

## 과제 4 보고서

지능기전공학부 스마트기기공학전공

18011789

조혜수

### 목차

1. 실행 결과
  - A. DFT
    - i. 블록 크기  $N = 8$
    - ii. 블록 크기  $N = 512$ , low pass filter 적용 안함
    - iii. 블록 크기  $N = 512$ , low pass filter 적용 함
  - B. DCT
    - i. 블록 크기  $N = 8$
    - ii. 블록 크기  $N = 512$
2. 코드 ( 주석 포함 )
  - A. DFT
  - B. DCT

## 1. 실행 결과

### A. DFT

#### i. 블록 크기 N = 8

Microsoft Visual Studio 디버그 콘솔

```
IDFT : 99.66 %  
IDFT : 99.68 %  
IDFT : 99.71 %  
IDFT : 99.73 %  
IDFT : 99.76 %  
IDFT : 99.78 %  
IDFT : 99.80 %  
IDFT : 99.83 %  
IDFT : 99.85 %  
IDFT : 99.88 %  
IDFT : 99.90 %  
IDFT : 99.93 %  
IDFT : 99.95 %  
IDFT : 99.98 %  
IDFT : 100.00 %  
MSE : 0.000000  
PSNR : inf  
  
영상 일치  
  
DFT 소요된 시간 : 8.079  
start = 0  
end = 8079
```

Rec\_DFT\_8 - frame: 1/1

File Size Color Zoom Segment




#### ii. 블록 크기 N = 512, low pass filter 적용 안함

Microsoft Visual Studio 디버그 콘솔

```
IDFT : 97.27 %  
IDFT : 97.46 %  
IDFT : 97.66 %  
IDFT : 97.85 %  
IDFT : 98.05 %  
IDFT : 98.24 %  
IDFT : 98.44 %  
IDFT : 98.63 %  
IDFT : 98.83 %  
IDFT : 99.02 %  
IDFT : 99.22 %  
IDFT : 99.41 %  
IDFT : 99.61 %  
IDFT : 99.80 %  
IDFT : 100.00 %  
MSE : 0.000000  
PSNR : inf  
  
영상 일치  
  
DFT 소요된 시간 : 1246.342  
start = 1  
end = 1246343
```

Rec\_DFT - frame: 1/1

File Size Color Zoom Segment



### iii. 블록 크기 $N = 512$ , low pass filter 적용 함

Microsoft Visual Studio 디버그 콘솔

```
IDFT : 97.27 %  
IDFT : 97.46 %  
IDFT : 97.66 %  
IDFT : 97.85 %  
IDFT : 98.05 %  
IDFT : 98.24 %  
IDFT : 98.44 %  
IDFT : 98.63 %  
IDFT : 98.83 %  
IDFT : 99.02 %  
IDFT : 99.22 %  
IDFT : 99.41 %  
IDFT : 99.61 %  
IDFT : 99.80 %  
IDFT : 100.00 %  
MSE : 161.566257  
PSNR : 25.770432  
  
영상 불일치  
  
DFT 소요된 시간 : 1313.461  
start = 0  
end = 1313461
```

Rec\_DFT\_LPF - frame: 1/1

File Size Color Zoom Segment



Rec\_DFT\_LPF - frame: 1/1

## B. DCT


### i. 블록 크기 $N = 8$

Microsoft Visual Studio 디버그 콘솔

```
IDCT : 99.66 %  
IDCT : 99.68 %  
IDCT : 99.71 %  
IDCT : 99.73 %  
IDCT : 99.76 %  
IDCT : 99.78 %  
IDCT : 99.80 %  
IDCT : 99.83 %  
IDCT : 99.85 %  
IDCT : 99.88 %  
IDCT : 99.90 %  
IDCT : 99.93 %  
IDCT : 99.95 %  
IDCT : 99.98 %  
IDCT : 100.00 %  
MSE : 0.000000  
PSNR : inf  
  
영상 일치  
  
DCT 소요된 시간 : 10.598  
start = 0  
end = 10598
```

