```
// QM_CODE_SHANGLIY.cpp

#include "stdafx.h"
#include "fstream"
#include "iostream"
#include "vector"
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include "bitio.h"
#include "errhand.h"
#include <malloc.h>
#define PACIFIER_COUNT 2047
#define INF 2047
using namespace std;
errno_t err;

double test_count = 0;


BIT_FILE *OpenOutputBitFile(char *name)

{
	BIT_FILE *bit_file;

	bit_file = (BIT_FILE *)calloc(1, sizeof(BIT_FILE));
	if (bit_file == NULL)
		return(bit_file);
	if ((err = fopen_s(&bit_file->file, name, "wb")) != 0)
		printf("The Input file was not opened\n");
	else
		printf("The Input file was opened\n");

	bit_file->rack = 0;
	bit_file->mask = 0x80;
	bit_file->pacifier_counter = 0;
	return(bit_file);
}

void OutputBit(BIT_FILE *bit_file, int bit)

{
	if (bit)
		bit_file->rack |= bit_file->mask;
	bit_file->mask >>= 1;
	if (bit_file->mask == 0) {
		if (putc(bit_file->rack, bit_file->file) != bit_file->rack);
		// fatal_error( );
		else
			if ((bit_file->pacifier_counter++ & PACIFIER_COUNT) == 0)
				putc('.', stdout);
		bit_file->rack = 0;
```

```
                bit_file->mask = 0x80;
        }
}

void OutputBits(BIT_FILE *bit_file, unsigned long code, int count)
{
        unsigned long mask;

        mask = 1L << (count - 1);
        while (mask != 0) {
                if (mask & code)
                        bit_file->rack |= bit_file->mask;
                bit_file->mask >>= 1;
                if (bit_file->mask == 0) {
                        if (putc(bit_file->rack, bit_file->file) != bit_file->rack);
                        //fatal_error();
                        else if ((bit_file->pacifier_counter++ & PACIFIER_COUNT) == 0)
                                putc('.', stdout);
                        bit_file->rack = 0;
                        bit_file->mask = 0x80;
                }
                mask >>= 1;
        }
}

void CloseOutputBitFile(BIT_FILE *bit_file)

{
        if (bit_file->mask != 0x80)
                if (putc(bit_file->rack, bit_file->file) != bit_file->rack);
        //fatal_error();
        fclose(bit_file->file);
        free((char *)bit_file);
}



/*Huffman Function*/
/*-----------------------------------------------------------------------------*/
//the structure of Huffman root node
typedef vector<unsigned long> Sample_Code;  //the vector of code
                                                                        //Structure of the
code
class CODSYM
{
public:
        Sample_Code code;
        unsigned char name;
        CODSYM() { name = NULL; };
}CODE[256];

class HT_NODE
```

```
{
public:
	HT_NODE* left;
	HT_NODE* right;
	HT_NODE* parent;

	int name;
	double weight;
	int order;

	HT_NODE() { left = right = parent = NULL; name = 256; weight = 0; order = 0; };
	HT_NODE(HT_NODE* l, HT_NODE* r, HT_NODE* p, unsigned char n, double w, int o)
	{
		left = l;	right = r;	parent = p; name = n; weight = w; order = o;
	}
	~HT_NODE() { delete left; delete right; delete parent; }

};
//the vector of Huffman root
typedef vector<HT_NODE*> TreeVector;
TreeVector node_arr;
/*Re-sort the HUFFMAN array according to the weight*/
void sort_ARR(TreeVector &pro_data, double L)
{
	int i, j;

	HT_NODE* le;
	HT_NODE* ri;
	HT_NODE* pa;
	unsigned char na;
	double we;
	int ord;

	HT_NODE *temple;
	for (i = 0; i<L - 1; i++)
	{
		for (j = i; j<L; j++)
		{
			if ((pro_data[j]->weight)<(pro_data[i]->weight))
			{
				temple = pro_data[i];
				pro_data[i] = pro_data[j];
				pro_data[j] = temple;

			}
		}
	}
	return;
}
/*Generate the Huffman tree*/
void Build_tree(double len)
{
```

```
	int i;
	int j = 0;
	int a = 0;
	int b = 0;
	int N = 0;
	HT_NODE* node_par = new HT_NODE;

	for (i = 0; i<len; i++)
	{
		if (j == 2)  //Find two least weight node
			break;
		if (node_arr[i]->parent == NULL) //Not the root node
		{
			N++;
			switch (j)
			{
			case 0: a = i, j++; break;
			case 1: b = i, j++; break;
			}
		}

	}

	if (N != 0 && N != 1) //combine two least weight node
	{

		node_arr[a]->parent = node_par;
		node_arr[b]->parent = node_par;
		node_par->left = node_arr[a];
		node_par->right = node_arr[b];
		node_par->weight = node_arr[a]->weight + node_arr[b]->weight;
		node_arr.push_back(node_par);
		len++;
		sort_ARR(node_arr, len);  //sort new root array
		Build_tree(len);
		return;
	}
	else return;

}
/*Generate the Huffman CODE according to the root tree*/
void generate_code(HT_NODE &root, Sample_Code&scode)
{
	int i;

	if (((root.left) == NULL) && ((root.right) == NULL)) //Achieve the bottom node
	{

		(CODE[root.name]).code = scode;
		(CODE[root.name]).name = root.name;
		return;
```

```
        }

        Sample_Code lcode = scode;
        Sample_Code rcode = scode;
        lcode.push_back(false);
        rcode.push_back(true);

        generate_code(*root.left, lcode); //Left down generate code
        generate_code(*root.right, rcode);//Right down generate code
}

/*QE_TABLE along with states*/
int Qe[46] = { 0x59EB, 0x5522, 0x504F, 0x4B85, 0x4639, 0x415E, 0x3C3D, 0x375E, 0x32B4,
0x2E17,
                                        0x299A, 0x2516, 0x1EDF, 0x1AA9, 0x174E, 0x1424,
0x119C, 0x0F6B, 0x0D51, 0x0BB6,
                                        0x0A40, 0x0861, 0x0706, 0x05CD, 0x04DE, 0x040F,
0x0363, 0x02D4, 0x025C, 0x01F8,
                                        0x01A4, 0x0160, 0x0125, 0x00F6, 0x00CB, 0x00AB,
0x008F, 0x0068, 0x004E, 0x003B,
                                        0x002C, 0x001A, 0x000D, 0x0006, 0x0003, 0x0001
};
/*State changes rececving MPS symbol */
char mps_stchage[46] = {1,1,1,1,1,1,1,1,1,1,
                                                1,1,1,1,1,1,1,1,1,1,
                                                1,1,1,1,1,1,1,1,1,1,
                                                1,1,1,1,1,1,1,1,1,1,
                                                1,1,1,1,1,0
};
/*State changes rececving LPS symbol */
char lps_stchage[46] = { 'S',1,1,1,1,1,1,1,2,1,
                                                    2,1,1,2,1,2,1,2,2,1,
                                                2,2,2,2,1,2,2,2,2,2,
                                                2,2,2,2,2,1,2,2,2,2,
                                                    2,3,2,2,2,1
};

typedef vector<unsigned char> charVector;
charVector Code_arr; // The binary code after mapping
charVector QM_arr; // The Final code after compressing through QM code

typedef vector<int> BUFFVector;
BUFFVector buffvector;  // Input BUFF
BUFFVector SECbuffvector;
BUFFVector Disbuffvector;
BUFFVector planebuff[8];
BUFFVector decovector;
BIT_FILE *output_file;

unsigned long       C_register; // C register with 32 bits
unsigned long       A_register; // A register with 32 bits
```

```
unsigned long	SC;		// The number of stack
unsigned char		Outbuff;	// The output buffer with 8 bits
char				CT;		// Counting the number of symbol in the the buffer
unsigned char	MPS;		// The MPS symbol
int BPST;			// The Start points
int s;			// The state

void Initenc()
{
	A_register = 0x10000;
	C_register = 0;
	s = 0;
	CT = 11;
	MPS = 0;
	Outbuff = 0;
	BPST = 0;
}

void Stuff_0()
{
	if (Outbuff == 0xff)
	{
		QM_arr.push_back(Outbuff);
		Outbuff = 0;
	}
}

void Output_stacked_zeros()
{
	while (SC > 0)
	{
		QM_arr.push_back(Outbuff);
		Outbuff = 0;
		SC--;
	}
}

void Output_stacked_0xffs()
{
	while (SC > 0)
	{
		QM_arr.push_back(Outbuff);
		Outbuff = 0xff;
		QM_arr.push_back(Outbuff);
		Outbuff = 0;
		SC--;
	}
}

void Byte_out()
{
	unsigned t = C_register >> 19;
```

```
        if (t > 0xff)
        {
                Outbuff++;
                Stuff_0();
                Output_stacked_zeros();
                QM_arr.push_back(Outbuff);
                Outbuff = t;
        }
        else
        {
                if (t == 0xff)
                {
                        SC++;
                }
                else
                {
                        Output_stacked_0xffs();
                        QM_arr.push_back(Outbuff);
                        Outbuff = t;
                }
        }
        C_register &= 0x7ffff;
}

void Renorm()
{
        while (A_register < 0x8000)
        {
                A_register <<= 1;
                C_register <<= 1;
                CT--;

                if (CT == 0)
                {
                        Byte_out();
                        CT = 8;
                }
        }
}

void Code_LPS()
{
        A_register -= Qe[s];

        if (!(A_register < Qe[s]))
        {
                C_register += A_register;
                A_register = Qe[s];
        }

        s = s - lps_stchage[s];
```

```
	if (s == -'S')
	{
		MPS = 1 - MPS;
		s = 0;
	}

	Renorm();
}

void Code_MPS()
{
	A_register -= Qe[s];

	if (A_register < 0x8000)
	{
		if (A_register < Qe[s])
		{
			C_register += A_register;
			A_register = Qe[s];
		}
		s = s + mps_stchage[s];
		Renorm();
	}
}

void Clear_final_bits()
{
	unsigned long t;
	t = C_register + A_register - 1;
	t &= 0xffff0000;

	if (t < C_register) t += 0x8000;

	C_register = t;
}

void Discard_final_zeros()
{
	int i = 0;
	int flag = 0;
	for (i = QM_arr.size(); i >= 1; i--)
	{
		if (QM_arr[i-1] == 0)
			QM_arr.pop_back();
		else break;
	}
}

void Flush()
{
	Clear_final_bits();
```

```
	C_register <<= CT;
	Byte_out();
	C_register <<= 8;
	Byte_out();
	Discard_final_zeros();
	QM_arr.push_back(0xff);
	QM_arr.push_back(0xff);
}

int _tmain(int argc, _TCHAR* argv[])
{
	int i = 0;
	int k = 0;

	char file_ch;
	string file_name;
	HT_NODE node_root[512];
	int *Buff;
	int buff_pre = INF;
	double sam_weight[256] = { 0 };
	double sum_weight = 0;
	unsigned long code = 0;
	double ratio = 0;
	double out_count = 0;
	int TEXT_NUMBER[10] = { 1,1,0,0,0,1,0,0,1,0 };

	Sample_Code scode;



	int fun_tupe;


		printf("Choose which file to compress\n\n");
		printf("1:binary.dat 2:text.dat 3:audio.dat 4:image.dat\n\n");
		printf("Type in the index of file:");
		cin >> file_ch;

		if (file_ch == '5')  return 0;
		switch (file_ch)
		{
		case '1':file_name = "binary.dat";
			break;
		case '2':file_name = "text.dat";
			break;
		case '3':file_name = "audio.dat";
			break;
		case '4':file_name = "image.dat";
			break;
		default: cerr << "No choose of the file" << endl; abort();
		}
```

```
printf("Choose which mapping function \n\n");
printf("1:Basic binary map 2:Plane binary mapping 3:Huffman code\n\n");
printf("Type in the index of function:");
cin >> fun_tupe;

if (fun_tupe == 1)//Mapping function 1 : Byte to bit
{
	Initenc();  //Make initialization for the QM code
	int value = 0;
	unsigned char mask = 0x80;;

	Buff = (int*)calloc(sizeof(int), 1);
	ifstream infile(file_name, ios::binary);
	if (!infile)
	{
		cerr << "open error!" << endl;
		abort();
	}


	while (infile.peek() != EOF)
	{
		infile.read((char*)Buff, sizeof(char));
		mask = 0x80;
		for (i = 0; i < 8; i++)
		{
			value = (*Buff & mask ? 1 : 0);
			mask >>= 1;
			if (value == MPS)
			{
				Code_MPS();  //Recieve the MPS symbol
			}
			else Code_LPS(); //Recieve the LPS symbol
		}
		sam_weight[*Buff]++;
		sum_weight++;
	}
	Flush();

}

if (fun_tupe == 2)//Mapping function 2 : Plane Bite Slicing
{
	Initenc();  //Make initialization for the QM code
	int value = 0;
	int N = 0;
	unsigned char mask = 0x80;;
	ifstream infile(file_name, ios::binary);
	if (!infile)
	{
		cerr << "open error!" << endl;
```

