

Name: Jatin Prajapati

Roll No: 21BCP452D

Semester: 6th

Division: 5th (G-10)

**#CNS (20CP320P)**

**PANDIT DEENDAYAL ENERGY UNIVERSITY**

**COMPUTER SCIENCE AND ENGINEERING DEPARTMENT**

Practical 1

* 1. Implementation of SDES algorithm in C language.

Keygen.h

*// CODE FOR SDES*

*#include* <stdio.h>

*#include* <stdlib.h>

*#include* <string.h>

*#include* <math.h>

*char* inputString[100], encryptedString[100], decryptedString[100];

*char* key[10], permutedKey[10];

*char* leftKey[5], rightKey[5];

*char* key1[10], key2[10];

*char* outputString[100];

*void* input()

{

    printf("Enter the input string: ");

    scanf("%s", inputString);

    printf("Enter the 10 bit key: ");

    scanf("%s", key);

}

*void* permuteKey()

{

*int* i;

*int* permutationTable[10] = {3, 5, 2, 7, 4, 10, 1, 9, 8, 6};

*for* (i = 0; i < 10; i++)

    {

        permutedKey[i] = key[permutationTable[i] - 1];

    }

}

*void* genereateLeftRightKey()

{

*int* i;

*for* (i = 0; i < 5; i++)

    {

        leftKey[i] = permutedKey[i];

        rightKey[i] = permutedKey[i + 5];

    }

}

*void* leftShiftLeftRightKey()

{

*int* i;

*char* tempLeftKey = leftKey[0];

*char* tempRightKey = rightKey[0];

*for* (i = 0; i < 4; i++)

    {

        leftKey[i] = leftKey[i + 1];

        rightKey[i] = rightKey[i + 1];

    }

    leftKey[4] = tempLeftKey;

    rightKey[4] = tempRightKey;

}

*void* generateKey1()

{

*int* i;

    leftShiftLeftRightKey();

*char* tempKey[10];

*for* (i = 0; i < 5; i++)

    {

        tempKey[i] = leftKey[i];

        tempKey[i + 5] = rightKey[i];

    }

*int* permutationTable[8] = {6, 3, 7, 4, 8, 5, 10, 9};

*for* (i = 0; i < 8; i++)

    {

        key1[i] = tempKey[permutationTable[i] - 1];

    }

}

*void* generateKey2()

{

*int* i;

    leftShiftLeftRightKey();

    leftShiftLeftRightKey();

*char* tempKey[10];

*for* (i = 0; i < 5; i++)

    {

        tempKey[i] = leftKey[i];

        tempKey[i + 5] = rightKey[i];

    }

*int* permutationTable[8] = {6, 3, 7, 4, 8, 5, 10, 9};

*for* (i = 0; i < 8; i++)

    {

        key2[i] = tempKey[permutationTable[i] - 1];

    }

}

*void* generateKeys()

{

    permuteKey();

    genereateLeftRightKey();

    generateKey1();

    generateKey2();

}

*char* encryption(*char* input) {

*int* i, output;

*char* outputChar;

*// Converting input to binary*

*int* inputBinary[8];

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

        inputBinary[i] = input % 2;

        input = input / 2;

    }

*// Initial Permutation*

*int* initialPermutationTable[8] = {2, 6, 3, 1, 4, 8, 5, 7};

*int* initialPermutation[8];

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

        initialPermutation[i] = inputBinary[initialPermutationTable[i] - 1];

    }

*// Splitting into left and right*

*int* left[4], right[4];

*for* (i = 0; i < 4; i++) {

        left[i] = initialPermutation[i];

        right[i] = initialPermutation[i + 4];

    }

*// Expansion Permutation*

*int* expansionPermutationTable[8] = {4, 1, 2, 3, 2, 3, 4, 1};

*int* expandedRight[8];

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

        expandedRight[i] = right[expansionPermutationTable[i] - 1];

    }

*// XOR with key1*

*int* xorWithKey1[8];

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

*if* (expandedRight[i] == key1[i]) {

            xorWithKey1[i] = 0;

        } *else* {

            xorWithKey1[i] = 1;

        }

    }

*// Splitting into left and right*

*int* leftXorWithKey1[4], rightXorWithKey1[4];