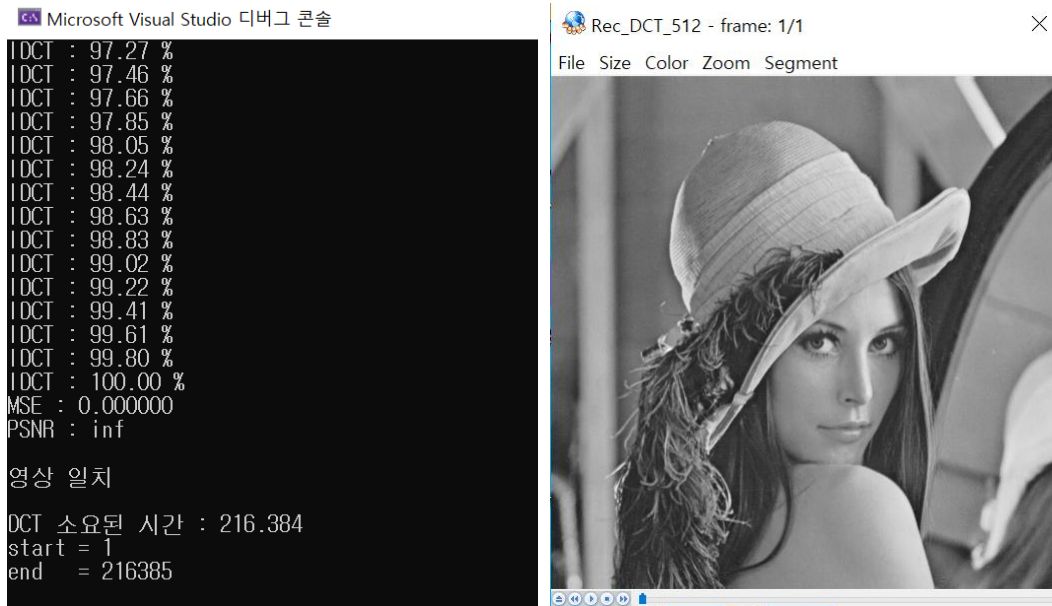
Rec\_DCT\_8 - frame: 1/1

File Size Color Zoom Segment



Rec\_DCT\_8 - frame: 1/1

## ii. 블록 크기 N = 512



## 2. 코드 ( 주석 포함 )

### A. DFT

main.h

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <memory.h>  
#include <math.h>  
#include <time.h>  
#define _CRT_SECURE_NO_WARNINGS  
  
#define WIDTH      512          // 영상의 가로 크기  
#define HEIGHT     512          // 영상의 세로 크기  
  
#define maxVal     255  
#define minVal     0  
  
#define pi          3.141592653589793238462643383279  
  
#define CLIP(x) (x < minVal) ? minVal : x > maxVal ? maxVal : x //클리핑
```

```

#define BLOCK_SIZE 512 //8, 512 변경
해야함 (본인이 수정해야함)
#if BLOCK_SIZE == 512 // 8일때는 거짓
#define TransType 1 //1이면 현재 블록은 512x512, 0이면 그 이외의 블록
#if TransType
#define flagLPF 1 // LPF 적용 유무, 1이면 적용, 0이면 미적용 //(본인이 수정해야함)
#if flagLPF
#define D0 32.0 // Cut off frequency (고정)
#define N0 4.0 // Filter dimension (고정)
#endif
#endif
#else // 8일때 참
#define TransType 0
#endif

```

```

typedef unsigned char UChar;
typedef char Char;
typedef double Double;
typedef int Int;

```

```

typedef struct _DFT_Buffer
{
    Double* picReal; // 실수부
    Double* picImag; // 허수부

    Double* picMagnitude; // 크기
    Double* picPhase; // 위상
}DFT_Val;

```

```

typedef struct _Image_Buffer
{
    UChar* ori; //원본 영상 저장을 위한 변수 선언
    UChar* rec; //복원 영상 저장
}Img_Buf;

```

```

void DFT_Process(Img_Buf* img, DFT_Val* DFT);
void DFT_Func(UChar* curBlk, Int blkSize, Int blkRow, Int blkCol, DFT_Val* DFT);

```

```

void IDFT_Process(Img_Buf* img, DFT_Val* DFT);
void IDFT_Func(Double* blkMag, Double* blkPha, Int blkSize, Int blkRow, Int blkCol, Img_Buf* img);

void LPF(Double* blkReal, Double* blkImag, Int wid, Int hei);
void PSNR(Img_Buf* img);

```

main.c

```

#include "main.h"

```

```

void ImageCreate(Img_Buf*img, Int picSize) // 이미지 불러오기 함수
{
    FILE* fp;

    fopen_s(&fp, "lena_512x512.raw", "rb"); //원본 영상 열기
    img->ori = (UChar*)malloc(sizeof(UChar) * picSize); //원본 영상 크기만큼 공간 선언
    img->rec = (UChar*)malloc(sizeof(UChar) * picSize); //결과 영상 크기만큼 공간 선언

    memset(img->ori, 0, sizeof(UChar) * picSize); //0으로 초기화
    fread(img->ori, sizeof(UChar), picSize, fp); // 원본 영상 읽기(원본 영상의 픽셀 값을 배열 변수에 저장)

    fclose(fp);
}

void ImageOutput(Img_Buf* img, Int picSize) //이미지 출력 함수
{
    FILE* fp;

    fopen_s(&fp, "Rec_DFT_LPF.raw", "wb"); //결과 영상 파일 열기

    //파일 이름 저장 형식
    //512*512 , LPF x : Rec_DFT.raw
    //512*512 , LPF o : Rec_DFT_LPF.raw
    // 8*8 : Rec_DFT_8.raw

```

```
    fwrite(img->rec, sizeof(UChar), picSize, fp); // 원본 영상 읽기(원본 영상의 픽셀 값을 배열 변수에 저장)
```

