Basic JPEG Compressor and Decompressor

Background

In this project we implemented the basic JPEG compressor and decompressor. The JPEG compression algorithm consists of a number of processing steps:

1. The input image is first converted to YUV color space.
2. This image is then split into 16x16 pixel macroblocks resulting in 3 16x16 arrays (one each for Y, U and V)
3. The Y macroblock is then split in 4 8x8 blocks, while the U and V blocks are subsampled to 1 8x8 block each.
4. Discrete Cosine Transform is then performed on each of these 8x8 blocks.
5. Quantization is performed with the help of an 8x8 quantization matrix with each entry of the DCT matrix divided by the product of corresponding entry in the quantization matrix and a quantization factor.
6. Each element of the resulting matrix is then entropy encoded, zig-zag ordered and run-length encoded, followed by Huffman or Arithmetic compression.

The resultant is a JPEG image.

Implementation

Compression

As a part of this project, we have implemented steps 2,3,4 and 5 from above on an image in the PGM format. The PGM files are greyscale images, hence potentially consisting of only the Y color space, i.e. 8 bits per pixel. The PGM files are binary encoded.

Our implementation starts with opening the quantization file with file pointer qFile, reading it and saving the quantization matrix into the *quant* matrix. The qScale variable holds the quantization factor.

The input image file (pointed by iImg) is opened and the values of xsize and *ysize* are read. The maximum value of the value of pixel is saved in the variable *maxVal*, and is assumed to be 255 (8 bits) for further calculations. Then the output file is opened in write mode where the results of further calculations are stored.

The processing of the input image data is done piece by piece. At one instance of time, we consider one 16X16 macroblock in matrix '*a*' which is further divided into 4 8x8 blocks: a1 (top left), a2 (top right), a3(bottom left) and a4(bottom right). Each of these blocks are then passed to the encode() function that does the rest of the processing and writes the results to an output file.

The first job of the encode() function is to perform Discrete Cosine transform in the 8X8 array passed into it. The pixels of the image (array) in space domain variables i(x) and j(y) are converted to the array elements (F) in frequency domain variables u and v according to the DCT formula with the help of for loops and another function that gives the coefficient with respect to the values of u and v.

The DCT formula is:

$$F(u,v) = \frac{C_u}{2}\frac{C_v}{2}\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\left[\frac{(2x+1)\,\mu\pi}{16}\right]\cos\left[\frac{(2y+1)\mu\pi}{16}\right]$$

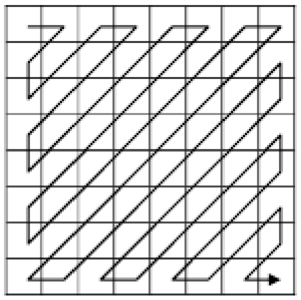
Where:

$$C_u = \begin{cases} \frac{1}{\sqrt{2}}\; if\ u = 0 \\ 1\; if\ u > 0 \end{cases} \quad C_v = \begin{cases} \frac{1}{\sqrt{2}}\; if\ v = 0 \\ 1\; if\ v > 0 \end{cases}$$

The elements of the resulting decimal array (F) are then divided by the product of the quantization factor (qScale) and the corresponding element of quantization array (quant), rounded off to an integer and stored in the array qt. Specifically:

$$qt(u,v) = \left|\frac{F(u,v)}{qScale * quant(u,v)}\right|$$

The value of qt is then truncated and shifted to fall in the range [0,255] and this value is written in zig-zag order to the output file. The figure on the right shows the zig-zag order. The algorithm used to traverse the matrix in this order is:



1. To initialize u and v with 0
2. v= v+1
3. While v is greater than zero v=v-1, u = u+1
4. u = u+1
5. while u is greater than zero u = u-1 and v = v+1
6. loop from step 2 until u =8 and v=8.

At each modification step, function printZZ was called to print in order to keep track of new line.

The result of this process on a PGM file (P5) will be a file with header MYDCT that occupies the frequency domain transformation of spatial pixels of the image. This file will be larger than the input image because we haven't completed the compression algorithm (use of redundancy by run length encoding and Huffman compression).

Decompression

In order to reconstruct the image from a MYDCT type file, the decompressor part of the JPEG algorithm is implemented in myIDCT.c. This file follows similar initial steps like the myDCT.c file, i.e. reading and storing the quantization matrix, obtaining xsize, ysize and quantization factor from the input file (output of compressor).