```
                abort();
        }
        if (fun_tupe == 4) {
                Buff = (int*)calloc(sizeof(int), 1);
                while (infile.peek() != EOF)
                {
                        infile.read((char*)Buff, sizeof(char));
                        buffvector.push_back(*Buff);
                }
                /*Zigzag process*/
                for (N = 0; N <= 255; N++)
                {
                        for (i = 0; i <= N; i++)
                        {
                                SECbuffvector.push_back(buffvector[i * 256 + (N -
i)]);

                        }
                        N++;
                        for (i = N; i >= 0; i--)
                        {
                                SECbuffvector.push_back(buffvector[i * 256 + (N -
i)]);

                        }
                }

                for (N = 256; N <= 510; N++)
                {
                        for (i = N - 255; i < 256; i++)
                        {
                                SECbuffvector.push_back(buffvector[i * 256 + (N -
i)]);

                        }
                        N++;
                        for (i = 255; i >= N - 255; i--)
                        {
                                SECbuffvector.push_back(buffvector[i * 256 + (N -
i)]);

                        }
                }

        }
        else {
                Buff = (int*)calloc(sizeof(int), 1);
                while (infile.peek() != EOF)
                {
                        infile.read((char*)Buff, sizeof(char));
                        SECbuffvector.push_back(*Buff);
                }

        }
```

```
        int j = 0;

        for (j = 0; j < SECbuffvector.size();j++)
        {
                *Buff= SECbuffvector[j];
                mask = 0x80;
                for (i = 0; i < 8; i++)
                {
                        value = (*Buff & mask ? 1 : 0);
                        mask >>= 1;
                        planebuff[i].push_back(value);
                }

                sam_weight[*Buff]++;
                sum_weight++;
        }

        for (j = 0; j < 8; j++)
        {
                for (i = 0; i < planebuff[j].size(); i++)
                {
                        if (planebuff[j][i] == MPS)
                        {
                                Code_MPS();  //Recieve the MPS symbol
                        }
                        else Code_LPS(); //Recieve the LPS symbol
                }
        }
        Flush();
}


if (fun_tupe == 3)//Mapping function 3 : Huffman
{
        Initenc();  //Make initialization for the QM code
        ifstream infile(file_name, ios::binary);
        if (!infile)
        {
                cerr << "open error!" << endl;
                abort();
        }

        Buff = (int*)calloc(sizeof(int), 1);
        while (infile.peek() != EOF)
        {
                infile.read((char*)Buff, sizeof(char));
                sam_weight[*Buff]++;
                sum_weight++;

        }

        //Push data into vector
```

```
		for (i = 0; i<256; i++)
		{
			if (sam_weight[i]) {
				node_arr.push_back(new  HT_NODE(NULL, NULL, NULL,
(unsigned char)i, sam_weight[i], k++));
			}
		}
		sort_ARR(node_arr, node_arr.size());
		Build_tree(node_arr.size());  //Build huffman tree
		generate_code(*node_arr[node_arr.size() - 1], scode);  //generate code
		node_arr.clear();

		/*Rescan the input files and generate code i.e map the sample to binary
sequence */
		/*Do QM coed for each symbol*/
		infile.clear();
		infile.seekg(0);    //Return to the top of the input files
		while (infile.peek() != EOF)
		{
			infile.read((char*)Buff, sizeof(char));
			for (i = 0; i < CODE[(*Buff)].code.size(); i++)  //map procedure
			{
				test_count++;
				if (CODE[(*Buff)].code[i] == MPS)
				{
					Code_MPS();  //Recieve the MPS symbol
				}
				else Code_LPS(); //Recieve the LPS symbol
			}
		}
		infile.close();
		Flush();
	}
	if (fun_tupe == 4) //written question
	{
		Initenc();
		s = 10;
		for (i = 0; i < 10; i++)  //map procedure
		{
			test_count++;
			if (TEXT_NUMBER[i] == MPS)
			{
				Code_MPS();  //Recieve the MPS symbol
			}

			else Code_LPS(); //Recieve the LPS symbol
		}
	}


	BIT_FILE *bit_file;
	bit_file = OpenOutputBitFile("binary_stream.dat"); //Build the binary stream file
```

```
		for (i = 1; i < QM_arr.size(); i++)  //map procedure
		{
			OutputBits(bit_file, QM_arr[i], 8);
			out_count++;
		}
		CloseOutputBitFile(bit_file);
		printf("The size of output file is %d Bytes \n\n", QM_arr.size() - 1);



	system("pause");
	delete Buff;
	return 0;
}

// QM_CODE_SHANGLIY.cpp

#include "stdafx.h"
#include "fstream"
#include "iostream"
#include "vector"
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include "bitio.h"
#include "errhand.h"
#include <malloc.h>
#include "qmcoder.h"

#define QMputc(BP, m_File)		if (bFirst) {fputc(BP, m_File);} else {bFirst = 1;};

#define PACIFIER_COUNT 2047
#define INF 2047
using namespace std;
errno_t err;

double test_count = 0;


BIT_FILE *OpenOutputBitFile(char *name)

{
	BIT_FILE *bit_file;

	bit_file = (BIT_FILE *)calloc(1, sizeof(BIT_FILE));
	if (bit_file == NULL)
		return(bit_file);
	if ((err = fopen_s(&bit_file->file, name, "wb")) != 0)
		cout<<"The Input file was not opened\n";
	else
		cout << "The Input file was opened\n";
```

```
        bit_file->rack = 0;
        bit_file->mask = 0x80;
        bit_file->pacifier_counter = 0;
        return(bit_file);
}

void OutputBit(BIT_FILE *bit_file, int bit)

{
        if (bit)
                bit_file->rack |= bit_file->mask;
        bit_file->mask >>= 1;
        if (bit_file->mask == 0) {
                if (putc(bit_file->rack, bit_file->file) != bit_file->rack);
                // fatal_error( );
                else
                        if ((bit_file->pacifier_counter++ & PACIFIER_COUNT) == 0)
                                putc('.', stdout);
                bit_file->rack = 0;
                bit_file->mask = 0x80;
        }
}

void OutputBits(BIT_FILE *bit_file, unsigned long code, int count)
{
        unsigned long mask;

        mask = 1L << (count - 1);
        while (mask != 0) {
                if (mask & code)
                        bit_file->rack |= bit_file->mask;
                bit_file->mask >>= 1;
                if (bit_file->mask == 0) {
                        if (putc(bit_file->rack, bit_file->file) != bit_file->rack);
                        //fatal_error();
                        else if ((bit_file->pacifier_counter++ & PACIFIER_COUNT) == 0)
                                putc('.', stdout);
                        bit_file->rack = 0;
                        bit_file->mask = 0x80;
                }
                mask >>= 1;
        }
}

void CloseOutputBitFile(BIT_FILE *bit_file)

{
        if (bit_file->mask != 0x80)
                if (putc(bit_file->rack, bit_file->file) != bit_file->rack);
        //fatal_error();
        fclose(bit_file->file);
        free((char *)bit_file);
```

```
}



/*QE_TABLE along with states*/
int Qe[46] = { 0x59EB, 0x5522, 0x504F, 0x4B85, 0x4639, 0x415E, 0x3C3D, 0x375E, 0x32B4,
0x2E17,
0x299A, 0x2516, 0x1EDF, 0x1AA9, 0x174E, 0x1424, 0x119C, 0x0F6B, 0x0D51, 0x0BB6,
0x0A40, 0x0861, 0x0706, 0x05CD, 0x04DE, 0x040F, 0x0363, 0x02D4, 0x025C, 0x01F8,
0x01A4, 0x0160, 0x0125, 0x00F6, 0x00CB, 0x00AB, 0x008F, 0x0068, 0x004E, 0x003B,
0x002C, 0x001A, 0x000D, 0x0006, 0x0003, 0x0001
};
/*State changes rececving MPS symbol */
char mps_stchage[46] = { 1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,0
};
/*State changes rececving LPS symbol */
char lps_stchage[46] = { 'S',1,1,1,1,1,1,2,1,
2,1,1,2,1,2,1,2,2,1,
2,2,2,2,1,2,2,2,2,2,
2,2,2,2,2,1,2,2,2,2,
2,3,2,2,2,1
};
int lsz[256] = {
        0x5a1d, 0x2586, 0x1114, 0x080b, 0x03d8,
        0x01da, 0x0015, 0x006f, 0x0036, 0x001a,
        0x000d, 0x0006, 0x0003, 0x0001, 0x5a7f,
        0x3f25, 0x2cf2, 0x207c, 0x17b9, 0x1182,
        0x0cef, 0x09a1, 0x072f, 0x055c, 0x0406,
        0x0303, 0x0240, 0x01b1, 0x0144, 0x00f5,
        0x00b7, 0x008a, 0x0068, 0x004e, 0x003b,
        0x002c, 0x5ae1, 0x484c, 0x3a0d, 0x2ef1,
        0x261f, 0x1f33, 0x19a8, 0x1518, 0x1177,
        0x0e74, 0x0bfb, 0x09f8, 0x0861, 0x0706,
        0x05cd, 0x04de, 0x040f, 0x0363, 0x02d4,
        0x025c, 0x01f8, 0x01a4, 0x0160, 0x0125,
        0x00f6, 0x00cb, 0x00ab, 0x008f, 0x5b12,
        0x4d04, 0x412c, 0x37d8, 0x2fe8, 0x293c,
        0x2379, 0x1edf, 0x1aa9, 0x174e, 0x1424,
        0x119c, 0x0f6b, 0x0d51, 0x0bb6, 0x0a40,
        0x5832, 0x4d1c, 0x438e, 0x3bdd, 0x34ee,
        0x2eae, 0x299a, 0x2516, 0x5570, 0x4ca9,
        0x44d9, 0x3e22, 0x3824, 0x32b4, 0x2e17,
        0x56a8, 0x4f46, 0x47e5, 0x41cf, 0x3c3d,
        0x375e, 0x5231, 0x4c0f, 0x4639, 0x415e,
        0x5627, 0x50e7, 0x4b85, 0x5597, 0x504f,
        0x5a10, 0x5522, 0x59eb
};
```

```
int nlps[256] = {
        1, 14, 16, 18, 20,   23, 25, 28, 30, 33,
        35,  9, 10, 12, 15,   36, 38, 39, 40, 42,
        43, 45, 46, 48, 49,   51, 52, 54, 56, 57,
        59, 60, 62, 63, 32,   33, 37, 64, 65, 67,
        68, 69, 70, 72, 73,   74, 75, 77, 78, 79,
        48, 50, 50, 51, 52,   53, 54, 55, 56, 57,
        58, 59, 61, 61, 65,   80, 81, 82, 83, 84,
        86, 87, 87, 72, 72,   74, 74, 75, 77, 77,
        80, 88, 89, 90, 91,   92, 93, 86, 88, 95,
        96, 97, 99, 99, 93,   95,101,102,103,104,
        99,105,106,107,103,  105,108,109,110,111,
        110,112,112
};

int nmps[256] = {
        1,  2,  3,  4,  5,    6,  7,  8,  9, 10,
        11, 12, 13, 13, 15,   16, 17, 18, 19, 20,
        21, 22, 23, 24, 25,   26, 27, 28, 29, 30,
        31, 32, 33, 34, 35,    9, 37, 38, 39, 40,
        41, 42, 43, 44, 45,   46, 47, 48, 49, 50,
        51, 52, 53, 54, 55,   56, 57, 58, 59, 60,
        61, 62, 63, 32, 65,   66, 67, 68, 69, 70,
        71, 72, 73, 74, 75,   76, 77, 78, 79, 48,
        81, 82, 83, 84, 85,   86, 87, 71, 89, 90,
        91, 92, 93, 94, 86,   96, 97, 98, 99,100,
        93,102,103,104, 99,  106,107,103,109,107,
        111,109,111
};

int swit[256] = {
        1,0,0,0,0,    0,0,0,0,0,
        0,0,0,0,1,    0,0,0,0,0,
        0,0,0,0,0,    0,0,0,0,0,
        0,0,0,0,0,    0,1,0,0,0,
        0,0,0,0,0,    0,0,0,0,0,
        0,0,0,0,0,    0,0,0,0,0,
        0,0,0,0,1,    0,0,0,0,0,
        0,0,0,0,0,    0,0,0,0,0,
        1,0,0,0,0,    0,0,0,1,0,
        0,0,0,0,0,    1,0,0,0,0,
        0,0,0,0,0,    1,0,0,0,0,
        1,0,1
};


typedef vector<unsigned char> charVector;
charVector Code_arr; // The binary code after mapping
charVector QM_arr; // The Final code after compressing through QM code

typedef vector<int> BUFFVector;
BUFFVector buffvector;  // Input BUFF
```

```
BUFFVector SECbuffvector;
BUFFVector planebuff[8];
BUFFVector decovector;
BIT_FILE *output_file;

unsigned long       C_register; // C register with 32 bits
unsigned long       A_register; // A register with 32 bits
unsigned long       SC;         // The number of stack
unsigned char            Outbuff;    // The output buffer with 8 bits
char                          CT;         // Counting the number of symbol in the the buffer
unsigned char       MPS;        // The MPS symbol
int BPST;                   // The Start points
int s;                      // The state



QM::QM(FILE *FP)
{