```
    fclose(fp);  
    free(img->ori);  
    free(img->rec);  
}
```

```
void main()
```

```
{
```

```
    Img_Buf img; //원본 영상과 복원 영상 버퍼를 잡아주는 버퍼가 들어있음
```

```
    Int wid = WIDTH; Int hei = HEIGHT;
```

```
    Int min = minVal; Int max = maxVal;
```

```
    Int picSize = wid * hei; //영상 사이즈
```

```
    DFT_Val DFT; // DFT 버퍼
```

```
    clock_t start, end; // 시간을 세기 위한 변수
```

```
    float res;
```

```
    ImageCreate(&img, picSize);
```

```
    start = clock(); // 시간 (초) 세기 시작
```

```
    DFT_Process (&img, &DFT); // Forward DFT
```

```
    IDFT_Process(&img, &DFT); // inverse DFT
```

```
    PSNR(&img); // 생겨난 원본 영상과 복원 영상의 MSE 와 PSNR을 계산함
```

```
    end = clock(); // 시간 세기 끝
```

```
    res = (float)(end - start) / CLOCKS_PER_SEC; // 기록된 시간 계산
```

```
    ImageOutput(&img, picSize); // 이미지 출력
```

```
    printf("\nDFT 소요된 시간 : %.3f \n", res);
```

```
    printf("start = %d \n", start);
```

```
    printf("end    = %d \n", end);
```

```
    printf("\n\n\n");
```

```
}
```

```

void PSNR(Img_Buf* img)
{
    Int i, j;
    Int wid = WIDTH; Int hei = HEIGHT;
    Double mse = 0, psnr = 0, max = 0; //mse : 평균제곱오차, psnr: 새로 생긴 영상에 대한 손실
    정보를 갖고 있음 ( 손실이 적을수록 높은 값을 가짐 )
    UChar Img1 = 0, Img2 = 0;

    for (i = 0; i < hei; i++)
    {
        for (j = 0; j < wid; j++)
        {
            Img1 = img->ori[i * wid + j];
            Img2 = img->rec[i * wid + j];

            mse += ((Img1 - Img2) * (Img1 - Img2)); // 오차 값 제곱 누적
            if (Img1 > max)
                max = Img1;
        }
    }

    mse = mse / (wid * hei); // 평균 구함
    printf("MSE : %lf\n", mse);
    psnr = 20 * log10(max / sqrt(mse)); // 무손실 영상의 경우 (= mse 가 0이면 ) 계산 할 수 없음
    -> 무한대
    printf("PSNR : %lf\n", psnr); //

    if (mse == 0)
        printf("\n영상 일치\n"); // 오차 0 이면 일치
    else
        printf("\n영상 불일치\n"); // 오차가 0 보다 크면 불일치
}

```

DFT.c

```
#include "main.h"
```



```

#if flagLPF
void LPF(Double* blkReal, Double* blkImag, Int wid, Int hei)
{
    // 버터워스 LPF 구현

    int x, y;
    int tempx, tempy;
    int halfcols, halfrows;
    double butterworth, coordinate;
    halfcols = hei / 2;
    halfrows = wid / 2; // cutoff 찾기 위해 1/2 값 구함

    for (y = 0; y < hei; y++)
    {
        if (y >= halfcols)
            tempy = y - halfcols;
        else
            tempy = y + halfcols; //DC로부터의 y좌표 거리 구함

        for (x = 0; x < wid; x++)
        {
            if (x >= halfrows)
                tempx = x - halfrows;
            else
                tempx = x + halfrows; //DC로부터의 x좌표 거리 구함

            coordinate = sqrt(pow((double)(tempx - halfcols), 2.0) + pow((double)(tempy -
halfcols), 2.0)); // DC로부터의 거리
            butterworth = 1.0 / (1.0 + pow(coordinate / D0, 2 * N0)); //버터워스 필터의
크기 계산

            blkReal[y * wid + x] *= butterworth;
            blkImag[y * wid + x] *= butterworth; // butterworth low pass filter 적용 ->
frequency 높은 것들(= DC로부터 거리 먼 것들) 없애기

        }
    }
}

```

```

}
#endif
void DFT_Func(UChar* curBlk, Int blkSize, Int blkRow, Int blkCol, DFT_Val* DFT)
{
    Double PI = pi;
    Int stride = WIDTH;

    Double* blkReal;
    Double* blkImag;
    Double* blkMag;
    Double* blkPha;

    blkReal = (Double*)calloc(blkSize * blkSize, sizeof(Double)); // 실수, 허수, 크기, 위상 부분 공간
    blkImag = (Double*)calloc(blkSize * blkSize, sizeof(Double)); // 블록 크기 blkSize * blkSize
    blkMag = (Double*)calloc(blkSize * blkSize, sizeof(Double));
    blkPha = (Double*)calloc(blkSize * blkSize, sizeof(Double));

    #if TransType // 512*512 의 경우
        Int count = 0;
        Double* cosTable, * sinTable;
        FILE* fp, * op;

        cosTable = (Double*)calloc(WIDTH * HEIGHT * 2, sizeof(Double));
        sinTable = (Double*)calloc(WIDTH * HEIGHT * 2, sizeof(Double));

        fopen_s(&fp, "cosTable.txt", "rb");
        fopen_s(&op, "sinTable.txt", "rb");

        fread(cosTable, sizeof(double), WIDTH * HEIGHT * 2, fp);
        fread(sinTable, sizeof(double), WIDTH * HEIGHT * 2, op); // 시간 단축을 위해 cos, sin 테이블
        fclose(fp);
        fclose(op);
    #endif // 8*8의 경우

    for (Int i = 0; i < blkSize; i++) // i,j: time 도메인. k,l: frequency 도메인
    {

```