The pixel data is then processed piece by piece for every 8x8 blocks as encoded in the input file. A block is read and stored in un-zig-zag order by reading the file and storing it into matrix qt(u,v) with u and v varying exactly as when we zig-zag ordered it. This matrix is then de-quantized, transformed to space domain integer matrix fun(i,j) by Inverse Discrete Cosine Transform:

$$f(x,y) = \sum_{u=0}^{7}\sum_{v=0}^{7}\frac{C_u}{2}\frac{C_v}{2}F(u,v)\cos\left[\frac{(2x+1)\,\mu\pi}{16}\right]\cos\left[\frac{(2y+1)\mu\pi}{16}\right]$$

With values of Cu and Cv as before.

The entries in matrix fun is then truncated and shifted to fit in the range [0,255] and written to output matrix output(i+m,j+n) where (m,n) is the position of the first pixel of the 8x8 block.

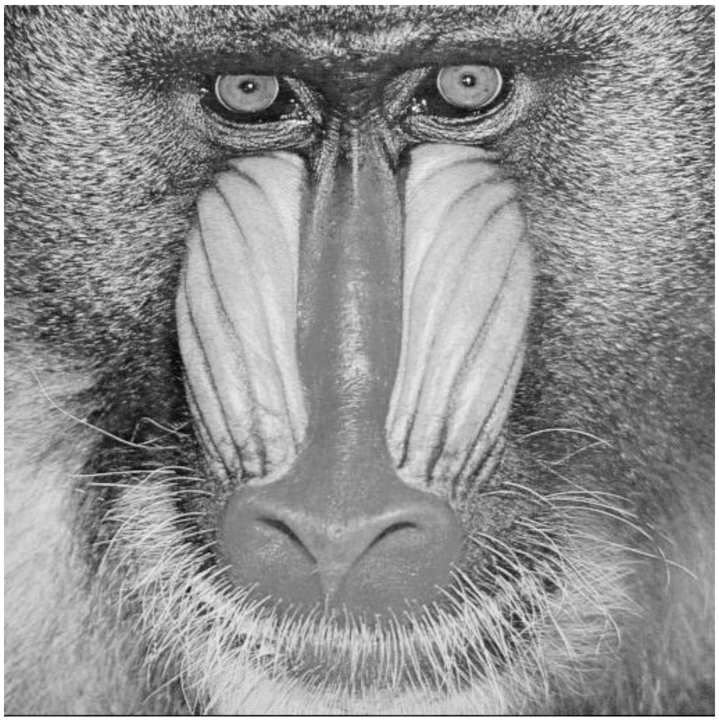
The output matrix is then printed to the output file in ASCII format.

Testing and Observation

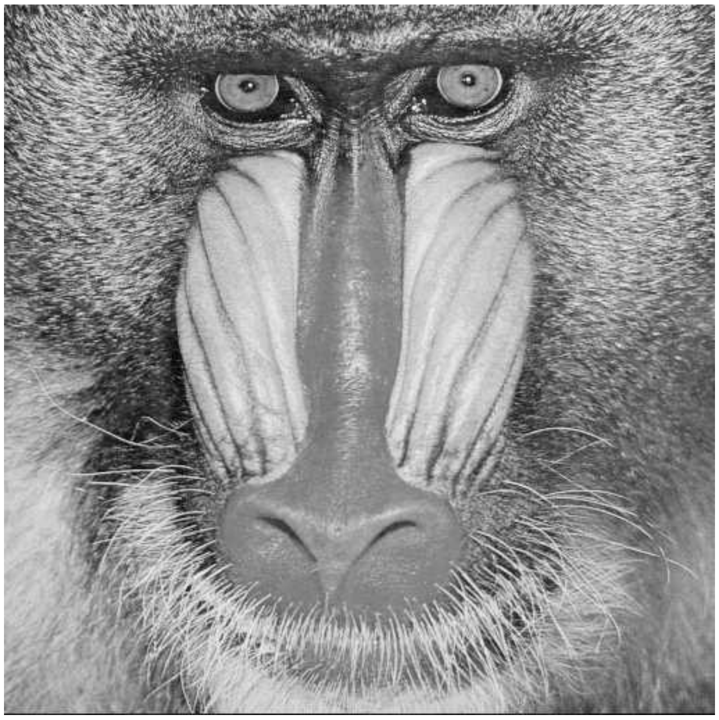
The images provided by the professor were effectively reproduced on a pass through our compressor (myDCT.c) and decompressor (myIDCT.c). We chose few other images to test our code on real world images like an image of a woman, a baboon , our favorite Friends' characters, few flowers, etc. Few of these are attached below.

