Lab-10

Name: Jenil J Gandhi

Roll-No: CE047

Subject: Network & Information Security.

Aim: Write a program to implement DES Cipher.

- **Round Function Implementation.**

Code:

```
/*****************************************************
Written by:  Jenil J Gandhi
Subject:     Network Information Security
Lab-no:      Lab-10
Description: Implement DES Encryption Round Function
Guidance by: Prof Mrudang T Mehta
Date:        24/02/2022
*****************************************************/

#include <bits/stdc++.h>
using namespace std;

void fin()
{
    freopen("approx_input.txt", "r", stdin);
```

```
    freopen("approx_output.txt", "w", stdout);
}

void traceBoolVec(vector<bool> vec, string message = "")
{
    cout << message << " ";
    for (auto x : vec)
    {
        cout << x;
    }
    cout << endl;
}

vector<bool> converHexToBinary(string HexInput)
{
    vector<bool> retAns;
    for (int i = 0; i < HexInput.size(); i++)
    {
        char current_alpha = HexInput[i];
        switch (current_alpha)
        {
        case '0':
            retAns.push_back(0);
            retAns.push_back(0);
            retAns.push_back(0);
            retAns.push_back(0);
            break;
        case '1':
            retAns.push_back(0);
            retAns.push_back(0);
            retAns.push_back(0);
            retAns.push_back(1);
            break;
        case '2':
            retAns.push_back(0);
            retAns.push_back(0);
            retAns.push_back(1);
            retAns.push_back(0);
            break;
        case '3':
            retAns.push_back(0);
```

```
            retAns.push_back(0);
            retAns.push_back(1);
            retAns.push_back(1);
            break;
        case '4':
            retAns.push_back(0);
            retAns.push_back(1);
            retAns.push_back(0);
            retAns.push_back(0);
            break;
        case '5':
            retAns.push_back(0);
            retAns.push_back(1);
            retAns.push_back(0);
            retAns.push_back(1);
            break;
        case '6':
            retAns.push_back(0);
            retAns.push_back(1);
            retAns.push_back(1);
            retAns.push_back(0);
            break;
        case '7':
            retAns.push_back(0);
            retAns.push_back(1);
            retAns.push_back(1);
            retAns.push_back(1);
            break;
        case '8':
            retAns.push_back(1);
            retAns.push_back(0);
            retAns.push_back(0);
            retAns.push_back(0);
            break;

        case '9':
            retAns.push_back(1);
            retAns.push_back(0);
            retAns.push_back(0);
            retAns.push_back(1);
            break;
```

```
        case 'A':
            retAns.push_back(1);
            retAns.push_back(0);
            retAns.push_back(1);
            retAns.push_back(0);
            break;

        case 'B':
            retAns.push_back(1);
            retAns.push_back(0);
            retAns.push_back(1);
            retAns.push_back(1);
            break;

        case 'C':
            retAns.push_back(1);
            retAns.push_back(1);
            retAns.push_back(0);
            retAns.push_back(0);
            break;

        case 'D':
            retAns.push_back(1);
            retAns.push_back(1);
            retAns.push_back(0);
            retAns.push_back(1);
            break;
        case 'E':
            retAns.push_back(1);
            retAns.push_back(1);
            retAns.push_back(1);
            retAns.push_back(0);
            break;
        case 'F':
            retAns.push_back(1);
            retAns.push_back(1);
            retAns.push_back(1);
            retAns.push_back(1);
            break;
```

```
        default:
            cout << "Error processing input please check it again!";
            exit(0);
            break;
        }
    }
    return retAns;
}

class DESRound
{
public:
    vector<bool> key48;
    vector<bool> input64;

    vector<bool> padZeros(vector<bool> vec, int len)
    {
        if (vec.size() == len)
        {
            return vec;
        }
        else
        {
            int cntr = len - vec.size();
            for (int i = 0; i < cntr; i++)