        m_File = FP;
        max_context = 4096;
        st_table = (unsigned char *)calloc(max_context, sizeof(unsigned char));
        mps_table = (unsigned char *)calloc(max_context, sizeof(unsigned char));
}

void QM::StartQM()
{

                sc = 0;
                A_interval = 0x10000;
                C_register = 0;
                ct = 11;

                count = -1;
                debug = 0;
                BP = 0;
                bFirst = 0;

}


QM::~QM()
{
        free(st_table);
        free(mps_table);
}


void
QM::reset()
{
        for (int i = 0; i < max_context; i++)
        {
```

```
                st_table[i] = 0;
                mps_table[i] = 0;
        }
}

void
QM::encode(unsigned char symbol, int context)
{
        if (this->debug) cout << (char)(symbol + '0') << " " << context << endl;

        if (context >= max_context)
        {
                unsigned char *new_st, *new_mps;
                new_st = (unsigned char *)calloc(max_context * 2, sizeof(unsigned char));
                new_mps = (unsigned char *)calloc(max_context * 2, sizeof(unsigned char));
                memcpy(new_st, st_table, max_context*sizeof(unsigned char));
                memcpy(new_mps, mps_table, max_context*sizeof(unsigned char));
                max_context *= 2;
                free(st_table);
                free(mps_table);
                st_table = new_st;
                mps_table = new_mps;
        }

        next_st = cur_st = st_table[context];
        next_MPS = MPS = mps_table[context];
        Qe = lsz[st_table[context]];

        if (MPS == symbol)
                Code_MPS();
        else
                Code_LPS();

        st_table[context] = next_st;
        mps_table[context] = next_MPS;
};

void
QM::encode(unsigned char symbol, int prob, int mps_symbol)
{
        if (this->debug) cout << (char)(symbol + '0') << " " << prob << endl;

        next_st = cur_st = 0;
        next_MPS = MPS = mps_symbol;
        Qe = prob;

        if (MPS == symbol)
                Code_MPS();
        else
                Code_LPS();
};
```

```
void
QM::Flush()
{
	Clear_final_bits();
	C_register <<= ct;
	Byte_out();
	C_register <<= 8;
	Byte_out();
	QMputc(BP, m_File);
	QMputc(0xff, m_File); count++;
	QMputc(0xff, m_File); count++;
}


unsigned char
QM::decode(int context)
{
	if (context >= max_context)
	{
		unsigned char *new_st, *new_mps;
		new_st = (unsigned char *)calloc(max_context * 2, sizeof(unsigned char));
		new_mps = (unsigned char *)calloc(max_context * 2, sizeof(unsigned char));
		memcpy(new_st, st_table, max_context*sizeof(unsigned char));
		memcpy(new_mps, mps_table, max_context*sizeof(unsigned char));
		max_context *= 2;
		free(st_table);
		free(mps_table);
		st_table = new_st;
		mps_table = new_mps;
	}
	next_st = cur_st = st_table[context];
	next_MPS = MPS = mps_table[context];
	Qe = lsz[st_table[context]];
	unsigned char ret_val = AM_decode_Symbol();
	st_table[context] = next_st;
	mps_table[context] = next_MPS;

	if (this->debug) cout << (char)(ret_val + '0') << " " << context << endl;
	return ret_val;
};


unsigned char
QM::decode(int prob, int mps_symbol)
{
	next_st = cur_st = 0;
	next_MPS = MPS = mps_symbol;
	Qe = prob;
	unsigned char ret_val = AM_decode_Symbol();

	if (this->debug) cout << (char)(ret_val + '0') << " " << prob << endl;
	return ret_val;
```

```
};


void
QM::Code_LPS()
{
	A_interval -= Qe;

	if (!(A_interval < Qe))
	{
		C_register += A_interval;
		A_interval = Qe;
	}

	if (swit[cur_st] == 1)
	{
		next_MPS = 1 - MPS;
	}
	next_st = nlps[cur_st];

	Renorm_e();
};


void
QM::Code_MPS()
{
	A_interval -= Qe;

	if (A_interval < 0x8000)
	{
		if (A_interval < Qe)
		{
			C_register += A_interval;
			A_interval = Qe;
		}
		next_st = nmps[cur_st];
		Renorm_e();
	}
}


void
QM::Renorm_e()
{
	while (A_interval < 0x8000)
	{
		A_interval <<= 1;
		C_register <<= 1;
		ct--;

		if (ct == 0)
```

```
                {
                        Byte_out();
                        ct = 8;
                }
        }
}


void
QM::Byte_out()
{
        unsigned t = C_register >> 19;

        if (t > 0xff)
        {
                BP++;
                Stuff_0();
                Output_stacked_zeros();
                QMputc(BP, m_File); count++;
                BP = t;
        }
        else
        {
                if (t == 0xff)
                {
                        sc++;
                }
                else
                {
                        Output_stacked_0xffs();
                        QMputc(BP, m_File); count++;
                        BP = t;
                }
        }
        C_register &= 0x7ffff;
}


void
QM::Output_stacked_zeros()
{
        while (sc > 0)
        {
                QMputc(BP, m_File); count++;
                BP = 0;
                sc--;
        }
}


void
QM::Output_stacked_0xffs()
```

```
{
	while (sc > 0)
	{
		QMputc(BP, m_File); count++;
		BP = 0xff;
		QMputc(BP, m_File); count++;
		BP = 0;
		sc--;
	}
}


void
QM::Stuff_0()
{
	if (BP == 0xff)
	{
		QMputc(BP, m_File); count++;
		BP = 0;
	}
}


void
QM::Clear_final_bits()
{
	unsigned long t;
	t = C_register + A_interval - 1;
	t &= 0xffff0000;

	if (t < C_register) t += 0x8000;

	C_register = t;
}


unsigned char
QM::AM_decode_Symbol()
{
	unsigned char D;

	A_interval -= Qe;

	if (Cx < A_interval)
	{
		if (A_interval < 0x8000)
		{
			D = Cond_MPS_exchange();
			Renorm_d();
		}
		else
			D = MPS;
```

```
	}
	else
	{
		D = Cond_LPS_exchange();
		Renorm_d();
	}

	return D;
}


unsigned char
QM::Cond_LPS_exchange()
{
	unsigned char D;
	unsigned  C_low;


	if (A_interval < Qe)
	{
		D = MPS;
		Cx -= A_interval;
		C_low = C_register & 0x0000ffff;

		C_register = ((unsigned long)Cx << 16) + (unsigned long)C_low;
		A_interval = Qe;
		next_st = nmps[cur_st];
	}
	else
	{
		D = 1 - MPS;
		Cx -= A_interval;
		C_low = C_register & 0x0000ffff;
		C_register = ((unsigned long)Cx << 16) + (unsigned long)C_low;
		A_interval = Qe;

		if (swit[cur_st] == 1)
		{
			next_MPS = 1 - MPS;
		}
		next_st = nlps[cur_st];
	}

	return D;
}


unsigned char
QM::Cond_MPS_exchange()
{
	unsigned char D;
```

```
	if (A_interval < Qe)
	{
		D = 1 - MPS;
		if (swit[cur_st] == 1)
		{
			next_MPS = 1 - MPS;
		}
		next_st = nlps[cur_st];
	}
	else
	{
		D = MPS;
		next_st = nmps[cur_st];
	}

	return D;
}


void
QM::Renorm_d()
{
	while (A_interval<0x8000)
	{
		if (ct == 0)
		{
			if (bEnd == 0) Byte_in();
			ct = 8;
		}
		A_interval <<= 1;
		C_register <<= 1;
		ct--;
	}

	Cx = (unsigned)((C_register & 0xffff0000) >> 16);
};


void
QM::Byte_in()
{
	unsigned char B;
	B = fgetc(m_File), count++;

	if (B == 0xff)
	{
		Unstuff_0();
	}
	else
	{
		C_register += (unsigned)B << 8;
	}
```

```
};


void
QM::Unstuff_0()
{
	unsigned char B;
	B = fgetc(m_File), count++;

	if (B == 0)
	{
		C_register |= 0xff00;
	}
	else
	{
		if (B == 0xff)
		{
			//cerr << "\nEnd marker has been met!\n";
			bEnd = 1;
		}
	}
}


int QM::Counting()
{
	if (ct == 0)
	{
		return count * 8;
	}
	else
	{
		return count * 8 + 8 - ct;
	}
}



int _tmain(int argc, _TCHAR* argv[])
{
	int i = 0;
	int k = 0;

	char file_ch;
	const char* file_name;
	const char* DIRECTION="encode";
	int *Buff;
	int buff_pre = INF;
	double sam_weight[256] = { 0 };
	double sum_weight = 0;
	unsigned long code = 0;
	double ratio = 0;
```

```
double out_count = 0;
int TEXT_NUMBER[10] = { 1,1,0,0,0,1,0,0,1,0 };
FILE *output_file;
int file_start = 0;

QM *qm;
unsigned char value = 0;
unsigned char mask = 0x80;
int context = 0;
int rule_num = 0;

printf("Choose which file to compress\n\n");
printf("1:binary.dat 2:text.dat 3:audio.dat 4:image.dat\n\n");
printf("Type in the index of file:");
cin >> file_ch;
switch (file_ch)
{
case '1':file_name = "binary.dat";
        break;
case '2':file_name = "text.dat";
        break;
case '3':file_name = "audio.dat";
        break;
case '4':file_name = "image.dat";
        break;
default: cerr << "No choose of the file" << endl; abort();
}

printf("type in context rule\n\n");
cin >> rule_num;

if ((err = fopen_s(&output_file, "binary_outcome", "wb")) != 0)
        printf("The output file was not opened\n");
else
        printf("The output file was opened\n");

qm = new QM(output_file);
qm->StartQM();
qm->reset();


ifstream infile(file_name, ios::binary);
if (!infile)
        {
                cerr << "open error!" << endl;
                abort();
        }
int N = 0;
if (file_ch == '4') {
        Buff = (int*)calloc(sizeof(int), 1);
        while (infile.peek() != EOF)
        {
```

```
                infile.read((char*)Buff, sizeof(char));
                buffvector.push_back(*Buff);
        }
        /*Zigzag process*/
        for (N = 0; N <= 255; N++)
        {
                for (i = 0; i <= N; i++)
                {
                        SECbuffvector.push_back(buffvector[i * 256 + (N - i)]);
                }
                N++;
                for (i = N; i >= 0; i--)
                {
                        SECbuffvector.push_back(buffvector[i * 256 + (N - i)]);
                }
        }

        for (N = 256; N <= 510; N++)
        {
                for (i = N - 255; i < 256; i++)
                {
                        SECbuffvector.push_back(buffvector[i * 256 + (N - i)]);
                }
                N++;
                for (i = 255; i >= N - 255; i--)
                {
                        SECbuffvector.push_back(buffvector[i * 256 + (N - i)]);
                }
        }

}
else {
        Buff = (int*)calloc(sizeof(int), 1);
        while (infile.peek() != EOF)
        {
                infile.read((char*)Buff, sizeof(char));
                SECbuffvector.push_back(*Buff);
        }

}


int j = 0;

for (j = 0; j < SECbuffvector.size();j++)
        {
                *Buff = SECbuffvector[j];
                mask = 0x80;
                for (i = 0; i < 8; i++)
                {
                        value = (*Buff & mask ? 1 : 0);
                        mask >>= 1;
```

```
                        if (file_start < (rule_num))
                        {

                                qm->encode(value, 0);
                                file_start++;
                                buffvector.push_back(value);
                                context <<= 1;
                                context += value;
                        }
                        else
                        {
                                qm->encode(value, context);
                                context <<= 1;
                                context += value;
                                switch (rule_num)
                                {
                                case 0:context &= 0x00; break;
                                case 1:context &= 0x01; break;
                                case 2:context &= 0x03; break;
                                case 3:context &= 0x07; break;
                                }

                        }


                }
                sam_weight[*Buff]++;
                sum_weight++;
        }


   qm->Flush();
        out_count=qm->Counting();
        fclose(output_file);

        printf("The size of output file is %f Bits \n\n", out_count - 1);
        system("pause");
        delete Buff;
        delete qm;
        return 0;