Baboon

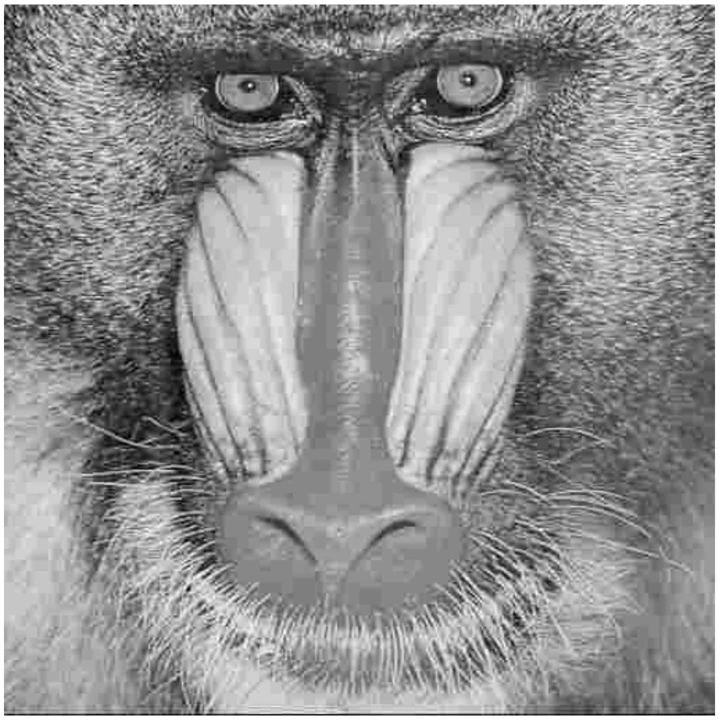
Original baboon image: It is a 512 x512 image.



Output with Qfactor =1.0:



Output with Qfactor = 4.5

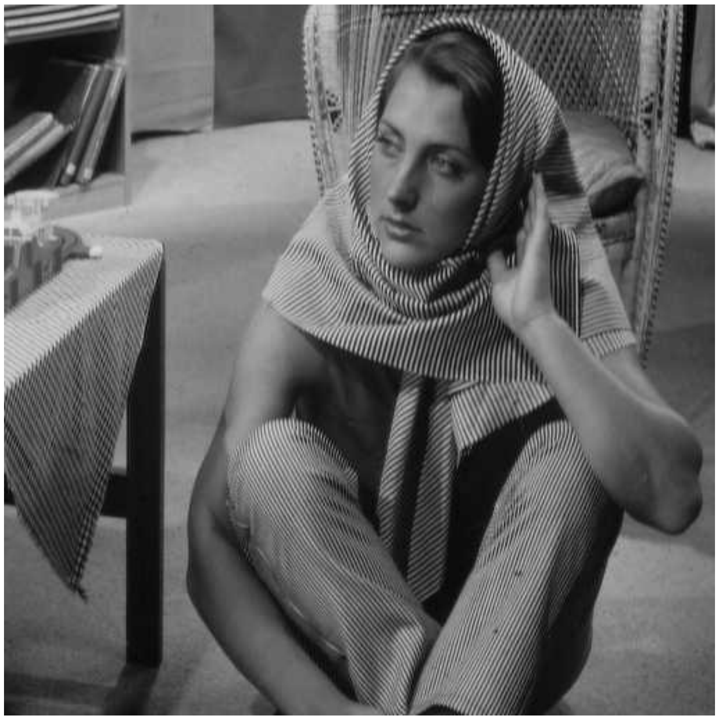


The difference between original image and that at quantization factor of 4.5 can be easily seen at the nose of the animal.

