**Dharmsinh Desai University, Nadiad**

**Department of Information Technology**

**DAIE, IT704**

**B.Tech. IT, Sem: VII**

**Submitted By**

**Roll No:** IT076

**Name:** Dishant Modh

# Experiment 4: Write a C/C++/Java program to implement S-DES algorithm for data encryption along with key generation of S-DES.

# Code

# #include <stdio.h>

# #include <stdlib.h>

# #include <string.h>

# void leftShift(char \*sh)

# {

# char t1 = sh[0], t2 = sh[5];

# for (int i = 0; i < 5; i++)

# {

# sh[i] = sh[(i + 1) % 5];

# sh[i + 5] = sh[(i + 1) % 5 + 5];

# }

# sh[4] = t1;

# sh[9] = t2;

# }

# void permutation(char \*src, char \*dest, int \*per, int len)

# {

# int i;

# for (i = 0; i < len; i++)

# dest[i] = src[per[i] - 1];

# dest[i] = 0;

# }

# void generateKey(char \*key, char \*k1, char \*k2, int \*p10, int \*p8)

# {

# char tmp[11];

# int i;

# printf("\tKey Generation Algorithm\n");

# printf("\t\tInitial 10 bit Key: %s\n", key);

# permutation(key, tmp, p10, 10);

# printf("\t\tP10 : %s\n", tmp);

# leftShift(tmp);

# printf("\t\tSH1 : %s\n", tmp);

# permutation(tmp, k1, p8, 8);

# printf("\t\tKey1 : %s\n", k1);

# leftShift(tmp);

# leftShift(tmp);

# printf("\t\tSH2 : %s\n", tmp);

# permutation(tmp, k2, p8, 8);

# printf("\t\tKey2 : %s\n", k2);

# }

# void generateIpPrime(int \*ip, int \*ipprime)

# {

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

# ipprime[ip[i] - 1] = i + 1;

# }

# void scanInput(int \*arr, int size)

# {

# for (int i = 0; i < size; i++)

# scanf("%d", &arr[i]);

# }

# void scanSubMatrix(char \*\*\*mat)

# {

# for (int i = 0; i < 4; i++)

# {

# for (int j = 0; j < 4; j++)

# {

# mat[i][j] = (char \*)calloc(3, sizeof(char));

# scanf("%s", mat[i][j]);

# }

# }

# }

# void xorOperation(char \*a, char \*b, int len)

# {

# for (int i = 0; i < len; i++)

# {

# if (a[i] != b[i])

# a[i] = '1';

# else

# a[i] = '0';

# }

# }

# void complexFunction(char \*src, int \*ep, char \*key, char \*\*\*s0, char \*\*\*s1, int \*p4)

# {

# char expansion[9], substitution[5];

# int si, sj;

# permutation(src + 4, expansion, ep, 8);

# printf("\t\t\tE/P right part: %s\n", expansion);

# xorOperation(expansion, key, 8);

# printf("\t\t\tE/P xor Key: %s\n", expansion);

# si = 2 \* (expansion[0] == '1') + (expansion[3] == '1');

# sj = 2 \* (expansion[1] == '1') + (expansion[2] == '1');

# strncpy(substitution, s0[si][sj], 2);

# si = 2 \* (expansion[4] == '1') + (expansion[7] == '1');

# sj = 2 \* (expansion[5] == '1') + (expansion[6] == '1');

# strncpy(substitution + 2, s1[si][sj], 2);

# printf("\t\t\tSubstitution sequence: %s\n", substitution);

# strcpy(expansion, substitution);

# expansion[4] = 0;

# permutation(expansion, substitution, p4, 4);

# printf("\t\t\tP4: %s\n", substitution);

# xorOperation(src, substitution, 4);

# printf("\t\t\tP4 xor Left Part, Right Part: %s\n", src);

# }

# void swap(char \*arr)

# {

# char tmp;

# for (int i = 0; i < 4; i++)

# {

# tmp = arr[i];

# arr[i] = arr[i + 4];

# arr[i + 4] = tmp;

# }

# }

# void encryption(char \*msg, char \*enc, char \*\*\*s0, char \*\*\*s1, int \*ep, int \*ip, int \*p4, int \*ipprime, char \*k1, char \*k2)

