[R E P O R T]

정보통신공학전공

200301582

김성태

암호 알고리즘 DES 구현 과제 보고서

1. 과제의 목적

① DES 알고리즘을 이해한다.

② DES 암호 알고리즘을 프로그래밍 언어로 구현한다.

2. 필요 환경

① C 언어 컴파일러

3. 과제 수행 내역

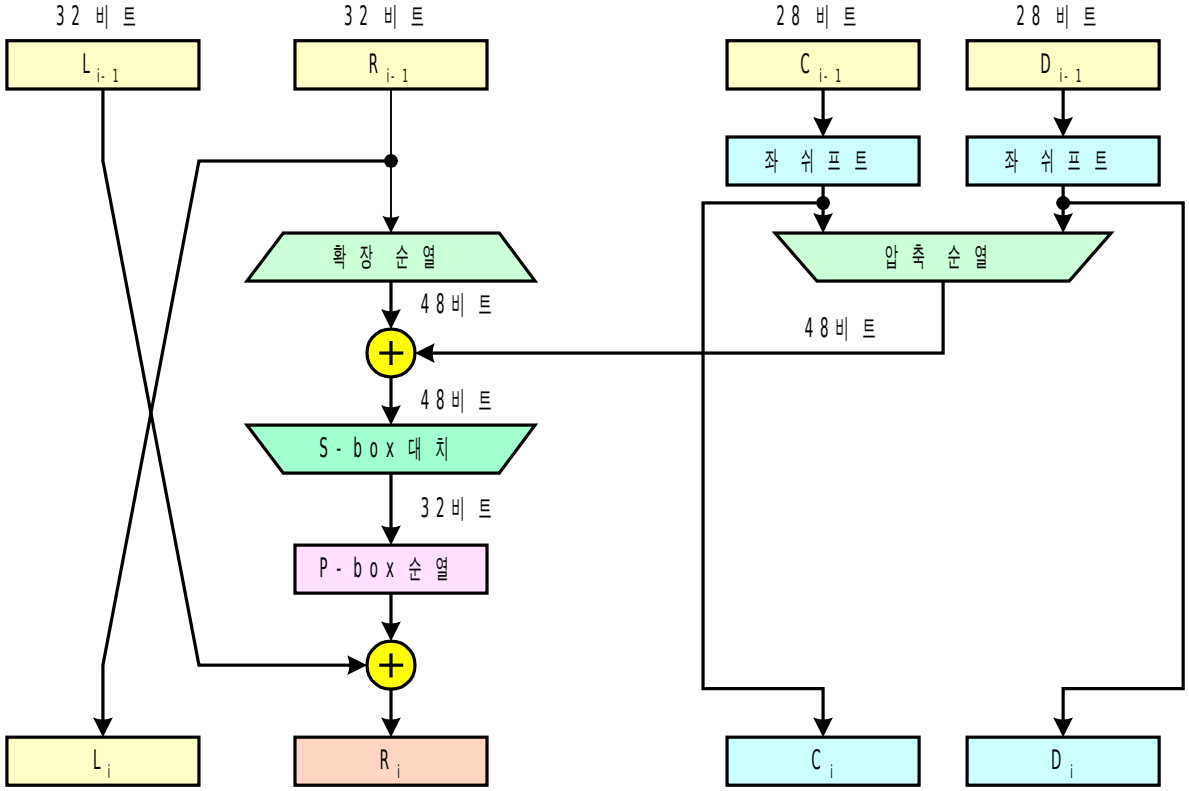
① DES 암호 알고리즘 조사

② DES 암호 알고리즘 구현

4. 과제 수행

① DES 알고리즘의 이해

전체적인 DES 알고리즘의 구성도



문장으로서의관점

① 64비트의 배열을 반씩 나눈다.
② 나뉘어진 배열중 오른쪽의 배열을 다음번 차례의 왼쪽 배열에 복사한다.
③ 오른쪽 배열 32비트를 확장배열하여 48비트로 변경한다.
④ 확장된 오른쪽 배열을 해당 키열과 XOR 연산한다.
⑤ XOR 연산된 오른쪽 배열을 32비트로 변경한다.
⑥ 변경된 32비트 배열을 P-box 를 이용, 섞는다. (shuffle!!)
⑦ Suffle 된 오른쪽 배열 32비트를 왼쪽 배열과 XOR 연산한다.
⑧ 이와 같은 순서를 16번 반복한다.

key 로서의 관점

① 입력된 키를 순열을 이용하여 28비트 두 배열로 나눈다.
② 각각의 배열을 해당 차수에 맞는 count에 따라 왼쪽으로 로테이션 연산한다.
③ 로테이션 연산된 키 배열(왼쪽, 오른쪽)을 합친 후, 48비트로 압축 연산한다.
④ 연산된 48비트는 해당 차수의 키값이 된다.
⑤ 압축되기전 키배열은 다음 차수의 입력 키값으로 사용된다.