Woman

Original woman image: 512x512

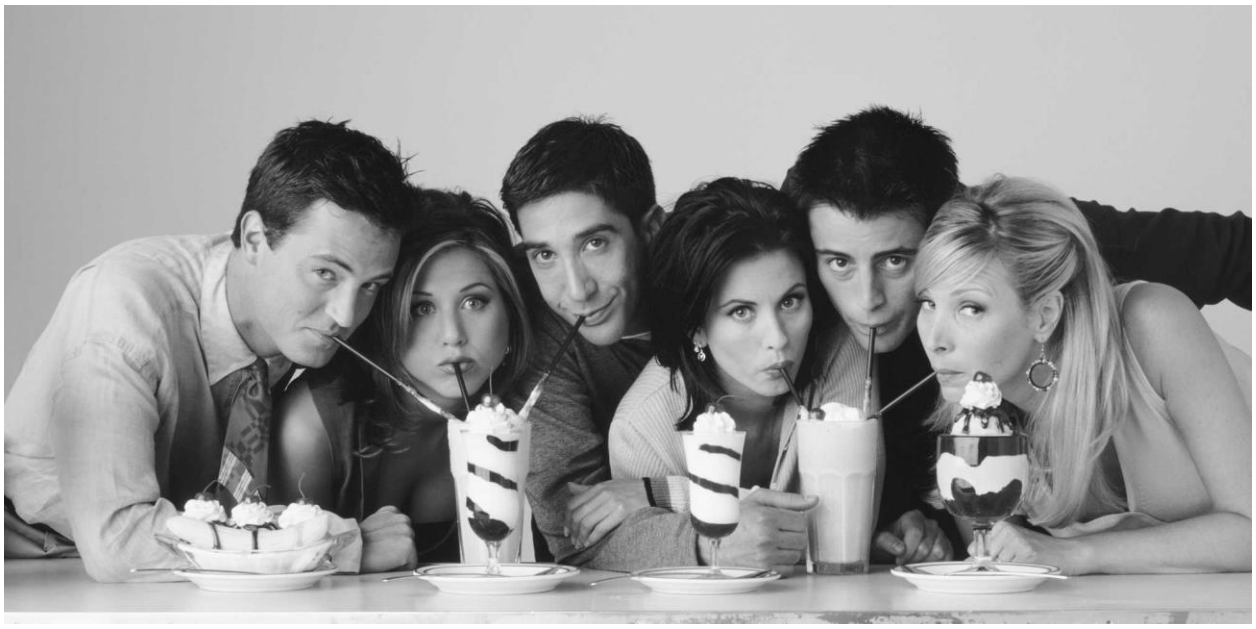
Output file with quantization factor =1.0:



