{

    unordered\_map<char, string> hex;

    hex.insert({'0', string("0000")});

    hex.insert({'1', string("0001")});

    hex.insert({'2', string("0010")});

    hex.insert({'3', string("0011")});

    hex.insert({'4', string("0100")});

    hex.insert({'5', string("0101")});

    hex.insert({'6', string("0110")});

    hex.insert({'7', string("0111")});

    hex.insert({'8', string("1000")});

    hex.insert({'9', string("1001")});

    hex.insert({'A', string("1010")});

    hex.insert({'B', string("1011")});

    hex.insert({'C', string("1100")});

    hex.insert({'D', string("1101")});

    hex.insert({'E', string("1110")});

hex.insert({'F', string("1111")});

    string binary = "";

    for (char ch : str)

    {

        binary += hex[ch];

    }

    return binary;

}

string int2bin(int num)

{

    unordered\_map<int, string> bin;

    bin.insert({0, string("0000")});

    bin.insert({1, string("0001")});

    bin.insert({2, string("0010")});

    bin.insert({3, string("0011")});

    bin.insert({4, string("0100")});

    bin.insert({5, string("0101")});

    bin.insert({6, string("0110")});

    bin.insert({7, string("0111")});

    bin.insert({8, string("1000")});

    bin.insert({9, string("1001")});

    bin.insert({10, string("1010")});

    bin.insert({11, string("1011")});

    bin.insert({12, string("1100")});

    bin.insert({13, string("1101")});

    bin.insert({14, string("1110")});

    bin.insert({15, string("1111")});

    return bin[num];

}

int bin2num(string bin)

{

    unordered\_map<string, int> number;

    number.insert({string("0000"), 0});

    number.insert({string("0001"), 1});

    number.insert({string("0010"), 2});

    number.insert({string("0011"), 3});

    number.insert({string("0100"), 4});

    number.insert({string("0101"), 5});

    number.insert({string("0110"), 6});

    number.insert({string("0111"), 7});

    number.insert({string("1000"), 8});

    number.insert({string("1001"), 9});

    number.insert({string("1010"), 10});

    number.insert({string("1011"), 11});

    number.insert({string("1100"), 12});

    number.insert({string("1101"), 13});

    number.insert({string("1110"), 14});

    number.insert({string("1111"), 15});

    return number[bin];

}

// P-box

string expansion(string s)

{

    s = hex2bin(s);

    int expansion\_p\_box[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, 25, 24, 25, 26, 27,

        28, 29, 28, 29, 30, 31, 32, 1};

    string result = "";

    int len = s.length();

    for (int i = 0; i < 48; i++)

    {

        result += s[expansion\_p\_box[i] - 1];

    }

    return bin2hex(result);

}

void printValuesOfEachStep(string key, string r\_expanded, string r\_xored, string r\_sbox, string r\_dbox)

{

    cout << "\nRound key in binary";

    cout << "\n-----------------------------------------------------------------------------------------------" << endl;

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

        cout << key[i] << " ";

    cout << "\n-----------------------------------------------------------------------------------------------" << endl

         << endl;

    cout << "\nafter expansion of P-box, plain text in binary";

    cout << "\n-----------------------------------------------------------------------------------------------" << endl;

    for (int i = 0; i < r\_expanded.length(); ++i)

        cout << r\_expanded[i] << " ";

    cout << "\n-----------------------------------------------------------------------------------------------" << endl

         << endl;

    cout << "\nafter expansion of P-box, plain text in hexadecimal";

    cout << "\n-----------------------------------------------------------------------------------------------" << endl;

    cout << bin2hex(r\_expanded);

    cout << "\n-----------------------------------------------------------------------------------------------" << endl

         << endl;

    cout << "\nafter xor with round key, plain text in binary";

    cout << "\n-----------------------------------------------------------------------------------------------" << endl;

    for (int i = 0; i < r\_xored.length(); ++i)

        cout << r\_xored[i] << " ";

    cout << "\n-----------------------------------------------------------------------------------------------" << endl

         << endl;

    cout << "\nafter xor with round key, plain text in hexadecimal";

    cout << "\n-----------------------------------------------------------------------------------------------" << endl;

    cout << bin2hex(r\_xored);

    cout << "\n-----------------------------------------------------------------------------------------------" << endl