            {
                vec.insert(vec.begin(), 0);
            }
        }
        return vec;
    }

    DESRound(vector<bool> key48, vector<bool> input)
    {
        this->key48 = key48;
        this->input64 = input;
    }

    int binaryToInteger(string bin)
    {
        int ans = 0;
```

```
        reverse(bin.begin(), bin.end());
        for (int i = 0; i < bin.size(); i++)
        {
            if (bin[i] == '1')
            {
                ans += (int(pow(2, i)));
            }
        }
        return ans;
    }

    vector<bool> intToBinary(int num)
    {
        vector<bool> retVect;
        while (num > 0)
        {
            retVect.push_back(num % 2);
            num /= 2;
        }
        reverse(retVect.begin(), retVect.end());
        return padZeros(retVect, 4);
    }
    vector<vector<bool>> splitInputHelper()
    {
        vector<bool> l1(this->input64.begin(), this->input64.begin() +
32);
        vector<bool> r1(this->input64.begin() + 32, this->input64.end());
        vector<vector<bool>> retVec;
        retVec.push_back(l1);
        retVec.push_back(r1);
        return retVec;
    }

    vector<vector<bool>> splitVector(vector<bool> vec, int row, int col)
    {
        vector<vector<bool>> retVect;
        int colCnt = 0;
        for (int i = 0; i < row; i++)
        {
            vector<bool> temp;
            for (int j = 0; j < col; j++)
```

```
            {
                temp.push_back(vec[i * col + j]);
            }
            retVect.push_back(temp);
        }
        return retVect;
    }

    vector<bool> expansionPBox(vector<bool> r1)
    {
        vector<int> expansionTable{
            31, 0, 1, 2, 3, 4,
            3, 4, 5, 6, 7, 8,
            7, 8, 9, 10, 11, 12,
            11, 12, 13, 14, 15, 16,
            15, 16, 17, 18, 19, 20,
            19, 20, 21, 22, 23, 24,
            23, 24, 25, 26, 27, 28,
            27, 28, 29, 30, 31, 0};
        vector<bool> returnVec;
        for (int i = 0; i < expansionTable.size(); i++)
        {
            returnVec.push_back(r1[expansionTable[i]]);
        }
        return returnVec;
    }

    vector<bool> XorVectors(vector<bool> first, vector<bool> second)
    {
        if (first.size() != second.size())
        {
            exit(0);
        }
        vector<bool> retVect;
        for (int i = 0; i < first.size(); i++)
        {
            retVect.push_back(first[i] ^ second[i]);
        }
        return retVect;
    }
```

```
    vector<string> SboxHelper(vector<bool> inp)
    {
        string row_selector = "";
        string col_selector = "";
        for (int i = 0; i < inp.size(); i++)
        {
            char curChar;
            if (inp[i] == 1)
            {
                curChar = '1';
            }
            else
            {
                curChar = '0';
            }
            if (i == 0 || i == inp.size() - 1)
            {
                row_selector += curChar;
            }
            else
            {
                col_selector += curChar;
            }
        }
        vector<string> retVect;
        retVect.push_back(row_selector);
        retVect.push_back(col_selector);
        return retVect;
    }
    vector<bool> SBox(vector<vector<bool>> input)
    {
        vector<vector<int>> S1;
        vector<int> s1_1{14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9,
0, 7};
        vector<int> s1_2{0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5,
3, 8};
        vector<int> s1_3{4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10,
5, 0};
        vector<int> s1_4{15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6,
13};
        S1.push_back(s1_1);
```

```
        S1.push_back(s1_2);
        S1.push_back(s1_3);
        S1.push_back(s1_4);

        vector<vector<int>> S2;
        vector<int> s2_1{15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5,
10};
        vector<int> s2_2{3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9,
11, 5};
        vector<int> s2_3{0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2,
15};
        vector<int> s2_4{13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5,
14, 9};
        S2.push_back(s2_1);
        S2.push_back(s2_2);