```

#if TransType
    printf("DFT : %.2f %%\n", (++count) / (double)(WIDTH / blkSize * HEIGHT) * 100);
#endif

    for (Int j = 0; j < blkSize; j++)
    {
        for (Int k = 0; k < blkSize; k++)
        {
            for (Int l = 0; l < blkSize; l++)
            {
                #if TransType
                    blkReal[i * blkSize + j] += curBlk[k * blkSize + l] * cosTable[i
* k + j * l];
                    blkImag[i * blkSize + j] -= curBlk[k * blkSize + l] * sinTable[i
* k + j * l];
                #else
                    blkReal[i * blkSize + j] += curBlk[k * blkSize + l] * cos(2 * PI
* (i * k + j * l) / (double)blkSize); // 실수부 계산
                    blkImag[i * blkSize + j] -= curBlk[k * blkSize + l] * sin(2 * PI *
(i * k + j * l) / (double)blkSize); // 허수부 계산
                #endif
            }
        }
        blkReal[i * blkSize + j] = blkReal[i * blkSize + j] / (blkSize * blkSize);
        blkImag[i * blkSize + j] = blkImag[i * blkSize + j] / (blkSize * blkSize);
    }
}

#if TransType

    #if flagLPF // flagLPF=1 이면
        LPF(blkReal, blkImag, WIDTH, HEIGHT); // Low pass filter
    #endif
#endif

    for (Int i = 0; i < blkSize; i++)
    {
        for (Int j = 0; j < blkSize; j++)
        {

```

```

        blkMag[i * blkSize + j] = sqrt(blkReal[i * blkSize + j] * blkReal[i * blkSize + j] +
blkImag[i * blkSize + j] * blkImag[i * blkSize + j]); // 크기 구하기
        blkPha[i * blkSize + j] = atan2(blkImag[i * blkSize + j], blkReal[i * blkSize + j]); //
위상 구하기
    }
}

for (Int i = 0; i < blkSize; i++) //블록을 원래 있던 픽처 위치에 저장
{
    for (Int j = 0; j < blkSize; j++)
    {
        DFT->picReal    [(blkRow * blkSize + i) * stride + (blkCol * blkSize + j)] =
blkReal[i * blkSize + j];
        DFT->picImag    [(blkRow * blkSize + i) * stride + (blkCol * blkSize + j)] =
blkImag[i * blkSize + j];
        DFT->picMagnitude[(blkRow * blkSize + i) * stride + (blkCol * blkSize + j)] =
blkMag[i * blkSize + j];
        DFT->picPhase    [(blkRow * blkSize + i) * stride + (blkCol * blkSize + j)] =
blkPha[i * blkSize + j];
    }
}
// 스펙트럼 구하기
#ifdef TransType

Int wid = WIDTH;
Int hei = HEIGHT;

Double C, Temp, Spec;
Double* SpecTmp;
UChar* Shuffling;

SpecTmp    = (Double*)calloc(blkSize * blkSize, sizeof(Double));
Shuffling = (UChar*)calloc(blkSize * blkSize, sizeof(UChar));

if (blkSize == wid && blkSize == hei) // 주파수 동작 범위가 넓어서 스케일 다운을 위해 로그
취함
{
    C = hypot(DFT->picReal[(blkRow * blkSize) + (blkCol * (wid + (wid % blkSize)) *
blkSize)], DFT->picImag[(blkRow * blkSize) + (blkCol * (wid + (wid % blkSize)) * blkSize)]);
    for (Int i = 0; i < blkSize; i++)
    {

```

```

        Spec = 0;
        for (Int j = 0; j < blkSize; j++)
        {
            Temp = hypot(DFT->picReal[(blkRow * blkSize) + (blkCol * (wid +
(wid % blkSize)) * blkSize) + ((wid + (wid % blkSize)) * i) + j], DFT->picImag[(blkRow * blkSize) + (blkCol *
(wid + (wid % blkSize)) * blkSize) + ((wid + (wid % blkSize)) * i) + j]);
            Spec = (C * log(1.0 + abs(Temp))) < 0.0 ? 0.0 : (C * log(1.0 +
abs(Temp))) > 255.0 ? 255.0 : (C * log(1.0 + abs(Temp)));
            SpecTmp[i * blkSize + j] = Spec;
        }
    }

    // 셔플링 -> 대칭을 이용해 에너지 센터링
    for (Int i = 0; i < blkSize; i += (blkSize / 2))
    {
        for (Int j = 0; j < blkSize; j += (blkSize / 2))
        {
            for (Int k = 0; k < (blkSize / 2); k++)
            {
                for (Int l = 0; l < (blkSize / 2); l++)
                {
                    Shuffling[wid * (k + i) + (l + j)] =
(UChar)SpecTmp[wid * (255 - k + i) + (255 - l + j)];
                }
            }
        }
    }

    FILE* cp;
    fopen_s(&cp, "DFT_Spectrum.raw", "wb");

    fwrite(Shuffling, sizeof(UChar), blkSize * blkSize, cp); // 스펙트럼 출력

    free(SpecTmp);
    free(Shuffling);

    fclose(cp);
}

```

```

#endif

```

```

        free(blkReal);
        free(blkImag);
        free(blkMag);
        free(blkPha);

    #if TransType
        free(cosTable);
        free(sinTable);
    #endif

}

void DFT_Process(Img_Buf* img, DFT_Val* DFT)
{
    Int blkSize = BLOCK_SIZE; //블록 사이즈 8 or 512
    Int wid = WIDTH; Int hei = HEIGHT; // 영상 사이즈
    Int min = minVal; Int max = maxVal;
    Int picSize = wid * hei; //영상 사이즈
    Int stride = wid;

    UChar* TL_curBlk; // 블록의 왼쪽 위 좌표
    UChar* TMP;
    UChar* curBlk; //블록 반환 변수

    DFT->picReal      = (Double*)calloc(picSize, sizeof(Double)); //실수부분, 허수부분, 크기, 위상
메모리 할당
    DFT->picImag      = (Double*)calloc(picSize, sizeof(Double)); //영상 사이즈 만큼 할당
    DFT->picMagnitude = (Double*)calloc(picSize, sizeof(Double));
    DFT->picPhase      = (Double*)calloc(picSize, sizeof(Double));