*for* (i = 0; i < 4; i++) {

        leftXorWithKey1[i] = xorWithKey1[i];

        rightXorWithKey1[i] = xorWithKey1[i + 4];

    }

*// S-Box 1*

*int* sBox1[4][4] = {

        {1, 0, 3, 2},

        {3, 2, 1, 0},

        {0, 2, 1, 3},

        {3, 1, 3, 2}

    };

*int* row1 = leftXorWithKey1[0] \* 2 + leftXorWithKey1[3] \* 1;

*int* column1 = leftXorWithKey1[1] \* 2 + leftXorWithKey1[2] \* 1;

*int* sBox1Output = sBox1[row1][column1];

*// S-Box 2*

*int* sBox2[4][4] = {

        {0, 1, 2, 3},

        {2, 0, 1, 3},

        {3, 0, 1, 0},

        {2, 1, 0, 3}

    };

*int* row2 = rightXorWithKey1[0] \* 2 + rightXorWithKey1[3] \* 1;

*int* column2 = rightXorWithKey1[1] \* 2 + rightXorWithKey1[2] \* 1;

*int* sBox2Output = sBox2[row2][column2];

*// S-Box Output*

*int* sBoxOutput[4];

*for* (i = 0; i < 4; i++) {

*if* (i == 0) {

            sBoxOutput[i] = sBox1Output / 2;

        } *else* *if* (i == 1) {

            sBoxOutput[i] = sBox1Output % 2;

        } *else* *if* (i == 2) {

            sBoxOutput[i] = sBox2Output / 2;

        } *else* {

            sBoxOutput[i] = sBox2Output % 2;

        }

    }

*// P4 Permutation*

*int* p4PermutationTable[4] = {2, 4, 3, 1};

*int* p4Permutation[4];

*for* (i = 0; i < 4; i++) {

        p4Permutation[i] = sBoxOutput[p4PermutationTable[i] - 1];

    }

*// XOR with left*

*int* xorWithLeft[4];

*for* (i = 0; i < 4; i++) {

*if* (p4Permutation[i] == left[i]) {

            xorWithLeft[i] = 0;

        } *else* {

            xorWithLeft[i] = 1;

        }

    }

*// combine right[] and xorWithLeft[]*

*int* combined[8];

*for* (i = 0; i < 4; i++) {

        combined[i] = right[i];

        combined[i + 4] = xorWithLeft[i];

    }

*// Break into left and right*

*int* leftCombined[4], rightCombined[4];

*for* (i = 0; i < 4; i++) {

        leftCombined[i] = combined[i];

        rightCombined[i] = combined[i + 4];

    }

*// Swap leftCombined and rightCombined*

*int* temp[4];

*for* (i = 0; i < 4; i++) {

        temp[i] = leftCombined[i];

        leftCombined[i] = rightCombined[i];

        rightCombined[i] = temp[i];

    }

*// Again do above step for key2*

*// Initial Permutation*

*int* initialPermutationTable2[8] = {2, 6, 3, 1, 4, 8, 5, 7};

*int* initialPermutation2[8];

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

        initialPermutation2[i] = combined[initialPermutationTable2[i] - 1];

    }

*// Splitting into left and right*

*int* left2[4], right2[4];

*for* (i = 0; i < 4; i++) {

        left2[i] = initialPermutation2[i];

        right2[i] = initialPermutation2[i + 4];

    }

*// Expansion Permutation*

*int* expansionPermutationTable2[8] = {4, 1, 2, 3, 2, 3, 4, 1};

*int* expandedRight2[8];

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

        expandedRight2[i] = right2[expansionPermutationTable2[i] - 1];

    }

*// XOR with key2*

*int* xorWithKey2[8];

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

*if* (expandedRight2[i] == key2[i]) {

            xorWithKey2[i] = 0;

        } *else* {

            xorWithKey2[i] = 1;

        }

    }

*// Splitting into left and right*

*int* leftXorWithKey2[4], rightXorWithKey2[4];

*for* (i = 0; i < 4; i++) {

        leftXorWithKey2[i] = xorWithKey2[i];

        rightXorWithKey2[i] = xorWithKey2[i + 4];