}

/*
s=10;
for (i = 0; i < 10; i++)  //map procedure
{
test_count++;
if (TEXT_NUMBER[i] == MPS)
{
Code_MPS();  //Recieve the MPS symbol
}
```

```
else Code_LPS(); //Recieve the LPS symbol
}
*/

// QM_CODE_SHANGLIY.cpp

#include "stdafx.h"
#include "fstream"
#include "iostream"
#include "vector"
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include "bitio.h"
#include "errhand.h"
#include <malloc.h>
#include "qmcoder.h"

#define PACIFIER_COUNT 2047
#define INF 65536
using namespace std;
errno_t err;

BIT_FILE *OpenOutputBitFile(char *name)

{
	BIT_FILE *bit_file;

	bit_file = (BIT_FILE *)calloc(1, sizeof(BIT_FILE));
	if (bit_file == NULL)
		return(bit_file);
	if ((err = fopen_s(&bit_file->file, name, "wb")) != 0)
		cout << "The Input file was not opened\n";
	else
		cout << "The Input file was opened\n";

	bit_file->rack = 0;
	bit_file->mask = 0x80;
	bit_file->pacifier_counter = 0;
	return(bit_file);
}

void OutputBit(BIT_FILE *bit_file, int bit)

{
	if (bit)
		bit_file->rack |= bit_file->mask;
	bit_file->mask >>= 1;
	if (bit_file->mask == 0) {
		if (putc(bit_file->rack, bit_file->file) != bit_file->rack);
		// fatal_error( );
		else
```

```
                        if ((bit_file->pacifier_counter++ & PACIFIER_COUNT) == 0)
                                putc('.', stdout);
                bit_file->rack = 0;
                bit_file->mask = 0x80;
        }
}

void OutputBits(BIT_FILE *bit_file, unsigned long code, int count)
{
        unsigned long mask;

        mask = 1L << (count - 1);
        while (mask != 0) {
                if (mask & code)
                        bit_file->rack |= bit_file->mask;
                bit_file->mask >>= 1;
                if (bit_file->mask == 0) {
                        if (putc(bit_file->rack, bit_file->file) != bit_file->rack);
                        //fatal_error();
                        else if ((bit_file->pacifier_counter++ & PACIFIER_COUNT) == 0)
                                putc('.', stdout);
                        bit_file->rack = 0;
                        bit_file->mask = 0x80;
                }
                mask >>= 1;
        }
}

void CloseOutputBitFile(BIT_FILE *bit_file)

{
        if (bit_file->mask != 0x80)
                if (putc(bit_file->rack, bit_file->file) != bit_file->rack);
        //fatal_error();
        fclose(bit_file->file);
        free((char *)bit_file);
}

double *t_table;
double *r_table;
double r[32] = { 0 };
double t[32] = { 0 };

double num_sum(int min, int max,double *pro)
{
        int i;
        double num_sum=0;
        for (i = min; i < max; i++)
        {
                num_sum += ((double)i)*pro[i];
        }
        return num_sum;
```

```
}

double pro_sum(int min, int max, double *pro)
{
	int i;
	double num_sum = 0;
	for (i = min; i < max; i++)
	{
		num_sum += pro[i];
	}
	return num_sum;
}

void Initial(int size,double *weight)
{
	int i;
	int j;
	int k;
	double dif;
	double sum = 0;
	double min_dif = INF;


	t_table = (double *)calloc(size, sizeof(double));
	r_table = (double *)calloc(size, sizeof(double));

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

		t_table[i] = 0;
		r_table[i] = 0;
	}

	for (j = 1; j < size; j++)
	{
		min_dif = INF;
		sum = 0;
		for (i = 0; i < 256; i++)
		{
			for (k = 0; k < weight[i]; k++)
			{
				sum ++;
				if (sum == (double)j * 65536 * 3 / size)
					t_table[j] = (double)i-1+(double)k/weight[i];
			}

		}
		t[j] = t_table[j];

	}

}
```

```
void Update(int size, double *pro)
{
	int j = 0;
	for (j = 0; j < size; j++)
	{

		if (j < size-1 )
		{
			r_table[j] = num_sum(t_table[j], t_table[j + 1], pro) / pro_sum(t_table[j],
t_table[j + 1], pro);
			r[j] = r_table[j];

		}
		else
		{
			r_table[j] = num_sum(t_table[j], 256, pro) / pro_sum(t_table[j], 256, pro);
			r[j] = r_table[j];

		}

	}

	for (j = 1; j <size; j++)
	{
		t_table[j] = (r_table[j] + r_table[j - 1]) / 2;
		t[j] = t_table[j];
	}

}

double Quanti_fun(int in_value,int size)
{
	int i = 0;
	for (i = 0; i < size-1;i++)
		if (in_value >= t_table[i] && in_value < t_table[i+1])
			return r_table[i];

	if (in_value >= t_table[i] && in_value < 256)
		return r_table[i];
}


int _tmain(int argc, _TCHAR* argv[])
{
	int i = 0;
	int j = 0;
	int k = 0;
	int index_size;

	char file_ch;
	const char* file_name[6];
	int *Buff;
```

```
int buff_pre = INF;
double sam_weight[6][256] = { 0 };
double qua_value[6][256] = { 0 };
double sum_weight[256] = { 0 };
double new_sam_weight[6][256] = { 0 };
double pro_each[6][256] = { 0 };
double pro[256]= { 0 };



unsigned long code = 0;
FILE *output_file;

unsigned char value = 0;
unsigned char mask = 0x80;
int context = 0;
int rule_num = 0;

file_name[0] = "chem.256";
file_name[1] = "house.256";
file_name[2] = "moon.256";
file_name[3] = "f16.256";
file_name[4] = "couple.256";
file_name[5] = "elaine.256";

for (i = 0; i < 3; i++)
{
	ifstream infile(file_name[i], ios::binary);
	if (!infile)
	{
		cerr << "open error!" << endl;
		abort();
	}

	Buff = (int*)calloc(sizeof(int), 1);

	while (infile.peek() != EOF)
	{
		infile.read((char*)Buff, sizeof(char));

		sam_weight[i][*Buff]++;
		sum_weight[*Buff]++;
		pro[*Buff]++;

	}
	infile.close();
}

for (i = 3; i < 6; i++)
{
	ifstream testfile(file_name[i], ios::binary);
```

```
                if (!testfile)
                {
                        cerr << "open error!" << endl;
                        abort();
                }

                Buff = (int*)calloc(sizeof(int), 1);

                while (testfile.peek() != EOF)
                {
                        testfile.read((char*)Buff, sizeof(char));

                        sam_weight[i][*Buff]++;
                }
                testfile.close();
        }

        for (i = 0; i < 256; i++)
        {
                pro[i]= pro[i]/(double)(3*65536);
        }

        double test_sum;
        test_sum = pro_sum(0,256,pro);

        index_size = 32;
        Initial(index_size, sum_weight);

        double e_dif = 65536;
        double PSNR = 0;
        double MSE[2] = { INF };
        double num_count = 0;
   k = 1;
        while (e_dif>0.001)
        {
                Update(index_size, pro);
                num_count = 0;
                MSE[1] = 0;
                for (i = 0; i < 256; i++)
                        {
                                MSE[1]+=(Quanti_fun(i, index_size)-i)*(Quanti_fun(i, index_size) -
i)*sum_weight[i]/65536;
                                num_count += sum_weight[i];
                        }
                MSE[1] /= 3;
                e_dif = (MSE[0] - MSE[1]) / MSE[1];

                PSNR = 10 * log10(255*255/ MSE[1]);
                cout << k << ":" << PSNR << endl;
                cout << "\n" << endl;
                k++;
                MSE[0] = MSE[1];
```

```
	}


	cout << "the number of iteration=" << k-1 << endl;

	int q_value=0;
	double PSNR_e[6] = { 0 };
	double MSE_e[6] = { 0 };
	double entro[6] = { 0 };
	for (i = 0; i < 6; i++)
	{
		for (j = 0; j < 256; j++)
		{
			q_value = round(Quanti_fun(j, index_size));
			new_sam_weight[i][q_value] += sam_weight[i][j];
			MSE_e[i] += (q_value - j)*(q_value - j)*sam_weight[i][j];
			/*Calculate the Entropy for the file*/

		}

		for (j = 0; j < 256; j++)
		{
			if (new_sam_weight[i][j])
				entro[i] = entro[i] + (new_sam_weight[i][j] /
(double)65536)*(log(new_sam_weight[i][j] / ((double)65536)) / log(2.0));

		}
		entro[i] = -entro[i];
		MSE_e[i] /= 65536;
		PSNR_e[i] = 10 * log10(255 * 255 / MSE_e[i]);
	}







	system("pause");
	i = 0;
	delete Buff;
	delete r_table;
	delete t_table;
	return 0;
}

// QM_CODE_SHANGLIY.cpp


#include "fstream"
```

```
#include "iostream"
#include "vector"
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include "bitio.h"
#include "errhand.h"
#include <math.h>
#include <malloc.h>
#define PACIFIER_COUNT 2047
#define INF 65536
using namespace std;


BIT_FILE *OpenOutputBitFile( char *name )
{
   BIT_FILE *bit_file;

   bit_file = (BIT_FILE *) calloc( 1, sizeof( BIT_FILE ) );
   if ( bit_file == NULL )
      return( bit_file );
   bit_file->file = fopen( name, "wb" );
   bit_file->rack = 0;
   bit_file->mask = 0x80;
   bit_file->pacifier_counter = 0;
   return( bit_file );
}

void OutputBit(BIT_FILE *bit_file, int bit)

{
        if (bit)
                bit_file->rack |= bit_file->mask;
        bit_file->mask >>= 1;
        if (bit_file->mask == 0) {
                if (putc(bit_file->rack, bit_file->file) != bit_file->rack);
                // fatal_error( );
                else
                        if ((bit_file->pacifier_counter++ & PACIFIER_COUNT) == 0)
                                putc('.', stdout);
                bit_file->rack = 0;
                bit_file->mask = 0x80;
        }
}

void OutputBits(BIT_FILE *bit_file, unsigned long code, int count)
{
        unsigned long mask;

        mask = 1L << (count - 1);
        while (mask != 0) {
                if (mask & code)
```

```
                bit_file->rack |= bit_file->mask;
            bit_file->mask >>= 1;
            if (bit_file->mask == 0) {
                if (putc(bit_file->rack, bit_file->file) != bit_file->rack);
                //fatal_error();
                else if ((bit_file->pacifier_counter++ & PACIFIER_COUNT) == 0)
                    putc('.', stdout);
                bit_file->rack = 0;
                bit_file->mask = 0x80;
            }
            mask >>= 1;
        }
}

void CloseOutputBitFile(BIT_FILE *bit_file)

{
        if (bit_file->mask != 0x80)
            if (putc(bit_file->rack, bit_file->file) != bit_file->rack);
        //fatal_error();
        fclose(bit_file->file);
        free((char *)bit_file);
}


int main()
{
        int i = 0;
        int j = 0;
        int k = 0;
        int f = 0;
        int m = 0;

        int codebook_size;
        int dimension;

   int *Buff;
   Buff = (int*)calloc(sizeof(int), 1);
   int image_data[256][256]={0};
   double outcount=0;

   const char* file_name[6];
        file_name[0] = "chem.256";
        file_name[1] = "house.256";
        file_name[2] = "moon.256";
        file_name[3] = "f16.256";
        file_name[4] = "couple.256";
        file_name[5] = "elaine.256";

        cout<<"Please type in the dimension of the vector (1 or 2 or 3): ";
        cin >> dimension;
```

```
        BIT_FILE *output_file;

        int vector_SIZE=pow(2,dimension)*pow(2,dimension);
        char *tem_vector;
    tem_vector=(char*)calloc(sizeof(char), vector_SIZE);
    output_file=OpenOutputBitFile("trainset_64");

    for (f=0;f<3;f++)
       {

            ifstream infile(file_name[f], ios::binary);
            if (!infile)
               {
                  cerr << "unable to open input files!" << endl;
                  abort();
               }

                  for (i = 0; i < 256; i++)
                     {
                        for (j = 0; j < 256; j++)
                        {
                           if (infile.peek() != EOF)
                           {
                              infile.read((char*)Buff, sizeof(char));
                              image_data[i][j]=*Buff;
                           }
                        }
                     }

                  i=0;
                  j=0;

          for(m=0;m<256/pow(2,dimension);m++)