    #if TransType == 0
        Int count = 0;
    #endif

    //DFT
    curBlk = (UChar*)calloc(blkSize * blkSize, sizeof(UChar)); // 블록 공간 할당

    for (Int blkRow = 0; blkRow < hei / blkSize; blkRow++) //블록의 왼쪽 위 좌표를 찾기 위한
반복문
    {

```

```

        for (Int blkCol = 0; blkCol < wid / blkSize; blkCol++)//블록의 왼쪽 위 좌표를 찾기 위한
반복문
    {
        memset(curBlk, 0, sizeof(UChar) * blkSize * blkSize);
        //////////////////////////////////////
        TL_curBlk = img->ori +(blkRow* blkSize)*stride + (blkCol* blkSize);        //
블록의 왼쪽 위 좌표
        for (int k = 0; k < blkSize; k++)        //블록의 왼쪽 위 좌표를 기준으로 한
blkSize x blkSize 추출
            for (int l = 0; l < blkSize; l++)
            {
                TMP = TL_curBlk + (k * stride + l);
                curBlk[k * blkSize + l] = TMP[0];
            }//curBlk에 blkSize x blkSize 의 화소 값 들어있음 -> DFT 에 넣기

        DFT_Func(curBlk, blkSize, blkRow, blkCol, DFT); //curBlk: 현재 블록(blkSize
x blkSize)
#ifdef TransType == 0
        count += (blkSize * blkSize);// 현재 DFT가 얼마나 진행 되었는지 알려주는
것
        printf("DFT : %.2f %%\n", (double)count/(wid*hei)*100);
#endif
    }
}
free(curBlk);
}

```

IDFT.c

```

#include "main.h"

void IDFT_Func(Double* blkMag, Double* blkPha, Int blkSize, Int blkRow, Int blkCol, Img_Buf* img)
{
    Double PI = pi;
    Int stride = WIDTH;

    Double* DFT_Real;
    Double* DFT_Imag;
    Double Recon_R;

```

```

DFT_Real = (Double*)calloc(blkSize * blkSize, sizeof(Double));
DFT_Imag = (Double*)calloc(blkSize * blkSize, sizeof(Double));

#if TransType //512 경우
    Int count = 0;
    Double* cosTable, * sinTable;
    FILE* fp, * op;

    cosTable = (Double*)calloc(WIDTH * HEIGHT * 2, sizeof(Double));
    sinTable = (Double*)calloc(WIDTH * HEIGHT * 2, sizeof(Double)); // 계산 시간 줄이기 위해 sin,
cos table 만들어놓음

    fopen_s(&fp, "cosTable.txt", "rb");
    fopen_s(&op, "sinTable.txt", "rb");

    fread(cosTable, sizeof(double), WIDTH * HEIGHT * 2, fp);
    fread(sinTable, sizeof(double), WIDTH * HEIGHT * 2, op);

    fclose(fp);
    fclose(op);
#endif //8의 경우

    for (Int i = 0; i < blkSize; i++)// 크기와 위상으로 실수부와 허수부 구해주기
    {
        for (Int j = 0; j < blkSize; j++)
        {
            DFT_Real[i * blkSize + j] = blkMag[i * blkSize + j] * cos(blkPha[i * blkSize + j]);
            DFT_Imag[i * blkSize + j] = blkMag[i * blkSize + j] * sin(blkPha[i * blkSize + j]);
        }
    }

    for (Int i = 0; i < blkSize; i++)
    {
        #if TransType
            printf("IDFT : %.2f %%\n", (++count) / (double)(WIDTH / blkSize * HEIGHT) * 100);
        #endif

        for (Int j = 0; j < blkSize; j++)
        {
            Recon_R = 0;

```



```

        for (Int k = 0; k < blkSize; k++)
        {
            for (Int l = 0; l < blkSize; l++)
            {
                #if TransType //512
                    Recon_R += DFT_Real[k * blkSize + l] * cosTable[i * k + j *
l] -
                    DFT_Imag[k * blkSize + l] * sinTable[i * k + j * l];
                #else //8
                    Recon_R += DFT_Real[k * blkSize + l] * cos(2 * PI * (i * k +
j * l) / (double)blkSize) -
                    DFT_Imag[k * blkSize + l] * sin(2 * PI * (i * k + j * l)
/ (double)blkSize);
                #endif
            }
        }

        if (Recon_R < 0)
            Recon_R = (int)(Recon_R - 0.5);

        else
            Recon_R = (int)(Recon_R + 0.5); //클리핑

        img->rec[(blkRow * blkSize + i) * stride + (blkCol * blkSize + j)] =
CLIP(Recon_R); //각각 복원된 것을 복원영상에 넣어줌
    }
}

free(DFT_Real);
free(DFT_Imag);
#if TransType
free(cosTable);
free(sinTable);
#endif

}

void IDFT_Process(Img_Buf* img, DFT_Val* DFT)
{
    Int blkSize = BLOCK_SIZE;
    Int wid = WIDTH; Int hei = HEIGHT;

```

```

Int stride = wid;

// 실수부분에서 크기와 페이지만 갖고 옴

Double* TL_blkMag;
Double* Mag_TMP;
Double* blkMag;