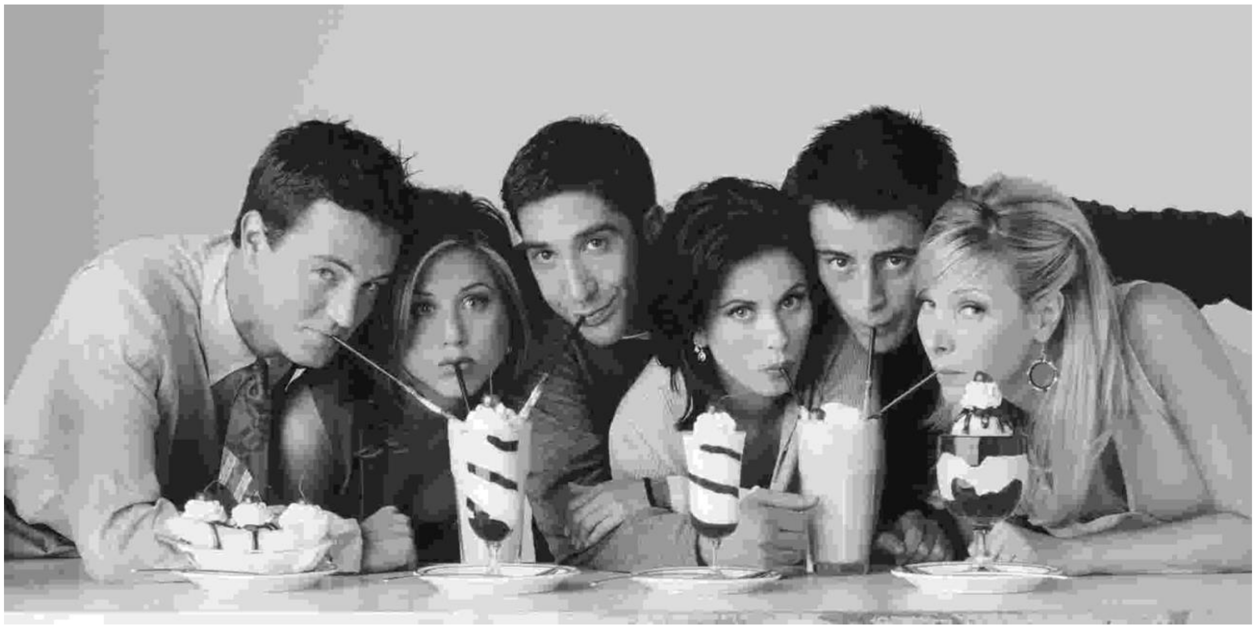
Friends

Original input friends image 2000x992

Output file with quantization factor =1:



Output file with quantization factor = 8.5



Conclusion

Testing was done on a variety of images. These can be classified according to 2 categories: the shape, i.e. square or rectangle and length, i.e xsize and ysize are a multiple of 16 or not.

According to our observations, our code works fine with images whose xsize and ysize are multiples of 16. However, since we have not implemented any method to abstract incomplete macroblocks (i.e. when xsize and ysize are not multiple of 16), our code ideally throws segmentation faults and other types of illegal access exceptions.

Our implementation seems to be working for both square and rectangle images whose size fit the requirement (i.e. xsize and ysize multiple of 16).

From the observations, we can see that the picture becomes hazy as the quantization factor increases. This is because of the lossy nature of use of quantization factor, resulting in loss of coefficients.