② DES 암호 알고리즘 구현

encryt.h

```
/*************************************************************************
 *        encrypt.h
 *
 * Sun Oct 14 14:06:46 2007
 * Copyright  2007  pchero
 * pchero21@gmail.com
 *************************************************************************/

/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
```

```
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 */

#ifndef ENCRYPT_H
#define ENCRYPT_H

typedef unsigned long Huge;

void des_encipher(unsigned char *plaintext, unsigned char *ciphertext, unsigned char *key);

void des_decipher(unsigned char *ciphertext, unsigned char *plaintext, unsigned char *key);

#endif
```

bit.h

```
/*************************************************************************
 *         bit.h
 *
 *  Sun Oct 14 14:02:13 2007
 *  Copyright  2007  pchero
 *  pchero21@gmail.com
 *************************************************************************/

/*
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 */
```

```
int bit_get(const unsigned char *bits, int pos);

void bit_set(unsigned char *bits, int pos, int state);

void bit_rot_left(unsigned char *bits, int size, int count);

void bit_xor(const unsigned char *bits1, const unsigned char *bits2, unsigned char *bitsx, int
size);
```

bit.c

```
/*************************************************************************
 *         bit.c
 *
 *  Sun Oct 14 13:35:46 2007
 *  Copyright  2007  pchero
 *  pchero21@gmail.com
 *************************************************************************/

/*
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 */

#include <string.h>

#include "bit.h"

// bit_get
// 해당 순번의 비트열의 값을 리턴한다.
```

```
// bits : 비트값을 찾기를 원하는 문자열
// pos : 찾기를 원하는 위치
int bit_get(const unsigned char *bits, int pos)
{
	unsigned char mask;
	int i;

	// 얻을 비트의 마스크 세트하기.
	mask = 0x80;

	for(i = 0; i < (pos % 8); i++)
		mask = mask >> 1;

	// 비트 얻기
	return (((mask & bits[(int)(pos / 8)]) == mask) ? 1 : 0);
}

// bit_set
// 해당 순번의 비트값을 세트 한다.
// bits : 세팅을 원하는 문자열
// pos : 세팅을 원하는 위치
// state : 세팅을 원하는 값
void bit_set(unsigned char *bits, int pos, int state)
{
	unsigned char mask;

	int i;

	// 세트할 비트의 마스크 세트하기.
	mask = 0x80;

	for(i = 0; i < (pos % 8); i++)
		mask = mask >> 1;

	// 비트 세트하기.
	if(state)
		bits[pos / 8] = bits[pos / 8] | mask;
	else
		bits[pos / 8] = bits[pos / 8] & (~mask);
```

```
	return;
}

// bit_xor
// 두 문자열을 서로 xor 연산한다.
// bits1 : xor 연산을 원하는 문자열1
// bits2 : xor 연산을 원하는 문자열2
// bitsx : 연산되어 반환되어지는 문자열
// size : 연산을 원하는 size(갯수)
void bit_xor(const unsigned char *bits1, const unsigned char *bits2, unsigned char *bitsx, int
size)
{
	int i;

	// 두 버퍼의 비트 xor 계산.
	for(i = 0; i < size; i++) {
		if(bit_get(bits1, i) != bit_get(bits2, i))
			bit_set(bitsx, i, 1);
		else
			bit_set(bitsx, i, 0);
	}

	return;
}



// bit_rot_left
// 해당 문자열을 왼쪽으로 count번 로테이션시킨다.
// bits : shift 원하는 문자열
// size : shift 를 원하는 size(갯수)
// count : 로테이션 횟수.
void bit_rot_left(unsigned char *bits, int size, int count)
{
	int fbit, lbit, i, j;

	// 지정된 비트의 수만큼 버퍼를 왼쪽으로 회전.
	if(size > 0) {
		for(j = 0; j < count; j++) {
			for(i = 0; i <= ((size - 1) / 8); i++) {
				// 현재 바이트에서 왼쪽으로 시프트 오프될 비트 얻기.
```

```
                                lbit = bit_get(&bits[i], 0);

                                if(i == 0) {
                                        // 나중을 위해 첫 바이트에서 시프트 오프되는 비트 저장.
                                        fbit = lbit;
                                } else {
                                        // 이전 바이트의 제일 오른쪽 비트를 현재 바이트에서 시프트 오프
될 비트로 세트.
                                        bit_set(&bits[i - 1], 7, lbit);
                                }
                                // 현재 바이트를 왼쪽으로 시프트.
                                bits[i] = bits[i] << 1;
                        }
                        // 버퍼의 제일 오른쪽 비트를 첫 바이트에서 시프트 오프된 비트로 세트.
                        bit_set(bits, size - 1, fbit);
                }
        }
        return;
}
```