Double* TL_blkPha;
Double* Pha_TMP;
Double* blkPha;

#if TransType == 0
    Int count = 0;
#endif

blkMag = (Double*)calloc(blkSize * blkSize, sizeof(Double));
blkPha = (Double*)calloc(blkSize * blkSize, sizeof(Double));
for (Int blkRow = 0; blkRow < hei / blkSize; blkRow++)
{
    for (Int blkCol = 0; blkCol < wid / blkSize; blkCol++)
    {
        memset(blkMag, 0, sizeof(Double) * blkSize * blkSize);
        memset(blkPha, 0, sizeof(Double) * blkSize * blkSize);
        // 픽처 단위 -> 블록단위로 다시 바꿔줌
        TL_blkMag = DFT->picMagnitude + (blkRow * blkSize) * stride + (blkCol *
blkSize);
        TL_blkPha = DFT->picPhase      + (blkRow * blkSize) * stride + (blkCol *
blkSize);

        for (int k = 0; k < blkSize; k++)
            for (int l = 0; l < blkSize; l++)
            {
                Mag_TMP = TL_blkMag + (k * stride + l);
                blkMag[k * blkSize + l] = Mag_TMP[0]; // 크기를 블록
단위로 가져오기

                Pha_TMP = TL_blkPha + (k * stride + l);
                blkPha[k * blkSize + l] = Pha_TMP[0]; // 위상을 블록단위로
가져오기
            }
    }
}

```

```

        IDFT_Func(blkMag, blkPha, blkSize, blkRow, blkCol, img);
    #if TransType == 0
        count += (blkSize * blkSize);
        printf("IDFT : %.2f %%\n", (double)count / (wid * hei) * 100);
    #endif
    }
}
free(blkMag);
free(blkPha);

free(DFT->picMagnitude);
free(DFT->picPhase);
free(DFT->picReal);
free(DFT->picImag);
}

```

DCT

main.h

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include <math.h>
#include <time.h>
#define _CRT_SECURE_NO_WARNINGS

#define WIDTH      512          // 영상의 가로 크기
#define HEIGHT     512          // 영상의 세로 크기

#define maxVal     255
#define minVal     0

#define pi         3.141592653589793238462643383279

```

```

#define CLIP(x) (x < minVal ? minVal : x > maxVal ? maxVal : x

#define BLOCK_SIZE 512 //본인이
수정해야함)
#define BLOCK_SIZE == 512
#define TransType 1 //1이면 현재 블록은 512x512, 0이면 그 이외의 블록
#else
#define TransType 0
#endif

typedef unsigned char UChar;
typedef char Char;
typedef double Double;
typedef int Int;

typedef struct _DCT_Buffer
{
    Double* imgDCT; //DCT가 적용된 결과 저장 버퍼( 실수 값만 존재 )
}DCT_Val;

typedef struct _Image_Buffer
{
    UChar* ori; //원본 영상 저장을 위한 변수 선언
    UChar* rec; //복원 영상 저장
}Img_Buf;

void DCT_Process(Img_Buf* img, DCT_Val* DCT);
void DCT_Func(UChar* curBlk, Int blkSize, Int blkRow, Int blkCol, DCT_Val* DCT);

void IDCT_Process(Img_Buf* img, DCT_Val* DCT);
void IDCT_Func(Double* curBlk, Int blkSize, Int blkRow, Int blkCol, Img_Buf *img);

void PSNR(Img_Buf* img);

```

main.c

```
#include "main.h"
```

```
void ImageCreate(Img_Buf* img, Int picSize) //이미지 열기
```

```
{
```

```
    FILE* fp;
```

```
    fopen_s(&fp, "lena_512x512.raw", "rb"); //원본 영상 열기
```

```
    img->ori = (UChar*)malloc(sizeof(UChar) * picSize); //원본 영상 크기만큼 공간 선언
```

```
    img->rec = (UChar*)malloc(sizeof(UChar) * picSize); //결과 영상 크기만큼 공간 선언
```

```
    memset(img->ori, 0, sizeof(UChar) * picSize); //0으로 초기화
```

```
    fread(img->ori, sizeof(UChar), picSize, fp); // 원본 영상 읽기(원본 영상의 픽셀 값을 배열  
변수에 저장)
```

```
    fclose(fp);
```

```
}
```

```
void ImageOutput(Img_Buf* img, Int picSize) // 이미지 출력
```

```
{
```

```
    FILE* fp;
```

```
    fopen_s(&fp, "Rec_DCT_512.raw", "wb"); //결과 영상 파일 열기
```

```
    // 8*8 : Rec_DCT_8.raw
```

```
    // 512*512 : Rec_DCT_512.raw
```

```
    fwrite(img->rec, sizeof(UChar), picSize, fp); // 원본 영상 읽기(원본 영상의 픽셀 값을 배열  
변수에 저장)
```

```
    fclose(fp);
```

```
    free(img->ori);
```

```
    free(img->rec);
```

```
}
```

```
void main()
```

```
{
```

```
    Img_Buf img;
```

```
Int wid = WIDTH; Int hei = HEIGHT;  
Int min = minVal; Int max = maxVal;
```

```
Int picSize = wid * hei; //영상 사이즈
```

```
DCT_Val DCT;  
clock_t start, end;  
float res;
```

```
ImageCreate(&img, picSize);
```

```
start = clock(); //시간 기록 시작  
DCT_Process(&img, &DCT);  
IDCT_Process(&img, &DCT);  
PSNR(&img);  
end = clock(); //시간 기록 끝  
res = (float)(end - start) / CLOCKS_PER_SEC; // 시간 기록  
ImageOutput(&img, picSize);
```

```
printf("\nDCT 소요된 시간 : %.3f \n", res);  
printf("start = %d \n", start);  
printf("end    = %d \n", end);
```

```
printf("\n\n\n");
```

```
}
```

```
void PSNR(Img_Buf* img) //DFT 참고
```

```
{
```

```
    Int i, j;  
    Int wid = WIDTH; Int hei = HEIGHT;  
    Double mse = 0, psnr = 0, max = 0;  
    UChar Img1 = 0, Img2 = 0;
```

```
    for (i = 0; i < hei; i++)
```

```
    {
```

```
        for (j = 0; j < wid; j++)
```

```
        {
```

```
            Img1 = img->ori[i * wid + j];
```

```

        lmg2 = img->rec[i * wid + j];

        mse += ((lmg1 - lmg2) * (lmg1 - lmg2));
        if (lmg1 > max)
            max = lmg1;
    }
}

mse = mse / (wid * hei);
printf("MSE : %f\n", mse);
psnr = 20 * log10(max / sqrt(mse));
printf("PSNR : %f\n", psnr);

if (mse == 0)
    printf("\n영상 일치\n");
else
    printf("\n영상 불일치\n");
}