             {
                  for(k=0;k<256/pow(2,dimension);k++)
                  {
                   for (i=0;i<pow(2,dimension);i++)
                     {
                        for(j=0;j<pow(2,dimension);j++)
                        {
                           OutputBits(output_file,image_data[(int)pow(2,dimension)*m+i]
[(int)pow(2,dimension)*k+j],8);
                           outcount++;
                        }
                     }
                  }
             }

            infile.close();
       }

    CloseOutputBitFile(output_file);
```

```
   for (f=3;f<6;f++)
      {

            if (f==3) output_file=OpenOutputBitFile("testset_f16_4");
            if (f==4) output_file=OpenOutputBitFile("testset_couple_4");
            if (f==5) output_file=OpenOutputBitFile("testset_elaine_4");
            ifstream infile(file_name[f], ios::binary);
            if (!infile)
               {
                  cerr << "unable to open input files!" << endl;
                  abort();
               }

                  for (i = 0; i < 256; i++)
                     {
                        for (j = 0; j < 256; j++)
                        {
                           if (infile.peek() != EOF)
                           {
                              infile.read((char*)Buff, sizeof(char));
                              image_data[i][j]=*Buff;
                           }
                        }
                     }

         for(m=0;m<256/pow(2,dimension);m++)
            {
                  for(k=0;k<256/pow(2,dimension);k++)
                  {
                   for (i=0;i<pow(2,dimension);i++)
                     {
                        for(j=0;j<pow(2,dimension);j++)
                        {
                           OutputBits(output_file,image_data[(int)pow(2,dimension)*m+i]
[(int)pow(2,dimension)*k+j],8);
                           outcount++;
                        }
                     }
                  }
            }

            infile.close();
            CloseOutputBitFile(output_file);
      }


         delete Buff;
         return 0;
}

// The programe reads the image data from an image file "~.raw"
```

```
// Last updated on 02/20/2010 by Steve Cho

#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <fstream>


using namespace std;

// Here we assume the image is of size 256*256 and is of raw format
// You will need to make corresponding changes to accommodate images of different sizes and types

#define Size 16
#define N 8
#define PACIFIER_COUNT 2047

double DCT[N][N];
double Converdata[Size][Size]; //store the datas ready for DCT transform
int Qe_10[N][N];
int Qe_90[N][N];

int v[10]={0};
int v_new[10]={0};
double a[4][3]={0};
int QP=0;
double c1=2;
double c2=5;
double c3=8;

// Here the QE matrix with factor 50
double Qe[N][N]={
{ 16,   11,     10,     16,  24,   40,   51,     61},
{ 12,   12,     14,     19,  26,   58,   60,     55},
{ 14,   13,     16,     24,  40,   57,   69,     56},
{ 14,   17,     22,     29,  51,   87,   80,     62},
{ 18,   22,     37,     56,  68,   109,  103,    77},
{ 24,   35,     55,     64,  81,   104,  113,    92},
{ 49,   64,     78,     87,  103,  121,  120,    101},
{ 72,   92,     95,     98,  112,  100,  103,    99}
};

int Build_QE(int n)
{
   int i=0;
   int j=0;
   if (n==10)
   {
      for (i=0;i<N;i++)
      {
         for (j=0;j<N;j++)
```

```
            {
                Qe_10[i][j]=Qe[i][j]*50/n;
            }
        }
    }

    if (n==90)
    {
        for (i=0;i<N;i++)
        {
            for (j=0;j<N;j++)
            {
                Qe_90[i][j]=Qe[i][j]*(100-n)/50;
            }
        }
    }
    return 1;
}

int DCT_Tran()
{
    int i=0;
    int j=0;
    int m=0;
    int n=0;
    double Ci=0;
    double Cj=0;


    for (i=0;i<N;i++)
    {
        for (j=0;j<N;j++)
        {
            DCT[i][j]=0;
        }
    }

    for(i=0;i<N;i++)
    {
        for (j=0;j<N;j++)
        {
             for (m=0;m<N;m++)
                {
                    for (n=0;n<N;n++)
                        {
                            if (i==0) Ci=1/sqrt(2);
                            else Ci=1;
                            if (j==0) Cj=1/sqrt(2);
                            else Cj=1;
                            DCT[i][j]+=(Ci *Cj * (Converdata[m][n]) *cos( (2*m+1)*i*M_PI /(2*N) )
*cos((2*n+1)*j*M_PI /(2*N)) )/sqrt(2*N);
                        }
```

```
            }
        }
    }
    return 1;

}

int MAX_FUN(int n)
{
 int temp_max=0;
 int i=0;
 for (i=0;i<n;i++)
 {
  if(v[i]>temp_max)
  temp_max=v[i];

 }
 return temp_max;
 }

 int MIN_FUN(int n)
{
 int temp_min=256;
 int i=0;
 for (i=0;i<n;i++)
 {
  if(v[i]<temp_min)
  temp_min=v[i];

 }
 return temp_min;
}


int main(int argc, char *argv[])

{
        // file pointer
        FILE *file;
        //ByteesPerPixels
        int BytesPerPixel;
        //Data after shift
        char shift_data[Size][Size];
        //Calculate the PSNNR
        double PSNR[10];
        double MSE[10]={0};
        int i=0;
        int j=0;
        int m=0;
        int k=0;

    //For gray image =1 For colore =3
```

```
BytesPerPixel=1;

	// image data array
	unsigned char Imagedata[Size][Size][BytesPerPixel];

	// processed image data input
	unsigned char Imagedata_com[Size][Size][3];

	int Ima_origin[Size][Size];
	int Ima_outdata[Size][Size];

	char input_name[30];
	char output_name[30];
	int factor=50;

	sprintf(input_name,"lena.raw");


	// read orginal image  into image data matrix
	if (!(file=fopen("lena.raw","rb")))
	{
		cout << "Cannot open file: " << "clock.raw" <<endl;
		exit(1);
	}
	fread(Imagedata, sizeof(unsigned char), Size*Size*1, file);
	fclose(file);


// read another image_data  into image data matrix
	if (!(file=fopen(input_name,"rb")))
	{
		cout << "Cannot open file: " << argv[1] <<endl;
		exit(1);
	}
	fread(Imagedata_com, sizeof(unsigned char), Size*Size*BytesPerPixel, file);
	fclose(file);

/*Build different factor QUantinization*/
	Build_QE(10);
	Build_QE(90);


	ofstream outfile("Qe_10.txt", ios::out); //define the out stream
	if (!outfile)
	{
		cerr << "open error!" << endl;
		exit(1);
	}
	for (i = 0; i<N; i++)
	{
		outfile << "\n";
		for (j = 0; j<N; j++)
```

```
                {
                        outfile << Qe_10[i][j] << " ";
                }
        }
        outfile.close();

ofstream outfile1("Qe_90.txt", ios::out); //define the out stream
        if (!outfile1)
        {
                cerr << "open error!" << endl;
                exit(1);
        }
        for (i = 0; i<N; i++)
        {
                outfile1 << "\n";
                for (j = 0; j<N; j++)
                {
                        outfile1 << Qe_90[i][j] << " ";
                }
        }
        outfile1.close();

        //Do the shift
for (i=0;i<Size;i++)
        {
    for (j=0;j<Size;j++)
    {
    shift_data[i][j]=Imagedata[i][j][BytesPerPixel-1]-128;
    }
        }

        //For each block.do DCT
        int index=0;
        for(m=0;m<(Size/N);m++)
    {
      for(k=0;k<(Size/N);k++)
        {
        //iN ONE BLOCK
        for (i=0;i<N;i++)
            {
            for(j=0;j<N;j++)
                {
                Converdata[i][j]= shift_data[N*m+i][N*k+j];
                }
            }
        //Do DCT
        if (!DCT_Tran()) cout<<"DCT TAKE WRONGS";
        if (factor==50)
        {
            for (i=0;i<N;i++)
            {
                for (j=0;j<N;j++)
```

```
        {
            DCT[i][j]=round(DCT[i][j]/Qe[i][j]);
        }
    }
    sprintf (output_name,"DCT_%d_%d",factor,index);
    index++;

}
else if (factor==10)
{
    for (i=0;i<N;i++)
    {
        for (j=0;j<N;j++)
        {
            DCT[i][j]=round(DCT[i][j]/Qe_10[i][j]);
        }
    }
    sprintf (output_name,"DCT_%d_%d",factor,index);
    index++;

}
else if (factor==90)
{
    for (i=0;i<N;i++)
    {
        for (j=0;j<N;j++)
        {
            DCT[i][j]=round(DCT[i][j]/Qe_90[i][j]);
        }
    }
    sprintf (output_name,"DCT_%d_%d",factor,index);
    index++;

}

 for (i=0;i<N;i++)
    {
    for(j=0;j<N;j++)
        {
        Ima_outdata[i][j]= DCT[i][j];
        }
    }

    // write image data to "~.raw"

    if (!(file=fopen(output_name,"wb")))
        {
            cout << "Cannot open file: " << Ima_outdata << endl;
            exit(1);
        }

        fwrite(Ima_outdata, sizeof(unsigned char), N*N, file);
```

```
            fclose(file);

            ofstream outfile(output_name, ios::out); //define the out stream
            if (!outfile)
            {
               cerr << "open error!" << endl;
               exit(1);
            }
            for (i = 0; i<N; i++)
            {
               outfile << "\n";
               for (j = 0; j<N; j++)
               {
                  outfile << Ima_outdata[i][j] << " ";
               }
            }
            outfile.close();

         }

      }


         return 0;
}

// The programe reads the image data from an image file "~.raw"
// Last updated on 02/20/2010 by Steve Cho

#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <fstream>

using namespace std;

// Here we assume the image is of size 256*256 and is of raw format
// You will need to make corresponding changes to accommodate images of different sizes and types

#define Size 256
#define N 8
#define PACIFIER_COUNT 2047

double DCT[N][N];
double Converdata[Size][Size]; //store the datas ready for DCT transform
double Qe_10[N][N];
double Qe_90[N][N];

double Qe[N][N]={
{ 16,   11,     10,     16,  24,   40,   51,     61},
{ 12,   12,     14,     19,  26,   58,   60,     55},
```

```
{ 14,   13,     16,     24,  40,   57,   69,     56},
{ 14,   17,     22,     29,  51,   87,   80,     62},
{ 18,   22,     37,     56,  68,   109,  103,    77},
{ 24,   35,     55,     64,  81,   104,  113,    92},
{ 49,   64,     78,     87,  103,  121,  120,    101},
{ 72,   92,     95,     98,  112,  100,  103,    99}
};


int main(int argc, char *argv[])

{
        // file pointer
        FILE *file;

        int shift_data[Size][Size];

        double PSNR[2];
        double MSE[2]={0};

   int BytesPerPixel;
   BytesPerPixel=1;
        // image data array
        unsigned char origin_data[Size][Size][BytesPerPixel];
        unsigned char compare_data[Size][Size][3];
        int Ima_origin[Size][Size];
        int Ima_outdata[Size][Size];

   char input_name[30];
        char output_name[30];
        int factor=100;
while(1){

        cout<<"Type int the factor"<<endl;
        cin>>factor;
        if (factor>100) return 0;

        sprintf(input_name,"clock_qua%d.raw",factor);

        // read image "ride.raw" into image data matrix
        if (!(file=fopen("clock.raw","rb")))
        {
                cout << "Cannot open file: " << argv[1] <<endl;
                exit(1);
        }
        fread(origin_data, sizeof(unsigned char), Size*Size*BytesPerPixel, file);
        fclose(file);

        if (!(file=fopen(input_name,"rb")))
        {
                cout << "Cannot open file: " << argv[1] <<endl;
                exit(1);
```

```
        }
        fread(compare_data, sizeof(unsigned char), Size*Size*3, file);
        fclose(file);


        // do some image processing task...