        S2.push_back(s2_3);
        S2.push_back(s2_4);

        vector<vector<int>> S3;
        vector<int> s3_1{10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4,
2, 8};
        vector<int> s3_2{13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11,
15, 1};
        vector<int> s3_3{13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10,
14, 7};
        vector<int> s3_4{1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2,
12};
        S3.push_back(s3_1);
        S3.push_back(s3_2);
        S3.push_back(s3_3);
        S3.push_back(s3_4);

        vector<vector<int>> S4;
        vector<int> s4_1{7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4,
15};
        vector<int> s4_2{13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10,
14, 9};
        vector<int> s4_3{10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2,
8, 4};
        vector<int> s4_4{3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2,
14};
```

```
        S4.push_back(s4_1);
        S4.push_back(s4_2);
        S4.push_back(s4_3);
        S4.push_back(s4_4);

        vector<vector<int>> S5;
        vector<int> s5_1{2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0,
14, 9};
        vector<int> s5_2{14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9,
8, 6};
        vector<int> s5_3{4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0,
14};
        vector<int> s5_4{11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4,
5, 3};
        S5.push_back(s5_1);
        S5.push_back(s5_2);
        S5.push_back(s5_3);
        S5.push_back(s5_4);

        vector<vector<int>> S6;
        vector<int> s6_1{12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5,
11};
        vector<int> s6_2{10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11,
3, 8};
        vector<int> s6_3{9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13,
11, 6};
        vector<int> s6_4{4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8,
13};
        S6.push_back(s6_1);
        S6.push_back(s6_2);
        S6.push_back(s6_3);
        S6.push_back(s6_4);

        vector<vector<int>> S7;
        vector<int> s7_1{4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10,
6, 1};
        vector<int> s7_2{13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15,
8, 6};
        vector<int> s7_3{1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5,
9, 2};
```

```
        vector<int> s7_4{6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3,
12};
        S7.push_back(s7_1);
        S7.push_back(s7_2);
        S7.push_back(s7_3);
        S7.push_back(s7_4);

        vector<vector<int>> S8;
        vector<int> s8_1{13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0,
12, 7};
        vector<int> s8_2{1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14,
9, 2};
        vector<int> s8_3{7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3,
5, 8};
        vector<int> s8_4{2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6,
11};
        S8.push_back(s8_1);
        S8.push_back(s8_2);
        S8.push_back(s8_3);
        S8.push_back(s8_4);

        int c = 0;
        vector<bool> ans32;
        for (auto x : input)
        {
            vector<string> inps = SboxHelper(x);
            int rowIdx = binaryToInteger(inps[0]);
            int colIdx = binaryToInteger(inps[1]);
            if (c == 0)
            {
                int number = S1[rowIdx][colIdx];
                vector<bool> vec = intToBinary(number);
                ans32.insert(ans32.end(), vec.begin(), vec.end());
            }
            else if (c == 1)
            {
                int number = S2[rowIdx][colIdx];
                vector<bool> vec = intToBinary(number);
                ans32.insert(ans32.end(), vec.begin(), vec.end());
            }
            else if (c == 2)
```

```
            {
                int number = S3[rowIdx][colIdx];
                vector<bool> vec = intToBinary(number);
                ans32.insert(ans32.end(), vec.begin(), vec.end());
            }

            else if (c == 3)
            {
                int number = S4[rowIdx][colIdx];
                vector<bool> vec = intToBinary(number);
                ans32.insert(ans32.end(), vec.begin(), vec.end());