# {

# char tmp[9];

# printf("\tEncryption Algorithm\n");

# permutation(msg, tmp, ip, 8);

# printf("\t\tInitial Permutation: %s\n", tmp);

# printf("\t\tEncryption using Key1:\n");

# complexFunction(tmp, ep, k1, s0, s1, p4);

# swap(tmp);

# printf("\t\tSwapping of Left and Right Part: %s\n", tmp);

# printf("\t\tEncryption using Key2:\n");

# complexFunction(tmp, ep, k2, s0, s1, p4);

# permutation(tmp, enc, ipprime, 8);

# printf("\t\tReverse of Initial Permutation: %s\n", enc);

# }

# void decryption(char \*enc, char \*msg, char \*\*\*s0, char \*\*\*s1, int \*ep, int \*ip, int \*p4, int \*ipprime, char \*k1, char \*k2)

# {

# char tmp[9];

# printf("\tDecryption Algorithm\n");

# permutation(enc, tmp, ip, 8);

# printf("\t\tInitial Permutation: %s\n", tmp);

# printf("\t\tDecryption using Key2:\n");

# complexFunction(tmp, ep, k2, s0, s1, p4);

# swap(tmp);

# printf("\t\tSwapping of Left and Right Part: %s\n", tmp);

# printf("\t\tDecryption using Key1:\n");

# complexFunction(tmp, ep, k1, s0, s1, p4);

# permutation(tmp, msg, ipprime, 8);

# printf("\t\tReverse of Initial Permutation: %s\n", msg);

# }

# int main()

# {

# char message[9], encrypt[9], key[11], key1[11], key2[11];

# int p10[10], p8[8], p4[4], ep[8], i, ip[8], ipprime[8], choice;

# char \*\*\*s0, \*\*\*s1;

# s0 = (char \*\*\*)calloc(4, sizeof(char \*\*));

# s1 = (char \*\*\*)calloc(4, sizeof(char \*\*));

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

# {

# s0[i] = (char \*\*)calloc(4, sizeof(char \*));

# s1[i] = (char \*\*)calloc(4, sizeof(char \*));

# }

# printf("Enter key value(10 bit): ");

# scanf("%s", key);

# printf("Enter P10 permutation sequence: ");

# scanInput(p10, 10);

# printf("Enter P8 permutation sequence: ");

# scanInput(p8, 8);

# printf("Enter P4 permutation sequence: ");

# scanInput(p4, 4);

# printf("Enter IP sequence: ");

# scanInput(ip, 8);

# printf("Enter expnsion permutation sequence: ");

# scanInput(ep, 8);

# printf("Enter first subtitution matrix(row vise in 2bit binary): \n");

# scanSubMatrix(s0);

# printf("Enter second subtitution matrix(row vise in 2bit binary): \n");

# scanSubMatrix(s1);

# generateIpPrime(ip, ipprime);

# printf("\tIP inverse: ");

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

# printf("%d ", ipprime[i]);

# printf("\n");

# generateKey(key, key1, key2, p10, p8);

# while (1)

# {

# printf("1. Encrypt message\n2. Decrypt message\n3. Exit\nEnter your choice: ");

# scanf("%d", &choice);

# if (choice == 3)

# break;

# else if (choice == 1)

# {

# printf("Enter 8 bit initial message to encrypt: ");

# scanf("%s", message);

# encryption(message, encrypt, s0, s1, ep, ip, p4, ipprime, key1, key2);

# printf("S-DES encrypted message: %s\n", encrypt);

# printf("\n\n");

# }

# else if (choice == 2)

# {

# printf("Enter 8 bit S-DES encrypted message to encrypt: ");

# scanf("%s", encrypt);

# decryption(encrypt, message, s0, s1, ep, ip, p4, ipprime, key1, key2);

# printf("Initial Message: %s\n", message);

# printf("\n\n");

# }

# else

# {

# printf("\nEnter Valid Choice.\n");

# }

# }

# printf("\n\n");

# return 0;

# }

1. **Output**