```

DCT.c

```

#include "main.h"

void DCT_Func(UChar* curBlk, Int blkSize, Int blkRow, Int blkCol, DCT_Val* DCT)
{
    Double PI = pi;
    Int stride = WIDTH;

    Double* blkReal;

    blkReal = (Double*)calloc(blkSize * blkSize, sizeof(Double)); // 실수 부분만 존재. 메모리 할당

#ifdef TransType
    Int count = 0;
    Double* DCTcosTable;
    FILE* fp, * op;

```

```

DCTcosTable = (Double*)calloc((2 * WIDTH + 1) * HEIGHT, sizeof(Double));

fopen_s(&fp, "DCTcosTable.txt", "rb");

fread(DCTcosTable, sizeof(double), (2 * WIDTH + 1) * HEIGHT, fp);

fclose(fp);

#endif

for (Int i = 0; i < blkSize; i++)
{
#if TransType
printf("DCT : %.2f %%\n", (++count) / (double)(WIDTH / blkSize * HEIGHT) * 100);
#endif

for (Int j = 0; j < blkSize; j++)
{
for (Int k = 0; k < blkSize; k++)
{
for (Int l = 0; l < blkSize; l++)
{
#if TransType //512
// DCT 에서 구하는 것 : u,v. blkReal의 인덱스 : i *
blkSize + j
// 따라서 i = u, j = v, k = x, l = y 임
blkReal[i * blkSize + j] += curBlk[k * blkSize + l] *
DCTcosTable[(2 * k + 1) * i] * DCTcosTable[(2 * l + 1) * j];
#else //8
blkReal[i * blkSize + j] += curBlk[k * blkSize + l] * cos((2 * k
+ 1) * i * PI / (2 * (double)blkSize)) * cos((2 * l + 1) * j * PI / (2 * (double)blkSize));
#endif
}
}
//c(u),c(v) 구하기
if (i == 0 && j == 0)
blkReal[i * blkSize + j] /= blkSize;
else if (i == 0 || j == 0)
blkReal[i * blkSize + j] = (blkReal[i * blkSize + j] * (1.0 / sqrt(2.0))) /
(blkSize / 2);
}
}

```



```

        else
            blkReal[i * blkSize + j] = blkReal[i * blkSize + j] / (blkSize / 2);
    }
}

for (Int i = 0; i < blkSize; i++)    //블록을 원래 있던 픽처 위치에 저장
{
    for (Int j = 0; j < blkSize; j++)
    {
        DCT->imgDCT[(blkRow * blkSize + i) * stride + (blkCol * blkSize + j)] =
blkReal[i * blkSize + j]; //DCT 결과값 저장
    }
}

#if TransType
    Int wid = WIDTH;
    Int hei = HEIGHT;

    if (blkSize == wid && blkSize == hei)
    {
        UChar* spectrum;

        FILE* cp;
        fopen_s(&cp, "DCT_Spectrum.raw", "wb");

        spectrum = (UChar*)calloc(blkSize * blkSize, sizeof(UChar));
        for (Int i = 0; i < blkSize; i++)
        {
            for (Int j = 0; j < blkSize; j++)
            {
                spectrum[i * wid + j] = CLIP(DCT->imgDCT[i * wid + j]); // 클리핑
            }
        }

        fwrite(spectrum, sizeof(UChar), blkSize * blkSize, cp);

        free(spectrum);
        fclose(cp);
    }
#endif