            }
            else if (c == 4)
            {
                int number = S5[rowIdx][colIdx];
                vector<bool> vec = intToBinary(number);
                ans32.insert(ans32.end(), vec.begin(), vec.end());
            }
            else if (c == 5)
            {
                int number = S6[rowIdx][colIdx];
                vector<bool> vec = intToBinary(number);
                ans32.insert(ans32.end(), vec.begin(), vec.end());
            }
            else if (c == 6)
            {
                int number = S7[rowIdx][colIdx];
                vector<bool> vec = intToBinary(number);
                ans32.insert(ans32.end(), vec.begin(), vec.end());
            }
            else if (c == 7)
            {
                int number = S8[rowIdx][colIdx];
                vector<bool> vec = intToBinary(number);
                ans32.insert(ans32.end(), vec.begin(), vec.end());
            }

            c++;
        }
        return ans32;
    }
```

```
    vector<bool> StraightPBox(vector<bool> vec)
    {
        vector<int> mappings{15, 6, 19, 20, 28, 11, 27, 16, 0, 14, 22, 25,
4, 17, 30, 9, 1, 7, 23, 13, 31, 26, 2, 8, 18, 12, 29, 5, 21, 10, 3, 24};
        vector<bool> retVec;
        for (int i = 0; i < mappings.size(); i++)
        {
            retVec.push_back(vec[mappings[i]]);
        }
        return retVec;
    }

    vector<bool> DESFunctionHelper(vector<bool> R132)
    {
        vector<bool> expanded48 = expansionPBox(R132);
        vector<bool> XOR48 = XorVectors(expanded48, this->key48);
        vector<vector<bool>> sBoxInput = splitVector(XOR48, 8, 6);
        vector<bool> sBoxOutput = SBox(sBoxInput);
        vector<bool> straightPBoxOp = StraightPBox(sBoxOutput);
        return straightPBoxOp;
    }

    vector<vector<bool>> DESRoundFunc()
    {
        vector<vector<bool>> inputsSplitted64 = splitInputHelper();
        vector<bool> l132 = inputsSplitted64[0];
        vector<bool> r132 = inputsSplitted64[1];
        vector<bool> DESOp = DESFunctionHelper(r132);
        traceBoolVec(DESOp, "Output After DES Function");
        vector<bool> mixerOp = XorVectors(l132, DESOp);
        traceBoolVec(mixerOp, "Output After Mixer Function");
        vector<vector<bool>> finalOp;
        finalOp.push_back(r132);
        finalOp.push_back(mixerOp);
        return finalOp;
    }
};

int main()
{
```

```
    int tt;
    cin >> tt;
    while (tt--)
    {
        string hexMessageInput, key;
        cin >> hexMessageInput >> key;

        vector<bool> binInput = converHexToBinary(hexMessageInput);
        traceBoolVec(binInput, "Converted Binary Message");

        vector<bool> binInputKey = converHexToBinary(key);
        traceBoolVec(binInputKey, "Converted Binary Key");

        DESRound round(binInputKey, binInput);
        vector<vector<bool>> ans = round.DESRoundFunc();
        traceBoolVec(ans[0], "L-1 New");
        traceBoolVec(ans[1], "R-1 New");
    }
    return 0;
}
```

Outputs:

Input.txt

```
3
1111111111111111
111111111111

18CA18AD5A78E394
194CD072DE8C

308BEE9710AF9D37
84BB4473DCCC
```

Output.txt

```
Case #1
Converted Binary Message
0001000100010001000100010001000100010001000100010001000100010001
Converted Binary Key 0001000100010001000100010001000100010001000100010001
Output After DES Function 01111111101100000001000110111001
Output After Mixer Function 01101110101000010000000010101000
L-1 New 00010001000100010001000100010001
R-1 New 01101110101000010000000010101000

Case#2
Converted Binary Message
0001100011001010000110001010110101011010011110001110001110010100
Converted Binary Key 0001100101001100110100000111001011011110100011100
Output After DES Function 10010010110100011000010110100111
Output After Mixer Function 10001010000110111001110100001010
L-1 New 01011010011110001110001110010100
R-1 New 10001010000110111001110100001010

Case#3
Converted Binary Message
0011000010001011111011101001011100010000101011111001110100110111
Converted Binary Key 1000010010111011010001000111001111011100110011001100
Output After DES Function 00101111111101111000101001010010
Output After Mixer Function 00011111011111000110010011000101
L-1 New 00010000101011111001110100110111
R-1 New 00011111011111000110010011000101
```