        int i=0;
        int j=0;
        int m=0;
        int k=0;


  for (i=0;i<Size;i++)
          {
      for (j=0;j<Size;j++)
      {
      if (BytesPerPixel==1) MSE[0]+=(double)(origin_data[i][j][0]-compare_data[i][j][0])*(double)
(origin_data[i][j][0]-compare_data[i][j][0]);
      else MSE[0]+=(double)(origin_data[i][j][0]-compare_data[i][j][0])*(double)(origin_data[i][j]
[0]-compare_data[i][j][0])
                +(double)(origin_data[i][j][1]-compare_data[i][j][1])*(double)(origin_data[i][j][1]-
compare_data[i][j][1])
                +(double)(origin_data[i][j][2]-compare_data[i][j][2])*(double)(origin_data[i][j][2]-
compare_data[i][j][2]);
      }
         }

   MSE[0]=MSE[0]/(Size*Size*BytesPerPixel);
   PSNR[0]=10*log10(255*255/MSE[0]);
   MSE[0]=0;


   cout<< PSNR[0]<<endl;
   }
}


// The programe reads the image data from an image file "~.raw"
// Last updated on 02/20/2010 by Steve Cho

#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <fstream>
#include "bitio.h"
#include "errhand.h"
#include "jpeglib.h"
#include <setjmp.h>

using namespace std;
```

```
// Here we assume the image is of size 256*256 and is of raw format
// You will need to make corresponding changes to accommodate images of different sizes and types

#define Size 256
#define N 8
#define PACIFIER_COUNT 2047

double DCT[N][N];
unsigned char Converdata[Size][Size][3];
unsigned char Converdata_2[Size][Size][3];//store the datas ready for DCT transform
double Qe_10[N][N];
double Qe_90[N][N];

int v[10]={0};
int v_new[10]={0};
double a[4][3]={0};
int QP=0;
double c1=2;
double c2=5;
double c3=8;

double Qe[N][N]={
{ 16,   11,     10,     16,  24,   40,   51,     61},
{ 12,   12,     14,     19,  26,   58,   60,     55},
{ 14,   13,     16,     24,  40,   57,   69,     56},
{ 14,   17,     22,     29,  51,   87,   80,     62},
{ 18,   22,     37,     56,  68,   109,  103,    77},
{ 24,   35,     55,     64,  81,   104,  113,    92},
{ 49,   64,     78,     87,  103,  121,  120,    101},
{ 72,   92,     95,     98,  112,  100,  103,    99}
};

BIT_FILE *OpenOutputBitFile( char *name )

{
   BIT_FILE *bit_file;

   bit_file = (BIT_FILE *) calloc( 1, sizeof( BIT_FILE ) );
   if ( bit_file == NULL )
      return( bit_file );
   bit_file->file = fopen( name, "wb" );
   bit_file->rack = 0;
   bit_file->mask = 0x80;
   bit_file->pacifier_counter = 0;
   return( bit_file );
}



int main(int argc, char *argv[])

{
```

```
        // file pointer
        FILE *file;
        int BytesPerPixel;
        int shift_data[Size][Size];
        double PSNR[6];
        double MSE[6]={0};
int width=Size;
int height=Size;

int bitCountPerPix=3;
int in_BytesPerPixel=1;
BytesPerPixel=3;

        // do some image processing task...
        int i=0;
        int j=0;
        int m=0;
        int k=0;
        int vec=1;
        int hem=1;
        char name[20];
        int debug=0;
        int t=0;
        int l=0;

        // image data array
        unsigned char Imagedata[Size][Size][3];
        unsigned char Imagedata_com[Size][Size][3];
        int Ima_origin[Size][Size][3];
        int Ima_outdata[Size][Size][3];


        while(1)
        {
        cout<<"Type in the which file type"<<endl;
        cin>>t;
        cout<<"Type in the file index"<<endl;
        cin>>l;
        debug=0;

        char input_name[30];
        char output_name[30];


        // read image "ride.raw" into image data matrix
        if(t==1)
        {
            sprintf(input_name,"RAWDATA/clock_pro_%d.raw",l);
        }
        else if(t==2)
        {
           sprintf(input_name,"RAWDATA//pepper_pro_%d.raw",l);
```

```
}
else return 0;

if (!(file=fopen(input_name,"rb")))
   {
      cout << "Cannot open file: " << "clock.raw" <<endl;
      exit(1);
   }
   fread(Imagedata_com, sizeof(unsigned char), Size*Size*BytesPerPixel, file);
   fclose(file);

k=0;
for (vec=-3;vec<=4;vec++)
{
 for (hem=-3;hem<=4;hem++)
   {
      for (i=0;i<Size;i++)
         {
            for(j=0;j<Size;j++)
               {
                  if (j-hem>=Size)
                  {
                  Converdata[i][j][0]=Imagedata_com[i][j][0];
                  Converdata[i][j][1]=Imagedata_com[i][j][1];
                  Converdata[i][j][2]=Imagedata_com[i][j][2];
                  }

                  else if (j-hem<0)
                  {
                  Converdata[i][j][0]=Imagedata_com[i][j][0];
                  Converdata[i][j][1]=Imagedata_com[i][j][2];
                  Converdata[i][j][2]=Imagedata_com[i][j][2];

                  }

                  else
                  {
                  Converdata[i][j][0]=Imagedata_com[i][j-hem][0];
                  Converdata[i][j][1]=Imagedata_com[i][j-hem][1];
                  Converdata[i][j][2]=Imagedata_com[i][j-hem][2];

                  }
               }
         }

      for (i=0;i<Size;i++)
         {
            for(j=0;j<Size;j++)
               {
                  if ((i+vec)>=Size)
                  {
```

```
                    Converdata_2[i][j][0]=Converdata[i][j][0];
                    Converdata_2[i][j][1]=Converdata[i][j][1];
                    Converdata_2[i][j][2]=Converdata[i][j][2];
                    }

                    else if ((i+vec)<0)
                    {
                    Converdata_2[i][j][0]=Converdata[i][j][0];
                    Converdata_2[i][j][1]=Converdata[i][j][1];
                    Converdata_2[i][j][2]=Converdata[i][j][2];
                    }

                    else
                    {
                    Converdata_2[i][j][0]=Converdata[i+vec][j][0];
                    Converdata_2[i][j][1]=Converdata[i+vec][j][1];
                    Converdata_2[i][j][2]=Converdata[i+vec][j][2];
                    }
                }
            }
            if (t==1) sprintf(name,"RAW_64/clock%d_%d.raw",l,k);
            else sprintf(name,"RAW_64/pepper%d_%d.raw",l,k);
            k++;

            if (!(file=fopen(name,"wb")))
                {
                cout << "Cannot open file: " << name << endl;
                exit(1);
                }
            fwrite(Converdata_2, sizeof(unsigned char), Size*Size*3, file);
            fclose(file);
        }
    }



    unsigned char raw_data_in[Size][Size][3];
    unsigned char tem[Size][Size][3];
    int sum_data_in[Size][Size][3]={0};
    int pro_prodata[Size][Size][3];
    char raw_in[30];

    debug=0;
      if (debug==0)
{
        for (m=1;m<=5;m++)
            {
                k=0;
                for (vec=3;vec>=-4;vec--)
                {
                for (hem=3;hem>=-4;hem--)
                    {
```

```
if (t==1) sprintf(raw_in, "NEWRAW/dclock%d_%d.raw", m,k);
else sprintf(raw_in, "NEWRAW/dpepper%d_%d.raw", m,k);
k++;
if (!(file=fopen(raw_in,"rb")))
   printf("The Input file was not opened\n");
else
   printf("The Input file was opened\n");
fread(raw_data_in, sizeof(unsigned char), Size*Size * BytesPerPixel, file);
fclose (file);

for (i=0;i<Size;i++)
{
   for(j=0;j<Size;j++)
   {
      tem[i][j][0]=raw_data_in[Size-1-i][j][0];
      tem[i][j][1]=raw_data_in[Size-1-i][j][1];
      tem[i][j][2]=raw_data_in[Size-1-i][j][2];

   }

}

 for (i=0;i<Size;i++)
{
   for(j=0;j<Size;j++)
   {
      raw_data_in[i][j][0]=tem[i][j][0];
      raw_data_in[i][j][1]=tem[i][j][1];
      raw_data_in[i][j][2]=tem[i][j][2];

   }

}


for (i=0;i<Size;i++)
   {
         for(j=0;j<Size;j++)
         {
            if (j-hem>=Size)
            {
            Converdata[i][j][0]=raw_data_in[i][j][0];
            Converdata[i][j][1]=raw_data_in[i][j][1];
            Converdata[i][j][2]=raw_data_in[i][j][2];
            }

            else if (j-hem<0)
            {
            Converdata[i][j][0]=raw_data_in[i][j][0];
            Converdata[i][j][1]=raw_data_in[i][j][2];
            Converdata[i][j][2]=raw_data_in[i][j][2];
```

```