des.c

```
/*************************************************************************
 *         des.c
 *
 *  Sun Oct 14 14:10:00 2007
 *  Copyright  2007  pchero
 *  pchero21@gmail.com
 *************************************************************************/

/*
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
```

```
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 */

#include <math.h>
#include <stdlib.h>
#include <string.h>

#include "bit.h"
#include "encrypt.h"



// 키 변환을 위한 매칭 정의.
// 64비트의 초기 키 값을 56비트로 변환시 index값으로 쓰인다.
static const int DesTransform[56] = {
        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
};

// 부속키 계산을 위한 회전 횟수 정의
// 각 라운드에 몇번의 shift를 하는지를 정의한다.
static const int DesRotations[16] = {
        1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
};

// 부속키 순열 선택을 위한 매칭 정의
// shft된 부속키 순열을 매칭할 때 쓰인다.
static const int DesPermuted[48] = {
        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,
};

// 자료 블록의 확장 순열을 위한 매핑 정의.
// 초기비트[64비트]의 순열을 48비트로 줄이는데 사용한다.
```

```
static const int DesExpansion[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
};

// 자료 블록에 대해 수행되는 S-상자 대입을 위한 표 정의.
// 각각의 라운드 S-box에 대입된다.
static const int DesSbox[8][4][16] = {
{
        // 1라운드
        {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}
},

{
        // 2라운드
        {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}
},

{
        // 3라운드
        {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},
},

{
        // 4라운드
        { 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}
```

```
},

{
	// 5라운드
	{ 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}
},

{
	// 6라운드
	{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}
},

{
	// 7라운드
	{ 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,  2,  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}
},

{
	// 8라운드
	{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}
}

};

// 자료 블록의 p - box 순열을 위한 매핑 정의.
// shuffle 을 위한 p - box.
static const int DesPbox[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
};

// 자료의 암호화와 해독을 구별하는 형 정의.
typedef enum DesEorD_ {encipher, decipher} DesEorD;

// permute
// 순열을 갯수를 늘이거나 줄일 때 사용.
// bits : 대상 순열
// mapping : match 되는 순열
// n : matching 을 원하는 갯수
static void permute(unsigned char *bits, const int *mapping, int n)
{
        unsigned char temp[8];
        int i;

        // n개 항목의 매핑을 사용해서 버퍼 순열 만들기.
        memset(temp, 0, (int)ceil(n / 8));

        for(i = 0; i < n; i++)
                bit_set(temp, i, bit_get(bits, mapping[i] - 1));

        memcpy(bits, temp, (int)ceil(n / 8));

        return;
}

// des_main
// 자료를 암호화 하거나 복호화하는 모듈.
// des 의 핵심이 된다.
// source : 원문 or 암호문
// target : 복호화되거나 암호화되는 문장
// key : 복호화 or 암호화 하는데 사용되는 key
// direction : 복호화 or 암호화를 선택.
static int des_main(const unsigned char *source, unsigned char *target, const unsigned char
*key, DesEorD direction)
{
        static unsigned char subkeys[16][7];

        unsigned char temp[8], lkey[4], rkey[4], lblk[6], rblk[6], fblk[6], xblk[6], sblk;
```

```
int row, col, i, j, k, p;

// 키의 복사본 만들기.
memcpy(temp, key, 8);

// 키의 순열을 만들고 56비트로 압축.
permute(temp, DesTransform, 56);

// 키를 두 28비트 블록으로 나눔.
memset(lkey, 0, 4);
memset(rkey, 0, 4);

for(j = 0; j < 28; j++)
	bit_set(lkey, j, bit_get(temp, j));

for(j = 0; j < 28; j++)
	bit_set(rkey, j, bit_get(temp, j + 28));

// 각 라운드에 대한 부속키들 계산
for(i = 0; i < 16; i++) {
	// 해당 라운드에 따라 각 블록 회전.
	bit_rot_left(lkey, 28, DesRotations[i]);
	bit_rot_left(rkey, 28, DesRotations[i]);

	// 블록들을 하나의 부속키로 합침.
	for(j = 0; j < 28; j++)
		bit_set(subkeys[i], j, bit_get(lkey, j));

	for(j = 0; j < 28; j ++)
		bit_set(subkeys[i], j + 28, bit_get(rkey, j));

	// 순열 선택 순열 만들기.
	permute(subkeys[i], DesPermuted, 48);
} // for(i = 0; i < 16; i++)

// 원문의 복사본 만들기.
memcpy(temp, source, 8);

// 원문을 32비트의 왼쪽, 오른쪽 블록으로 나눔.
```

```
memcpy(lblk, &temp[0], 4);
memcpy(rblk, &temp[4], 4);

// 원문을 암호화하거나 해독함.
for(i = 0; i < 16; i++) {
	// f 의 계산 시작.
	memcpy(fblk, rblk, 4);

	// 오른쪽 블록의 복사본의 순열을 만들고 48비트로 확장.
	permute(fblk, DesExpansion, 48);

	// 라운드에 적절한 부속키 적용.
	if(direction == encipher) {
		// 암호화의 경우 부속키들이 오름차순으로 적용됨.
		bit_xor(fblk, subkeys[i], xblk, 48);
		memcpy(fblk, xblk, 6);
	} else {
		// 해독의 경우 부속키들이 내림차순으로 적용됨.
		bit_xor(fblk, subkeys[15 - i], xblk, 48);
		memcpy(fblk, xblk, 6);
	}

	// s - box 대입 수행.
	p = 0;
	for(j = 0; j < 8; j++) {
		// s - box 표에서 행과 열 계산.
		row = (bit_get(fblk, (j * 6) + 0) * 2) + (bit_get(fblk, (j * 6) + 5) * 1);
		col = (bit_get(fblk, (j * 6) + 1) * 8) + (bit_get(fblk, (j * 6) + 2) * 4) +
			(bit_get(fblk, (j * 6) + 3) * 2) + (bit_get(fblk, (j * 6) + 4) * 1);

		// 현재 6비트 블록에 대해 s - box 대입 수행.
		sblk = (unsigned char)DesSbox[j][row][col];

		for(k = 4; k < 8; k++) {
			bit_set(fblk, p, bit_get(&sblk, k));
			p++;
		}
	} // for(j = 0; j < 8; j++) s - box 대입.

	// f를 완료하기 위해 p - box 순열 수행.
```

```
                permute(fblk, DesPbox, 32);

                // 왼쪽 블록과 f의 XOR 계산.
                bit_xor(lblk, fblk, xblk, 32);

                // 라운드의 왼쪽 블록 세트하기.
                memcpy(lblk, rblk, 4);

                // 라운드의 오른쪽 블록 세트하기.
                memcpy(rblk, xblk, 4);
        } // for(i = 0; i < 16; i++) 암호화 or 복호화.

        // target 에 합쳐진 최종 오른쪽과 왼쪽 블록들 세트하기.
        memcpy(&target[0], rblk, 4);
        memcpy(&target[4], lblk, 4);

        return 0;
}

// des_encipher
// 자료 암호화를 위한 인터페이스
// 실제 사용자는 이부분만 사용하면 된다.
// plaintext : 암호화를 원하는 원문
// ciphertext :  암호화되어 리턴되는 문장
// key : 사용되는 key값
void des_encipher(unsigned char *plaintext, unsigned char *ciphertext, unsigned char *key)
{
        des_main(plaintext, ciphertext, key, encipher);
        return;
}

// des_decipher
// 자료 복호화를 위한 인터페이스
// 실제 사용자는 이 부분만 사용하면 된다.
// ciphertext : 복호화를 원하는 암호문
// plaintext : 복호화되는 원문
// key : 사용되는 key값
void des_decipher(unsigned char *ciphertext, unsigned char *plaintext, unsigned char *key)
{
        des_main(ciphertext, plaintext, key, decipher);
```

```
	return;
}
```

