Assignment 1 of ELEC 473

This is the C code of DES.

1. Run the DES code to encrypt your family name using your student ID as the secret key.

Student # : 20119884 Hex: 32 30 31 31 39 38 38 34 0a Bin: 00110010 00110000 00110001 00110001 00111001 00111000 00111000 00110100

Family Name: Li Hex: 4c 69 0a Bin: 01001100 01101001 00001010

2. Annotate the code of the function **uint64\_t des()**.

|  |
| --- |
|  |

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>

#define LB32\_MASK 0x00000001

#define LB64\_MASK 0x0000000000000001

#define L64\_MASK 0x00000000ffffffff

#define H64\_MASK 0xffffffff00000000

/\* Initial Permutation Table \*/

static char IP[] = {

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

};

/\* Inverse Initial Permutation Table \*/

static char PI[] = {

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

};

/\*Expansion table \*/

static char E[] = {

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

};

/\* Post S-Box permutation \*/

static char P[] = {

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

};

/\* The S-Box tables \*/

static char S[8][64] = { {

/\* S1 \*/

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

},{

/\* S2 \*/

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

},{

/\* S3 \*/

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

},{

/\* S4 \*/

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

},{

/\* S5 \*/

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

},{

/\* S6 \*/

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

},{

/\* S7 \*/

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

},{

/\* S8 \*/

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

} };

/\* Permuted Choice 1 Table \*/

static char PC1[] = {

57, 49, 41, 33, 25, 17, 9,

1, 58, 50, 42, 34, 26, 18,

10, 2, 59, 51, 43, 35, 27,

19, 11, 3, 60, 52, 44, 36,

63, 55, 47, 39, 31, 23, 15,

7, 62, 54, 46, 38, 30, 22,

14, 6, 61, 53, 45, 37, 29,

21, 13, 5, 28, 20, 12, 4

};

/\* Permuted Choice 2 Table \*/

static char PC2[] = {

14, 17, 11, 24, 1, 5,

3, 28, 15, 6, 21, 10,

23, 19, 12, 4, 26, 8,

16, 7, 27, 20, 13, 2,

41, 52, 31, 37, 47, 55,

30, 40, 51, 45, 33, 48,

44, 49, 39, 56, 34, 53,

46, 42, 50, 36, 29, 32

};

/\* Iteration Shift Array \*/

static char iteration\_shift[] = {

/\* 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 \*/

1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1

};

/\*

\* The DES function

\* input: 64 bit message

\* key: 64 bit key for encryption/decryption

\* mode: 'e' = encryption; 'd' = decryption

\*/

uint64\_t des(uint64\_t input, uint64\_t key, char mode) {

int i, j;

/\* 8 bits \*/

char row, column;

/\* 28 bits \*/

uint32\_t C = 0;

uint32\_t D = 0;

/\* 32 bits \*/

uint32\_t L = 0;

uint32\_t R = 0;

uint32\_t s\_output = 0;

uint32\_t f\_function\_res = 0;

uint32\_t temp = 0;

/\* 48 bits \*/

uint64\_t sub\_key[16] = { 0 };

uint64\_t s\_input = 0;

/\* 56 bits \*/

uint64\_t permuted\_choice\_1 = 0;

uint64\_t permuted\_choice\_2 = 0;

/\* 64 bits \*/

uint64\_t init\_perm\_res = 0;

uint64\_t inv\_init\_perm\_res = 0;

uint64\_t pre\_output = 0;

// the 64 bit plain text block is handed over to an initial Permutation (IP) function,

// and The initial permutation performed on plain text.

for (i = 0; i < 64; i++) {

//init\_perm\_res is the new varibal to save the text that are permutated

//each time it inputs a new digit, it should be shift left because the lowest bit is used to save new bit.

init\_perm\_res <<= 1;

//the pricinple of (input >> (64 - IP[i])) is let the replaced bit shift

//to the lowest bit and save to the lowest bit of the init\_perm\_res

init\_perm\_res |= (input >> (64 - IP[i]))& LB64\_MASK;

}

//seperate the block into two blocks

//shift right the higher 32 bits to the lower 32 bits, and do and operation with L64\_MASK to get the left half block

L = (uint32\_t)(init\_perm\_res >> 32) & L64\_MASK;

//get lower 32 bits, then do and operation with L64\_MASK to get the right half block

R = (uint32\_t)init\_perm\_res & L64\_MASK;

//Key transformation: initial 64-bit key is transformed into a 56-bit key by discarding every

//8th bit of the initial key. Thus, for each a 56-bit key is available.

for (i = 0; i < 56; i++) {

//use the same way that talked about in IP to permutate the key

//the difference between IP and PC is PC will throw away the last 8 bit, which means just use 56 bits.

permuted\_choice\_1 <<= 1;

permuted\_choice\_1 |= (key >> (64 - PC1[i]))& LB64\_MASK;

}

//this 56 bits key is divided into two halves, each of 28 bits.