```

```

        free(blkReal);

#ifdef TransType
        free(DCTcosTable);
#endif

    }

    void DCT_Process(Img_Buf* img, DCT_Val* DCT)
    {
        Int blkSize = BLOCK_SIZE;
        Int wid = WIDTH; Int hei = HEIGHT;
        Int min = minVal; Int max = maxVal;
        Int picSize = wid * hei; //영상 사이즈
        Int stride = wid;

        UChar* TL_curBlk;
        UChar* TMP;
        UChar* curBlk; //원본 영상을 블록 사이즈로 쪼개기 위해 함

        DCT->imgDCT = (Double*)calloc(picSize, sizeof(Double)); //DCT결과 임시 저장

#ifdef TransType == 0
        Int count = 0;
#endif

        //DCT
        curBlk = (UChar*)calloc(blkSize * blkSize, sizeof(UChar));

        for (Int blkRow = 0; blkRow < hei / blkSize; blkRow++) //블록의 왼쪽 위 좌표를 찾기 위한
반복문
        {
            for (Int blkCol = 0; blkCol < wid / blkSize; blkCol++)//블록의 왼쪽 위 좌표를 찾기 위한
반복문
            {
                memset(curBlk, 0, sizeof(UChar) * blkSize * blkSize);
                //////////////////////////////////////
                TL_curBlk = img->ori + (blkRow * blkSize) * stride + (blkCol * blkSize);

                //블록의 왼쪽 위 좌표
                for (int k = 0; k < blkSize; k++) //블록의 왼쪽 위 좌표를 기준으로 한
반복문
                {
                    blkSize x blkSize 추출

```

```

        for (int l = 0; l < blkSize; l++)
        {
            TMP = TL_curBlk + (k * stride + l);
            curBlk[k * blkSize + l] = TMP[0];
        }

DCT_Func(curBlk, blkSize, blkRow, blkCol, DCT); //curBlk: 현재 블록(blkSize
x blkSize)
#ifdef TransType == 0
    count += (blkSize * blkSize);
    printf("DCT : %.2f %%\n", (double)count / (wid * hei) * 100);
#endif
    }
}
free(curBlk);
}

```

IDFT.c

```

#include "main.h"

void IDCT_Func(Double* curBlk, Int blkSize, Int blkRow, Int blkCol, Img_Buf* img)
{
    Double PI = pi;
    Int stride = WIDTH;

    Double DCT_Real;
    Double Recon_R;

#ifdef TransType
    Int count = 0;

    Double* DCTcosTable;
    FILE* fp, * op;

    DCTcosTable = (Double*)calloc((2 * WIDTH + 1) * HEIGHT, sizeof(Double)); //코사인 table

    fopen_s(&fp, "DCTcosTable.txt", "rb");

```

```

fread(DCTcosTable, sizeof(double), (2 * WIDTH + 1) * HEIGHT, fp);

fclose(fp);
#endif

for (int i = 0; i < blkSize; i++)
{
    #if TransType
        printf("IDCT : %.2f %%\n", (++count) / (double)(WIDTH / blkSize * HEIGHT) * 100);
    #endif

    for (int j = 0; j < blkSize; j++)
    {
        Recon_R = 0;
        for (int k = 0; k < blkSize; k++)
        {
            for (int l = 0; l < blkSize; l++)
            {
                #if TransType //512
                    //i=x,j=y, k=u, l=v.
                    DCT_Real = curBlk[k * blkSize + l] * DCTcosTable[(2 * i +
1) * k] * DCTcosTable[(2 * j + 1) * l];
                #else//8
                    DCT_Real = curBlk[k * blkSize + l] * cos((2 * i + 1) * k * PI /
(2 * blkSize)) * cos((2 * j + 1) * l * PI / (2 * blkSize));//?
                #endif

                //c(u),c(v) 구하기
                if (k == 0 && l == 0)
                    Recon_R += DCT_Real / blkSize;
                else if (k == 0 || l == 0)
                    Recon_R += (DCT_Real * (1.0 / sqrt(2.0))) /
(blkSize / 2);
                else
                    Recon_R += DCT_Real / (blkSize / 2);
            }
        }

        if (Recon_R < 0)//클리핑
            Recon_R = (int)(Recon_R - 0.5);
    }
}

```

```

else
    Recon_R = (int)(Recon_R + 0.5);

    img->rec[(blkRow * blkSize + i) * stride + (blkCol * blkSize + j)] =
CLIP(Recon_R);
    }
}

#if TransType
    free(DCTcosTable);

#endif

}

void IDCT_Process(Img_Buf* img, DCT_Val* DCT)
{
    Int blkSize = BLOCK_SIZE;
    Int wid = WIDTH; Int hei = HEIGHT;
    Int stride = wid;

    Double* TL_curBlk;
    Double* TMP;
    Double* curBlk;

    #if TransType == 0
        Int count = 0;
    #endif

    curBlk = (Double*)calloc(blkSize * blkSize, sizeof(Double));

    for (Int blkRow = 0; blkRow < hei / blkSize; blkRow++)//블록의 왼쪽 위 좌표를 찾기 위한
반복문
    {
        for (Int blkCol = 0; blkCol < wid / blkSize; blkCol++)
        {
            memset(curBlk, 0, sizeof(Double) * blkSize * blkSize);

            TL_curBlk = DCT->imgDCT + (blkRow * blkSize) * stride + (blkCol * blkSize);

            for (int k = 0; k < blkSize; k++)
                for (int l = 0; l < blkSize; l++)

```

```

        {
            TMP = TL_curBlk + (k * stride + l);
            curBlk[k * blkSize + l] = TMP[0];
        }

    IDCT_Func(curBlk, blkSize, blkRow, blkCol, img);

    #if TransType == 0

        count += (blkSize * blkSize);
        printf("IDCT : %.2f %%\n", (double)count / (wid * hei) * 100);

    #endif

    }

    free(curBlk);

    free(DCT->imgDCT);
}

```