Appendix

Compressor: myDCT.c

```c
/*
Authors: Vidhya Lakshmi Venkatarama & Geetha Madhuri Chittuluri
This file contains the program for Compression of an image into JPEG form, till the zig-zag reorder stage.
*/

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

int quant[8][8];
double qScale;

// This function returns the value of Cu*Cv according to the value of u and v
double coeff (int u, int v) {
 double Cu =1.0, Cv =1.0;
double pdt;
    if (u==0)
      Cu = ((double)1.0)/(sqrt(2));
    if( v==0)
      Cv= ((double)1.0)/(sqrt(2));
    pdt = (Cu*Cv);
    return pdt;
}

// This function is used to print the dct output in zig zag reorder
void printZZ( FILE *oFp, int val, int itr)
{
        if (itr%8 ==0 && itr !=0)
            fprintf(oFp,"\n");
    fprintf(oFp,"%5d",val);

}
```

```c

// This function performs DCT, Quantization and calls the function to print the output in zig-zag reorder
void encode (FILE *oFp, int array[8][8]){
  //qt is the matrix that holds the final integer value of a pixel after DCT and quantization
  int u,v,i,j , qt[8][8];
  // matrix F represents the result of DCT.
  double F[8][8], Sum[8][8];
  for (v=0;v<8;v++){
     for(u=0;u<8;u++){
          Sum[u][v]=0.0;
          for (i=0;i<8;i++){
              for(j=0;j<8;j++){
                  Sum[u][v] = Sum[u][v] + (array[i][j] * cos((double)((((2*i) +1)* u* M_PI)/16))* cos((double)((((2*j) +1)* v* M_PI)/16)));
              }
          }
          F[u][v]= ((float)(coeff(u,v) * Sum[u][v]))/4;
          if(quant[u][v] !=0)
              qt[u][v] = (int)round(F[u][v]/(qScale*quant[u][v]));
          else
              qt[u][v]= (int)round(F[u][v]);
          // set the range of qt to be [0,255]
          if(qt[u][v]>128)
             qt[u][v]=128;
          if (qt[u][v] < -127)
             qt[u][v]= -127;
          qt[u][v] = qt[u][v]+127;
     }
  }
 // This part of the function zig-zag reorders the matrix qt for output to file

  int itr =0;
  u=0;
  v=0;
  printZZ(oFp, qt[u][v], itr);
  itr++;
  while ( u<8 && v<8){
    v++;
    if(u<8 && v<8){
      printZZ(oFp, qt[u][v], itr);
      itr++;
    }
    else if (u<7 && v>0){
      v--;
      u++ ;
      printZZ(oFp, qt[u][v], itr);
      itr++;
    }
    else break;
    while (v>0 && u<7 && v<8){
      v--;
      u++;
      printZZ(oFp, qt[u][v], itr);
      itr++;
    }
    u++;
    if(u<8 && v<8){
      printZZ(oFp, qt[u][v], itr);
```

```c
        itr++;
    }
    else if (u>0 && v<7) {
        u--;
        v++ ;
        printZZ(oFp, qt[u][v], itr);
        itr++;
    }
    while(u>0 && u<8 && v<7){
            u--;
            v++;
            printZZ(oFp, qt[u][v], itr);
            itr++;
        }
    }
    fprintf(oFp,"\n");
}

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

    FILE *iImg, *qFile,  *oFp;
    iImg = fopen(argv[1], "r");  // input image file
    qFile = fopen(argv[2],"r"); // Quantization file
    qScale = atof(argv[3]);  // The value of quantization factor


    if (iImg == NULL) {
        fprintf(stderr, "Cannot open input file %s",argv[1]);
        exit(1);
    }

    if (qFile == NULL) {
        fprintf(stderr, "Cannot open input file %s",argv[2]);
        exit(1);
    }

    int count, i,j;
    // read in the quantization matrix
    for (i =0; i< 8; i++){
        for (j=0; j<8; j++){
            fscanf(qFile, "%d", &quant[i][j]);
        }
    }
    fclose(qFile);

    int xsize, ysize, maxVal;
    char line[40];
    // read in the header of the input pgm file.
    fscanf(iImg, "%s", line);
    fscanf(iImg, "%d" "%d", &xsize, &ysize);
    fscanf(iImg, "%d", &maxVal);
    // y corresponds to the row (vertical variation) and x corresponds to column

    int input[ysize][xsize];
    int c;
    // read the characters in the pgm file, convert them to corresponding ASCII code
    // and save it in the matrix for input image pixels
    c = fgetc(iImg);
    for (i=0; i<ysize; i++){
            for (j =0; j<xsize;j++){
                c = fgetc(iImg);
                unsigned char ch = (unsigned char) c;
                input[i][j] = ch;
            }
        }
        fclose(iImg);

        // open the output file and write its header
        oFp = fopen(argv[4],"w");
        if(oFp ==NULL) {
              fprintf(stderr, "Cannot open output file %s", argv[3]);
              exit(1);
          }
        char *str = "MYDCT\n" ;
        fprintf(oFp, "%s%d %d\n%lf\n", str,xsize,ysize,qScale);

        // a 16X16 macroblock 'a' is split into 4 8X8 macroblocks: a1, a2,a3,a4
        int a[16][16], a1[8][8], a2[8][8], a3[8][8], a4[8][8], k,l;
            for (k=0;k<ysize;k=k+16){
                for (l=0;l<xsize; l=l+16){
                for (i=0; i<16; i++){
                    for (j=0; j<16; j++){
                        a[i][j]= input[k+i][l+j];
                    }
               }
               for (i=0; i<8; i++){
                    for (j=0; j<8; j++){
                        a1[i][j] = a[i][j];
                        a3[i][j] = a[i+8][j];
                        a2[i][j] = a[i][j+8];
                        a4[i][j] = a[i+8][j+8];
                   }
               }
               // encode each of the four 8X8 blocks with the help of encode function
                fprintf(oFp,"%d %d\n", l,k);
                encode(oFp,a1);
                fprintf(oFp,"%d %d\n", l+8, k);
                encode(oFp,a2);
                fprintf(oFp,"%d %d\n", l,k+8);
                encode(oFp,a3);
                fprintf(oFp,"%d %d\n", l+8,k+8);
                encode(oFp,a4);

            }
      }
        fclose(oFp);
    return 0;
    }
```