main.c

```
/* Created by Anjuta version 1.2.4a */
/*	This file will not be overwritten */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "bit.h"
#include "encrypt.h"


int main(int argc, char **argv)
{

	if(argc != 3) {
		printf("Usage : %s <text> <key>\n", argv[0]);
		return 0;
	}

	unsigned char text[9];
	unsigned char ciphertext[9];
	unsigned char key[8];
	unsigned char test[9];

	strcpy(text, argv[1]);
	strcpy(key, argv[2]);

	// 암호화
	des_encipher(text, ciphertext, key);
	ciphertext[9] = '\0';
	printf("%s\n", ciphertext);

	// 복호화
	des_decipher(ciphertext, test, key);
	test[9] = '\0';
```

```
        printf("%s\n", test);

        return 0;
}
```

comfile option

```
gcc -o des -g -I./ -lm des.c bit.c main.c
```

실행 화면

5. 느낀점

상당부분...책을 참고했습니다.

비트 연산부분은 모르는 부분이 대부분이어서 인터넷을 참고했으나, 책에서의 설명이 더 알기 쉬웠습니다.

알고리즘을 구현하면서...이 알고리즘을 네트워크에서 접목을 시키면 어떨까...하고 생각을 했으나...DES 알고리즘의 경우, 어플리케이션 단계에서 구현되는 암호화이기 때문에 네트워크에서의 접목은 상당히 어렵다고 결론을 짓게 되었습니다.

컴파일은 gcc 를 이용하여 하였으며, gcc 버전은 4.1.2 버전입니다.

6. 참고

① C로 구현한 알고리즘. 카일 루든. O'REILLY

② http://kldp.org 한국 리눅스 문서 프로젝트