                }

                else
                {
                Converdata[i][j][0]=raw_data_in[i][j-hem][0];
                Converdata[i][j][1]=raw_data_in[i][j-hem][1];
                Converdata[i][j][2]=raw_data_in[i][j-hem][2];

                }
            }
        }

    for (i=0;i<Size;i++)
        {
            for(j=0;j<Size;j++)
                {
                if ((i+vec)>=Size)
                {
                Converdata_2[i][j][0]=Converdata[i][j][0];
                Converdata_2[i][j][1]=Converdata[i][j][1];
                Converdata_2[i][j][2]=Converdata[i][j][2];
                }

                else if ((i+vec)<0)
                {
                Converdata_2[i][j][0]=Converdata[i][j][0];
                Converdata_2[i][j][1]=Converdata[i][j][1];
                Converdata_2[i][j][2]=Converdata[i][j][2];
                }

                else
                {
                Converdata_2[i][j][0]=Converdata[i+vec][j][0];
                Converdata_2[i][j][1]=Converdata[i+vec][j][1];
                Converdata_2[i][j][2]=Converdata[i+vec][j][2];
                }
                sum_data_in[i][j][0]+=(int)Converdata_2[i][j][0];
                sum_data_in[i][j][1]+=(int)Converdata_2[i][j][1];
                sum_data_in[i][j][2]+=(int)Converdata_2[i][j][2];

                }
        }

    }

}

k=0;
for (i=0;i<Size;i++)
{
for (j=0;j<Size;j++)
    {
```

```
                    (sum_data_in[i][j][0]/=64);
                    (sum_data_in[i][j][1]/=64);
                    (sum_data_in[i][j][2]/=64);
                    Converdata_2[i][j][0]=sum_data_in[i][j][0];
                    Converdata_2[i][j][1]=sum_data_in[i][j][1];
                    Converdata_2[i][j][2]=sum_data_in[i][j][2];
                    sum_data_in[i][j][0]=0;
                    sum_data_in[i][j][1]=0;
                    sum_data_in[i][j][2]=0;

                    }
                }

                if (t==1)sprintf(name,"NEWPRORAW/clock%d.raw",m);
                else sprintf(name,"NEWPRORAW/pepper%d.raw",m);

                if (!(file=fopen(name,"wb")))
                    {
                       cout << "Cannot open file: " << name << endl;
                       exit(1);
                    }
                fwrite(Converdata_2, sizeof(unsigned char), Size*Size*3, file);
                fclose(file);

            }

        }
          }


          return 0;
}
```

```
// The programe reads the image data from an image file "~.raw"
// Last updated on 02/20/2010 by Steve Cho

#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <fstream>



using namespace std;

// Here we assume the image is of size 256*256 and is of raw format
// You will need to make corresponding changes to accommodate images of different sizes and types
```

```
typedef unsigned char BYTE;
typedef unsigned short WORD;
typedef unsigned int DWORD;
typedef int LONG;

#define Size 256
#define N 8
#define PACIFIER_COUNT 2047


typedef struct  tagBITMAPFILEHEADER {
        //WORD bfType;//
        DWORD bfSize;//
        WORD bfReserved1;//
        WORD bfReserved2;//
        DWORD bfOffBits;//
}BITMAPFILEHEADER;

typedef struct tagBITMAPINFOHEADER {
        DWORD biSize;//
        LONG biWidth;//
        LONG biHeight;//
        WORD biPlanes;//1
        WORD biBitCount;//
        DWORD  biCompression; //
        DWORD  biSizeImage; //
        LONG  biXPelsPerMeter; //
        LONG  biYPelsPerMeter; //
        DWORD  biClrUsed; //
        DWORD  biClrImportant; //
}BITMAPINFOHEADER; //

typedef struct tagRGBQUAD {
        BYTE rgbBlue;
        BYTE rgbGreen;
        BYTE rgbRed;
        BYTE rgbReserved; //
}RGBQUAD;//

                //
typedef struct tagIMAGEDATA
{
        BYTE red;
        BYTE green;
        BYTE blue;
}IMAGEDATA;



BITMAPFILEHEADER strHead;
RGBQUAD strPla[256];//256
```

```
BITMAPINFOHEADER strInfo;
IMAGEDATA imagedata[256][256];//
double DCT[N][N];
unsigned char Converdata[Size][Size][3]; //store the datas ready for DCT transform
unsigned char compare_data[Size][Size][3];
double Qe_10[N][N];
double Qe_90[N][N];

int v[10][3]={0};
int v_new[10][3]={0};
double a[4][3]={0};
int QP=0;
double c1=2;
double c2=5;
double c3=8;

double Qe[N][N]={
{ 16,   11,     10,     16,  24,   40,   51,     61},
{ 12,   12,     14,     19,  26,   58,   60,     55},
{ 14,   13,     16,     24,  40,   57,   69,     56},
{ 14,   17,     22,     29,  51,   87,   80,     62},
{ 18,   22,     37,     56,  68,   109,  103,    77},
{ 24,   35,     55,     64,  81,   104,  113,    92},
{ 49,   64,     78,     87,  103,  121,  120,    101},
{ 72,   92,     95,     98,  112,  100,  103,    99}
};


int MAX_FUN(int n,int p)
{
 int temp_max=0;
 int i=0;
 for (i=0;i<n;i++)
 {
   if(v[i][p]>temp_max)
   temp_max=v[i][p];

 }
 return temp_max;
 }

 int MIN_FUN(int n,int p)
{
 int temp_min=256;
 int i=0;
 for (i=0;i<n;i++)
 {
   if(v[i][p]<temp_min)
   temp_min=v[i][p];

 }
 return temp_min;
```

```
}

void Blur_9(int i,int j,int bitper)
{

int templates[9]={1,2,1,2,4,2,1,2,1};
int sum[3]= {0};
int k=0;
int a,b;

        for ( int m=i-1; m<i+2; m++)
        {
           for (int n=j-1; n<j+2; n++)
           {
           a=m;
           b=n;
              if(a<0) a=0;
              if(a>=Size) a=Size-1;
              if(b<0) b=0;
              if(b>=Size) b=Size-1;
              sum[0]+= Converdata[a][b][0] *templates[k] ;
               sum[1]+= Converdata[a][b][1] *templates[k] ;
                sum[2]+= Converdata[a][b][2] *templates[k] ;
                k++;
           }
        }
        sum[0] /= 16;
        sum[1] /= 16;
        sum[2] /= 16;
        if (sum[0]> 255)
           sum[0] = 255;
        if (sum[1] > 255)
           sum[1] = 255;
        if (sum[2] > 255)
           sum[2] = 255;
        Converdata[i][j][0] = sum[0];
        Converdata[i][j][1] = sum[1];
        Converdata[i][j][2] = sum[2];
        k=0;
}



void Smooth_region(int plane)
{
 int p=0;
 int b[9]={1,1,2,2,4,2,2,1,1};
 int m=0;
 int n=0;
 int sum=0;
```

```
 v_new[0][plane]=v[0][plane];
 v_new[9][plane]=v[9][plane];
 for(n=1;n<=8;n++)
 {
   for(m=-4;m<=4;m++)
   {
      if(m+n<1){
        if(abs(v[1][plane]-v[0][plane])<QP) p=v[0][plane];
          else p=v[1][plane];
      }
      if(m+n>8){
        if(abs(v[9][plane]-v[8][plane])<QP) p=v[9][plane];
          else p=v[8][plane];
      }
      if((m+n)<=8&&(m+n)>=1) p=v[m+n][plane];

      sum+=b[m+4]*p;
   }
   sum=sum/16;
   v_new[n][plane]=sum;
   sum=0;
 }

}

void Default_region(int plane)
{
   int d=0;
   int a_31=0;

   if(a[3][1]>0)
   {
    if( (abs(a[3][0])<abs(a[3][1])) && (abs(a[3][0])<abs(a[3][2])) )
       a_31=abs(a[3][0]);
    if( (abs(a[3][1])<abs(a[3][2])) && (abs(a[3][1])<abs(a[3][0])) )
       a_31=abs(a[3][1]);
    if( (abs(a[3][2])<abs(a[3][0])) && (abs(a[3][2])<abs(a[3][1])) )
       a_31=abs(a[3][2]);
   }
   else if (a[3][1]<0)
   {
    if( (abs(a[3][0])<abs(a[3][1])) && (abs(a[3][0])<abs(a[3][2])) )
       a_31=-abs(a[3][0]);
    if( (abs(a[3][1])<abs(a[3][2])) && (abs(a[3][1])<abs(a[3][0])) )
       a_31=-abs(a[3][1]);
    if( (abs(a[3][2])<abs(a[3][0])) && (abs(a[3][2])<abs(a[3][1])) )
       a_31=-abs(a[3][2]);
   }
   else a_31=0;

   d=c2*(a_31-a[3][1])/c3;
   if ((v[4][plane]-v[5][plane])>0)
```

```
    {
       if (d<0) d=0;
       else if (d>(v[4][plane]-v[5][plane])/2) d=(v[4][plane]-v[5][plane])/2;
    }
    else if ((v[4][plane]-v[5][plane])<0)
    {
       if (d>0) d=0;
       else if (d<(v[4][plane]-v[5][plane])/2) d=(v[4][plane]-v[5][plane])/2;
    }
    else d=0;
    v_new[4][plane]=v[4][plane]-d;
    v_new[5][plane]=v[5][plane]+d;
}


int main(int argc, char *argv[])

{
// file pointer
        FILE *file;
        FILE *fpi;

        int shift_data[Size][Size];

        double PSNR[6];
        double MSE[6]={0};

        // do some image processing task...