//shift right the higher 28 bits to the lower 28 bits, and do and operation with MASK to get higher 28 bits key

C = (uint32\_t)((permuted\_choice\_1 >> 28) & 0x000000000fffffff);

//get lower 28 bits, then do and operation with MASK to get lower 28 bits key

D = (uint32\_t)(permuted\_choice\_1 & 0x000000000fffffff);

//These halves are circularly shifted left by one or two positions, depending on the round.

for (i = 0; i < 16; i++) {

//This for loop control the round, we have 16 rounds

for (j = 0; j < iteration\_shift[i]; j++) {

//This for loop control the number of shift bits

//ex. if iteration\_shift[i] = 2, this loop will be looped 2 times, so C or D will be shifted 2 bits

C = 0x0fffffff & (C << 1) | 0x00000001 & (C >> 27);

D = 0x0fffffff & (D << 1) | 0x00000001 & (D >> 27);

}

//merge two 28 bits key to a 56 bits key

//shift C to the higher 28 bits, then put D in the lower 28 bits

permuted\_choice\_2 = 0;

permuted\_choice\_2 = (((uint64\_t)C) << 28) | (uint64\_t)D;

// initialize the varibal used in second permuted\_choice(PC2)

//sub\_key[i] is used to save the keys that have been permutated

sub\_key[i] = 0;

//The same way as aboves to permutate the key and just choose first 48 bits

for (j = 0; j < 48; j++) {

sub\_key[i] <<= 1;

sub\_key[i] |= (permuted\_choice\_2 >> (56 - PC2[j]))& LB64\_MASK;

}

}

for (i = 0; i < 16; i++) {

s\_input = 0;

// each sub-block is 32 bits

// in each round, we need to use expansion table to expand the size of the blocks from 32 bits to 48 bits.

for (j = 0; j < 48; j++) {

s\_input <<= 1;

//R is (uint32\_t)init\_perm\_res & L64\_MASK, which are the permutated text.

//The expansion table will be used to expand the size of R from 32 bits to 48 bits.

//The principle of expansion table is reuse some bits and expand the text

s\_input |= (uint64\_t)((R >> (32 - E[j]))& LB32\_MASK);

}

//if mode is decryption, we need to XOR text and key from round 16,

//so the subkey should be start at sub\_key[15]

if (mode == 'd') {

s\_input = s\_input ^ sub\_key[15 - i];

}

else {

//if mode is encryption, we need to XOR text and key from round 1

s\_input = s\_input ^ sub\_key[i];

}

// this loop is used to do the S box

//each S box will compress 6 bits to 4 bits

for (j = 0; j < 8; j++) {

//get the 6 bits that should input to the S box.

//ex. The first 6 bits of the text should be in the S box 1

row = (char)((s\_input & (0x0000840000000000 >> 6 \* j)) >> 42 - 6 \* j);

//shift right 4 bits of row, which can get the higher 2 bits of the row

// row is the outer bits

// row range is from 00 to 11

row = (row >> 4) | row & 0x01;

// column is the Middle 4 bits of input

// column range should be from 0000 to 1111

column = (char)((s\_input & (0x0000780000000000 >> 6 \* j)) >> 43 - 6 \* j);

// each time s\_output should shift left 4 bits before it saves new 4 bits

s\_output <<= 4;

//(uint32\_t)(S[j][16 \* row + column] & 0x0f) is the S-box output,

//j is the box number,16 \* row + column is the place in the S box where has the compressed bits of chosen 6 bits

s\_output |= (uint32\_t)(S[j][16 \* row + column] & 0x0f);

}

f\_function\_res = 0;

//use Post S-Box permutation to do the permutation

for (j = 0; j < 32; j++) {

// the same way to another permutation and save the new text

f\_function\_res <<= 1;

f\_function\_res |= (s\_output >> (32 - P[j]))& LB32\_MASK;

}

//The result of the Post S-Box permutation transposition is xor with the original 64-bit grouping of the left half block

temp = R;

R = L ^ f\_function\_res;

L = temp;

}

// the left and right halves are swapped, and another round begins

pre\_output = (((uint64\_t)R) << 32) | (uint64\_t)L;

// do the inverse IP

for (i = 0; i < 64; i++) {

// the same way to another permutation and save the new text

inv\_init\_perm\_res <<= 1;

inv\_init\_perm\_res |= (pre\_output >> (64 - PI[i]))& LB64\_MASK;

}

return inv\_init\_perm\_res;

}

int main(int argc, const char\* argv[]) {

int i;

uint64\_t input = 0x4C690A0000000000;

//uint64\_t key = 0x0000000000000000;

uint64\_t key = 0x3230313139383834;

uint64\_t result = input;

for (i = 0; i < 16; i++) {

if (i % 2 == 0) {

result = des(result, result, 'e');

printf("E: %016llx\n", result);

}

else {

result = des(result, result, 'd');

printf("D: %016llx\n", result);

}

}

result = des(input, key, 'e');

printf ("E: %016llx\n", result);

result = des(result, key, 'd');

printf ("D: %016llx\n", result);

exit(0);

}