    }

*// S-Box 1*

*int* sBox1\_2[4][4] = {

        {1, 0, 3, 2},

        {3, 2, 1, 0},

        {0, 2, 1, 3},

        {3, 1, 3, 2}

    };

*int* row1\_2 = leftXorWithKey2[0] \* 2 + leftXorWithKey2[3] \* 1;

*int* column1\_2 = leftXorWithKey2[1] \* 2 + leftXorWithKey2[2] \* 1;

*int* sBox1Output\_2 = sBox1\_2[row1\_2][column1\_2];

*// S-Box 2*

*int* sBox2\_2[4][4] = {

        {0, 1, 2, 3},

        {2, 0, 1, 3},

        {3, 0, 1, 0},

        {2, 1, 0, 3}

    };

*int* row2\_2 = rightXorWithKey2[0] \* 2 + rightXorWithKey2[3] \* 1;

*int* column2\_2 = rightXorWithKey2[1] \* 2 + rightXorWithKey2[2] \* 1;

*int* sBox2Output\_2 = sBox2\_2[row2\_2][column2\_2];

*// S-Box Output*

*int* sBoxOutput\_2[4];

*for* (i = 0; i < 4; i++) {

*if* (i == 0) {

            sBoxOutput\_2[i] = sBox1Output\_2 / 2;

        } *else* *if* (i == 1) {

            sBoxOutput\_2[i] = sBox1Output\_2 % 2;

        } *else* *if* (i == 2) {

            sBoxOutput\_2[i] = sBox2Output\_2 / 2;

        } *else* {

            sBoxOutput\_2[i] = sBox2Output\_2 % 2;

        }

    }

*// P4 Permutation*

*int* p4PermutationTable2[4] = {2, 4, 3, 1};

*int* p4Permutation2[4];

*for* (i = 0; i < 4; i++) {

        p4Permutation2[i] = sBoxOutput\_2[p4PermutationTable2[i] - 1];

    }

*// XOR with left*

*int* xorWithLeft2[4];

*for* (i = 0; i < 4; i++) {

*if* (p4Permutation2[i] == left2[i]) {

            xorWithLeft2[i] = 0;

        } *else* {

            xorWithLeft2[i] = 1;

        }

    }

*// combine right[] and xorWithLeft[]*

*int* combined2[8];

*for* (i = 0; i < 4; i++) {

        combined2[i] = right2[i];

        combined2[i + 4] = xorWithLeft2[i];

    }

*// Break into left and right*

*int* leftCombined2[4], rightCombined2[4];

*for* (i = 0; i < 4; i++) {

        leftCombined2[i] = combined2[i];

        rightCombined2[i] = combined2[i + 4];

    }

*// Swap leftCombined and rightCombined*

*int* temp2[4];

*for* (i = 0; i < 4; i++) {

        temp2[i] = leftCombined2[i];

        leftCombined2[i] = rightCombined2[i];

        rightCombined2[i] = temp2[i];

    }

*// Combine leftCombined2 and rightCombined2*

*int* combined3[8];

*for* (i = 0; i < 4; i++) {

        combined3[i] = leftCombined2[i];

        combined3[i + 4] = rightCombined2[i];

    }

*// Inverse Initial Permutation*

*int* inverseInitialPermutationTable[8] = {4, 1, 3, 5, 7, 2, 8, 6};

*int* inverseInitialPermutation[8];

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

        inverseInitialPermutation[i] = combined3[inverseInitialPermutationTable[i] - 1];

    }

*// Converting binary to decimal*

    output = 0;

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

        output = output + inverseInitialPermutation[i] \* pow(2, i);

    }

    output = output % 26 + 65;

*// Converting decimal to character*

    outputChar = (*char*)output;

*return* outputChar;

}

*void* encrypt() {

*int* i;

*char* encryptedChar;

*for* (i = 0; inputString[i] != '\0'; i++) {

        encryptedChar = encryption(inputString[i]);

        encryptedString[i] = encryptedChar;

    }

    encryptedString[i] = '\0';

    printf("Encrypted string: %s\n", encryptedString);

}

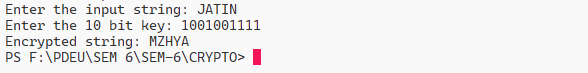
*void* main(){

    input();

    generateKeys();

    encrypt();

}

OUTPUT