         << endl;

    cout << "\nafter applying S-box, plain text in binary";

    cout << "\n-----------------------------------------------------------------------------------------------" << endl;

    for (int i = 0; i < r\_sbox.length(); ++i)

        cout << r\_sbox[i] << " ";

    cout << "\n-----------------------------------------------------------------------------------------------" << endl

         << endl;

    cout << "\nafter applying S-box, plain text in hexadecimal";

    cout << "\n-----------------------------------------------------------------------------------------------" << endl;

    cout << bin2hex(r\_sbox);

    cout << "\n-----------------------------------------------------------------------------------------------" << endl

         << endl;

    cout << "\nafter straight permutation of D-box, plain text in binary";

    cout << "\n-----------------------------------------------------------------------------------------------" << endl;

    for (int i = 0; i < r\_dbox.length(); ++i)

        cout << r\_dbox[i] << " ";

    cout << "\n-----------------------------------------------------------------------------------------------" << endl

         << endl;

    cout << "\nafter straight permutation of D-box, plain text in hexadecimal";

    cout << "\n-----------------------------------------------------------------------------------------------" << endl;

    cout << bin2hex(r\_dbox);

    cout << "\n-----------------------------------------------------------------------------------------------" << endl

         << endl;

}

string xor\_(string s1, string s2)

{

    s1 = hex2bin(s1);

    s2 = hex2bin(s2);

    string result = "";

    for (int i = 0; i < s1.length(); ++i)

    {

        if (s1[i] == s2[i])

            result += '0';

        else

            result += '1';

    }

    return bin2hex(result);

}

//S-Box

string applySbox(string str)

{

    str = hex2bin(str);

    // cout<<"l: "<<str.length()<<endl;

    string result = "";

    int sBox[8][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},

                          {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 j = 1;

    // cout<<str.length()<<endl;

    for (int i = 0; i < (str.length()); i += 6)

    {

        // cout<<"j: "<<j++<<endl;

        string s = str.substr(i, 6);

        char prefix = s[0], suffix = s[5];

        string row = "00", col = "";

        row = row + prefix + suffix;

        // cout<<row<<endl;

        col = s.substr(1, 4);

        result += int2bin(sBox[i / 6][bin2num(row)][bin2num(col)]);

    }

    return bin2hex(result);

}

string straightD(string str)

{

    str = hex2bin(str);

    string result = "";

    int straight\_d\_box[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};

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

    {

        result += str[straight\_d\_box[i] - 1];

    }

    return bin2hex(result);

}

string initialPertmutation(string str)

{

    str = hex2bin(str);

    string permuted = "";

    int initialPerm[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};

    for (int i = 0; i < 64; ++i)

    {

        permuted += str[initialPerm[i] - 1];

    }

    return bin2hex(permuted);

}

string f(string r, string key)

{

    string r\_expanded, r\_xor, r\_sbox, r\_dbox;

    r\_expanded = expansion(r);

    r\_xor = xor\_(r\_expanded, key);

    r\_sbox = applySbox(r\_xor);

    r\_dbox = straightD(r\_sbox);

    printValuesOfEachStep(hex2bin(key), hex2bin(r\_expanded), hex2bin(r\_xor), hex2bin(r\_sbox), hex2bin(r\_dbox));

    return r\_dbox;

}

pair<string, string> get\_data()

{

    string pt, key;

    cout << "Enter Hexadecimal plain text of 16 charecter long : ";

    cin>>pt;

    if (pt.length() != 16)

    {

        cout << "Length is not 16.Please Enter valid length of text\n"

             << endl;

        exit(1);

    }

    cout << "Enter Hexadecimal Round Key of 12 charecter long : ";

    cin>>key;

    if (key.length() != 12)

    {

        cout << "Length is not 12.Please Enter valid length of Key\n"

             << endl;

        exit(1);

    }

    return make\_pair(pt, key);

}

int main()

{

    string pt, key, pt\_permuted, left0, right0, right1, left1, pt\_after\_round\_1;

    pair<string, string> data = get\_data();

    pt = data.first;

    key = data.second;

    // pt\_permuted = (initialPertmutation(pt));

    left0 = pt.substr(0, 8);

    right0 = pt.substr(8, 8);

    left1 = right0;

    // R[i] = L[i-1] xor f(R[i-1],Key)

    right1 = (xor\_(left0, f(right0, key)));

    pt\_after\_round\_1 = left1 + right1;

    cout << "Text after first round: " << pt\_after\_round\_1 << endl;

    return 0;

}

**Input & Output Screenshots:**

**1)**