Decompressor: myIDCT.c

```
/*
Authors: Vidhya Lakshmi Venkatarama & Geetha Madhuri Chittuluri
This file contains the program for De-
compression of an image in the form of discrete cosine transformed, quantised and zig-
zag reordered.
*/


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


// This function returns the value of Cu*Cv according to the value of u and v
double coeff (int u, int v) {
double Cu =1.0, Cv =1.0;
double pdt;
    if (u==0)
      Cu = ((double)1.0)/(sqrt(2));
    if( v==0)
      Cv= ((double)1.0)/(sqrt(2));
    pdt = (Cu*Cv);
    return pdt;
}

int quant[8][8];
double qScale;
int main (int argc, char ** argv){
    FILE * iImg, *qFile, *oFp;
    iImg = fopen(argv[1], "r");        // open input image file read only mode
       qFile = fopen(argv[2],"r");     // open quantization file read only mode
    oFp = fopen(argv[3],"w");          // open output file write only mode
    if (iImg == NULL) {
          fprintf(stderr, "Cannot open input file %s",argv[1]);
          exit(1);
      }
    if (qFile == NULL) {
          fprintf(stderr, "Cannot open input file %s",argv[2]);
          exit(1);
    }
    if (oFp == NULL) {
          fprintf(stderr, "Cannot open input file %s",argv[2]);
          exit(1);
    }

    int count, i,j;
    // read and save quantization matrix in quant
    for (i =0; i< 8; i++){
      for (j=0; j<8; j++){
        fscanf(qFile, "%d", &quant[i][j]);
      }
    }
    fclose(qFile);

    int xsize, ysize, maxVal=255;
    char line[40];
    // read header of input dct file
    fscanf(iImg, "%s", line);
    fscanf(iImg, "%d" "%d", &xsize, &ysize);
    fscanf(iImg, "%lf", &qScale);
    // write the header to teh output pgm file
    fprintf(oFp,"%s\n%d %d\n%d\n", "P5", xsize, ysize, maxVal);
    int qt[8][8],fun[8][8];
    int output[ysize][xsize];
    double F[8][8], Sum[8][8];
    int c=0, m,n,u,v;

    //loop for all input and variable c is the loop variable
    while(c<(xsize*ysize/256*4)) {
        //read the starting points (pixel) of the 8X* blocks
        fscanf(iImg, "%d %d", &n, &m);

        // read and unzig-zag the input matrix into qt
        u=0,v=0;
        fscanf(iImg, "%d", &qt[u][v]);
        while ( u<8 && v<8){
            v++;
            if(u<8 && v<8){
                fscanf(iImg,"%d", &qt[u][v]);
            }
            else if (u<7 && v>0){
                v--;
                u++ ;
                fscanf(iImg,"%d", &qt[u][v]);
            }
            else break;
            while (v>0 && u<7 && v<8){
                v--;
                u++;
                fscanf(iImg,"%d", &qt[u][v]);
            }
            u++;
            if(u<8 && v<8){
                fscanf(iImg,"%d", &qt[u][v]);
            }
            else if (u>0 && v<7) {
                u--;
                v++ ;
                fscanf(iImg,"%d", &qt[u][v]);
            }
                  while(u>0 && u<8 && v<7){
                      u--;
                      v++;
                      fscanf(iImg,"%d", &qt[u][v]);
                  }
              }
              c++;
              // multiplying the coefficients with the quantization matrix values and quantization factor
              for (v =0; v<8; v++){
                  for (u=0; u<8; u++){
                      qt[v][u] = qt[v][u] -127;
                      F[v][u] = (float)qt[v][u] * qScale* quant[v][u];
                  }
              }

            // code for IDCT
            for (i=0; i<8; i++){
                for (j =0; j<8;j++){
                    Sum[i][j]= 0;
                    for (v=0; v<8; v++){
                        for(u=0; u<8;u++){
                            if(F[v][u] != 0){
                                Sum[i][j] = Sum[i][j]+ (F[v][u] *coeff(u,v)* cos((double)((((2*i) +1)* v* M_PI)/16))* cos((double)((((2*j) +1)* u* M_PI)/16)));// summations inside the IDCT formula
                            }
                        }
                    }
                    // Result of IDCT
                    fun[i][j] =  (int)round(Sum[i][j]/4);
                // set the range of function value to be [-127,128]
                    if (fun[i][j]>255)
                        fun[i][j]=255;
                    if (fun[i][j]<0)
                        fun[i][j] =0;
                    output[i+m][j+n]= fun[i][j];
                }
            }
        }
        // writing out the 8 bit per pixel values of teh entire image to the pgm file
        for(i=0; i<ysize;i++){
            for (j=0; j<xsize; j++){
                char ch = output[i][j];
                fputc(ch, oFp);
            }
        }

        fclose(iImg);
        fclose(oFp);
    }
```