        int i=0;
        int j=0;
        int m=0;
        int k=0;
        int vec=1;
        int hem=1;
        int method=0; //0--mymethod;
                //1--deblocking_filter;
                //2--reapplying
    int index=0;

    int BytesPerPixel=1;
    cout<<"Type int the BytesPerPixel"<<endl;
        cin>>BytesPerPixel;


        unsigned char origin_data[Size][Size][BytesPerPixel];
        unsigned char colore_data[Size][Size][BytesPerPixel];
        unsigned char test_data[Size][Size][BytesPerPixel];

        int Ima_origin[Size][Size];
        int Ima_outdata[Size][Size];

    char input_name[30];
```

```
   char output_name[30];
   char NEWinput_name[30];
   int factor=100;

   // read image "ride.raw" into image data matrix
   if(BytesPerPixel==1)
   {
if (!(file=fopen("ORIGIN/clock.raw","rb")))
{
   cout << "Cannot open file: " << "clock.raw" <<endl;
   exit(1);
}
fread(origin_data, sizeof(unsigned char), Size*Size*BytesPerPixel, file);
fclose(file);


   if (!(fpi=fopen("ORIGIN/clock.bmp","rb")))
      printf("The bmg Input file was not opened\n");
   else
      printf("The bmg Input file was opened\n");

   if (fpi != NULL) {
      //file type
      WORD bfType;
      fread(&bfType, 1, sizeof(WORD), fpi);
      fread(&strHead, 1, sizeof(tagBITMAPFILEHEADER), fpi);
      fread(&strInfo, 1, sizeof(tagBITMAPINFOHEADER), fpi);
      //
      for (int nCounti = 0; nCounti<strInfo.biClrUsed; nCounti++) {
         //remove rgbReserved
         fread((char *)&strPla[nCounti].rgbBlue, 1, sizeof(BYTE), fpi);
         fread((char *)&strPla[nCounti].rgbGreen, 1, sizeof(BYTE), fpi);
         fread((char *)&strPla[nCounti].rgbRed, 1, sizeof(BYTE), fpi);
         fread((char *)&strPla[nCounti].rgbReserved, 1, sizeof(BYTE), fpi);
      }
      fread(test_data, sizeof(unsigned char), Size*Size*BytesPerPixel, fpi);
      fclose(fpi);

   }


   }
   else {
if (!(file=fopen("ORIGIN/pepper.raw","rb")))
{
   cout << "Cannot open file: " << "pepper.raw" <<endl;
   exit(1);
}
fread(origin_data, sizeof(unsigned char), Size*Size*BytesPerPixel, file);
fclose(file);

 if (!(fpi=fopen("ORIGIN/pepper.bmp","rb")))
```

```
      printf("The bmg Input file was not opened\n");
    else
      printf("The bmg Input file was opened\n");

    if (fpi != NULL) {

      WORD bfType;
      fread(&bfType, 1, sizeof(WORD), fpi);
      fread(&strHead, 1, sizeof(tagBITMAPFILEHEADER), fpi);
      fread(&strInfo, 1, sizeof(tagBITMAPINFOHEADER), fpi);

      for (int nCounti = 0; nCounti<256; nCounti++) {

        fread((char *)&strPla[nCounti].rgbBlue, 1, sizeof(BYTE), fpi);
        fread((char *)&strPla[nCounti].rgbGreen, 1, sizeof(BYTE), fpi);
        fread((char *)&strPla[nCounti].rgbRed, 1, sizeof(BYTE), fpi);
        //fread((char *)&strPla[nCounti].rgbReserved, 1, sizeof(BYTE), fpi);
      }
      fread(test_data, sizeof(unsigned char), Size*Size*BytesPerPixel, fpi);
      fclose(fpi);
    }
    }



    while (1)
    {
cout<<"Type int the method to use "<<endl;
cout<<"0--mymethod"<<endl;
cout<<"1--deblocking_filte"<<endl;
cout<<"2--reapplying"<<endl;
cout<<"3--exit the program"<<endl;
cin>>method;

if (method==3) return 0;

for(index=1;index<=5;index++)
{

   if(BytesPerPixel==1) sprintf(input_name,"RAW/clock_pro_%d.raw",index);
   else sprintf(input_name,"RAW/pepper_pro_%d.raw",index);
   //if(BytesPerPixel==1) sprintf(input_name,"RAW/clock_pro_%d.raw",index);
    //else sprintf(input_name,"RAWDATA/pepper%d.raw",index);

    if (!(file=fopen(input_name,"rb")))
    {
      cout << "Cannot open file: " << input_name <<endl;
      exit(1);
    }
    fread(compare_data, sizeof(unsigned char), Size*Size*3, file);
    fclose(file);
```

```
        for (i=0;i<Size;i++)
                {
                  for (j=0;j<Size;j++)
                  {
                  if (BytesPerPixel==1) MSE[0]+=(double)(origin_data[i][j][0]-compare_data[i][j]
[0])*(double)(origin_data[i][j][0]-compare_data[i][j][0]);
                  else MSE[0]+=(double)(origin_data[i][j][0]-compare_data[i][j][0])*(double)
(origin_data[i][j][0]-compare_data[i][j][0])
                              +(double)(origin_data[i][j][1]-compare_data[i][j][1])*(double)(origin_data[i]
[j][1]-compare_data[i][j][1])
                              +(double)(origin_data[i][j][2]-compare_data[i][j][2])*(double)(origin_data[i]
[j][2]-compare_data[i][j][2]);
                  }
               }

        MSE[0]=MSE[0]/(Size*Size*BytesPerPixel);
        PSNR[0]=10*log10(255*255/MSE[0]);
        MSE[0]=0;
        cout<< PSNR[0]<<endl;

        for(i=0;i<Size;i++)
        {
           for(j=0;j<Size;j++)
           {
              Converdata[i][j][0]=compare_data[i][j][0];
              Converdata[i][j][1]=compare_data[i][j][1];
              Converdata[i][j][2]=compare_data[i][j][2];
           }

        }
        /*My method of using 4 block gussian filter*/
        if (method==0)

        {

           for (int i=7;i<Size-1;i+=N)
              {
                 for (int j=0;j<Size;j++)
                 {
                   Blur_9(i,j,BytesPerPixel);
                   Blur_9(i+1,j,BytesPerPixel);
                 }

              }

           for (int j=7;j<Size-1;j+=N)
              {
                 for (int i=0;i<Size;i++)
                 {
                   Blur_9(i,j,BytesPerPixel);
                   Blur_9(i,j+1,BytesPerPixel);
```

```
            }

        }

}
if (method==1)
{

    int v_count=0;

    int v_max;
    int v_min;
    int p;

    int F=0;
    int T1=2;
    int T2=6;

    for(p=0;p<BytesPerPixel;p++)
    {
        v[10][p]={0};
        v_new[10][p]={0};
        a[4][3]={0};
        QP=0;
        c1=2;
        c2=5;
        c3=8;

        for (i=7;i<Size-1;i+=N)
        {
            for (j=0;j<Size;j++)
            {
                v[0][p]=Converdata[i-4][j][p];
                v[1][p]=Converdata[i-3][j][p];
                v[2][p]=Converdata[i-2][j][p];
                v[3][p]=Converdata[i-1][j][p];
                v[4][p]=Converdata[i][j][p];
                v[5][p]=Converdata[i+1][j][p];
                v[6][p]=Converdata[i+2][j][p];
                v[7][p]=Converdata[i+3][j][p];
                v[8][p]=Converdata[i+4][j][p];
                v[9][p]=Converdata[i+5][j][p];
                QP=Qe[(i+1)%8][j%8];
                v_max= MAX_FUN(10,p);
                v_min= MIN_FUN(10,p);
                for (k=0;k<3;k++)
                {
                    a[3][k]=(c1*v[2*k+1][p]-c2*v[2*k+2][p]+c2*v[2*k+3][p]-c1*v[2*k+4][p])/c3;
                }
                for(k=0;k<8;k++)
                {
```

```
            if (abs(v[k][p]-v[k+1][p])<=T1)
            {
               F++;
            }
         }
         if(F>=T2)
         {
           if(abs(v_max-v_min)<2*QP)
           {

            Smooth_region(p);
            Converdata[i-4][j][p]=v_new[0][p];
            Converdata[i-3][j][p]=v_new[1][p];
            Converdata[i-2][j][p]=v_new[2][p];
            Converdata[i-1][j][p]=v_new[3][p];
            Converdata[i][j][p]=v_new[4][p];
            Converdata[i+1][j][p]=v_new[5][p];
            Converdata[i+2][j][p]=v_new[6][p];
            Converdata[i+3][j][p]=v_new[7][p];
            Converdata[i+4][j][p]=v_new[8][p];
            Converdata[i+5][j][p]=v_new[9][p];
            F=0;
           }
         }
         else
         {
           if(a[3][1]<QP)
           {
           Default_region(p);
           Converdata[i][j][p]=v_new[4][p];
           Converdata[i+1][j][p]=v_new[5][p];
           }
           F=0;
         }
      }
   }

   for (j=7;j<Size-1;j+=N)
   {
      for (i=0;i<Size;i++)
      {
         v[0][p]=Converdata[i][j-4][p];
         v[1][p]=Converdata[i][j-3][p];
         v[2][p]=Converdata[i][j-2][p];
         v[3][p]=Converdata[i][j-1][p];
         v[4][p]=Converdata[i][j][p];
         v[5][p]=Converdata[i][j+1][p];
         v[6][p]=Converdata[i][j+2][p];
         v[7][p]=Converdata[i][j+3][p];
         v[8][p]=Converdata[i][j+4][p];
         v[9][p]=Converdata[i][j+5][p];
         QP=Qe[(i)%8][(j+1)%8];
```

```
                v_max= MAX_FUN(10,p);
                v_min= MIN_FUN(10,p);
                for (k=0;k<3;k++)
                {
                   a[3][k]=(c1*v[2*k+1][p]-c2*v[2*k+2][p]+c2*v[2*k+3][p]-c1*v[2*k+4][p])/c3;
                }
                for(k=0;k<8;k++)
                {
                   if (abs(v[k][p]-v[k+1][p])<=T1)
                   {
                      F++;
                   }
                }
                if(F>=T2)
                {
                  if(abs(v_max-v_min)<2*QP)
                  {
                   Smooth_region(p);
                   Converdata[i][j-4][p]=v_new[0][p];
                   Converdata[i][j-3][p]=v_new[1][p];
                   Converdata[i][j-2][p]=v_new[2][p];
                   Converdata[i][j-1][p]=v_new[3][p];
                   Converdata[i][j][p]=v_new[4][p];
                   Converdata[i][j+1][p]=v_new[5][p];
                   Converdata[i][j+2][p]=v_new[6][p];
                   Converdata[i][j+3][p]=v_new[7][p];
                   Converdata[i][j+4][p]=v_new[8][p];
                   Converdata[i][j+5][p]=v_new[9][p];
                  }
                   F=0;
                }
                else
                {
                  if(a[3][1]<QP)
                  {
                      Default_region(p);
                      Converdata[i][j][p]=v_new[4][p];
                      Converdata[i][j+1][p]=v_new[5][p];
                  }
                  F=0;
                }
             }
          }

    }

}

if(method==2)
{

   if(BytesPerPixel==1) sprintf(NEWinput_name,"NEWPRORAW/clock%d.raw",index);
```

```
            else sprintf(NEWinput_name,"NEWPRORAW/pepper%d.raw",index);
          //if(BytesPerPixel==1) sprintf(NEWinput_name,"PROPRORAW/clock%d.raw",index);
            //else sprintf(NEWinput_name,"PROPRORAW/clock%d.raw",index);

                if (!(file=fopen(NEWinput_name,"rb")))
                {
                   cout << "Cannot open file: " << input_name <<endl;
                   exit(1);
                }
                fread(Converdata, sizeof(unsigned char), Size*Size*3, file);
                fclose(file);

          }



          for (i=0;i<Size;i++)
                {
                   for (j=0;j<Size;j++)
                   {
                   if (BytesPerPixel==1) MSE[index]+=(double)(origin_data[i][j][0]-Converdata[i][j]
[0])*(double)(origin_data[i][j][0]-Converdata[i][j][0]);
                   /*else MSE[index]+=(double)(origin_data[i][j][0]-Converdata[i][j][0])*(double)
(origin_data[i][j][0]-Converdata[i][j][0])
                              +(double)(origin_data[i][j][1]-Converdata[i][j][1])*(double)(origin_data[i][j]
[1]-Converdata[i][j][1])
                              +(double)(origin_data[i][j][2]-Converdata[i][j][2])*(double)(origin_data[i][j]
[2]-Converdata[i][j][2]);*/
                   else MSE[index]+=(double)(origin_data[i][j][0]-Converdata[i][j][2])*(double)
(origin_data[i][j][0]-Converdata[i][j][2])
                              +(double)(origin_data[i][j][1]-Converdata[i][j][1])*(double)(origin_data[i][j]
[1]-Converdata[i][j][1])
                              +(double)(origin_data[i][j][2]-Converdata[i][j][0])*(double)(origin_data[i][j]
[2]-Converdata[i][j][0]);


                   }
                }

                MSE[index]=MSE[index]/(Size*Size*BytesPerPixel);
                PSNR[index]=10*log10(255*255/MSE[index]);
                MSE[index]=0;
                cout<< PSNR[index]<<endl;


/*
                unsigned char tem[Size][Size][3];
                 for (i=0;i<Size;i++)
                      {
                         for(j=0;j<Size;j++)
                         {
                            tem[i][j][0]=Converdata[Size-1-i][j][2];
```

```
                tem[i][j][1]=Converdata[Size-1-i][j][1];
                tem[i][j][2]=Converdata[Size-1-i][j][0];

            }

        }

    for (i=0;i<Size;i++)
        {
            for(j=0;j<Size;j++)
            {
                Converdata[i][j][0]=tem[i][j][0];
                Converdata[i][j][1]=tem[i][j][1];
                Converdata[i][j][2]=tem[i][j][2];

            }

        }*/

    if(BytesPerPixel==1)
        {sprintf(output_name, "PROBMP/clock_pro%d.bmp", index);}
    else
        {sprintf(output_name, "PROBMP/pepper_pro%d.bmp", index);}

    if (!(fpi=fopen(output_name,"wb")))
        printf("The output file was not opened\n");
    else
        printf("The output file was opened\n");

    WORD bfType = 0x4d42;
    fwrite(&bfType, 1, sizeof(WORD), file);
    //fpw +=2;
    fwrite(&strHead, 1, sizeof(tagBITMAPFILEHEADER), file);
    fwrite(&strInfo, 1, sizeof(tagBITMAPINFOHEADER), file);
    for (int nCounti = 0; nCounti<strInfo.biClrUsed; nCounti++) {

        fwrite(&strPla[nCounti].rgbBlue, 1, sizeof(BYTE), file);
        fwrite(&strPla[nCounti].rgbGreen, 1, sizeof(BYTE), file);
        fwrite(&strPla[nCounti].rgbRed, 1, sizeof(BYTE), file);
        //fwrite(&strPla[nCounti].rgbReserved, 1, sizeof(BYTE), file);
      }
    fwrite(Converdata, sizeof(unsigned char), Size*Size * 3, file);
    //fwrite(test_data, sizeof(unsigned char), Size*Size * 3, file);
    fclose(file);

}

  }



  return 0;
```

}