```
// MFCApplication2Dlg.cpp : implementation file

/*

# EE669 Homework Assignment #1

# Feb 7, 2015

# Name: Shanglin Yang

# ID: 3795329308

# Email: shangliy@usc.edu

# compiled on WINDOWS 8 with Visul Studio 2015

*/

//

#include "stdafx.h"

#include "MFCApplication2.h"

#include "MFCApplication2Dlg.h"
```

```
#include "afxdialogex.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 6000000

using namespace std;
```

```
#ifdef _DEBUG

#define new DEBUG_NEW

#endif

errno_t err;   //Symbol of open_c

char *FILE_name; //Input file name

BIT_FILE *bit_file; //The pointer of file name

char flag_MTF = 0; //The flag of Move to front

double input_size = 0; // The size of input files

double output_size = 0;// The size of output files

double countnum = 0;

//Open files

BIT_FILE *OpenOutputBitFile(const 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);

}
```

```
//Output codes bit by bit

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;
    }
}
//Output codes each time count bits
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;
    }
}
//close output files
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);

}

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];

/*Shannon Function*/

/*---------------------------------------------------------------------------*/

class SYMBOL    //the SHANNON node construction

{

public:

    SYMBOL* left;   //the left side

    SYMBOL* right;  //the right side

    unsigned int pro;  //the weight of the node

    unsigned char name; //the symbol name of the node

    unsigned char code;  //the final code name of the node
    SYMBOL() { left = right = NULL; pro = 0; name = '\0'; code = NULL; };  //initial the
new constrction
    SYMBOL(SYMBOL* l, SYMBOL* r, int p, char n, char co) { left = l;    right = r;     pro
= p; name = n; code = co; } //send data to the construction
    ~SYMBOL() { delete left; delete right; } //delete the construction
};
typedef vector<SYMBOL*> SYMVector; // the vector array of the symbols
SYMVector SymArr;
/*Re-sort the symbol array*/
void SYsort_ARR(SYMVector &pro_data, double L)
{
    int i, j;
    int min = 0;
int temple[2];

for (i = 0; i<L - 1; i++)

{

    for (j = i; j<L; j++)

    {

        if ((pro_data[j]->pro)<(pro_data[i]->pro))

        {

            temple[0] = pro_data[i]->pro;

            temple[1] = pro_data[i]->name;

            pro_data[i]->pro = pro_data[j]->pro;

            pro_data[i]->name = pro_data[j]->name;

            pro_data[j]->pro = temple[0];

            pro_data[j]->name = temple[1];
            }

        }

    }

    return;

}

/*Find the least two symbol of the new array,then build the tree*/

//root: current node address

//le: the left index

//ri: the right index

void Root_find(SYMBOL &root, double *sum_p, int le, int ri)

{

    int i;

    int N;
```

```
double min_dif = INF;

double dif = 0;

int pos = 0;

double sum_tem[257] = { 0 };

int size;

for (i = le; i <= ri; i++)

    sum_tem[i] = *(sum_p + i);

SYMBOL *left_s = new SYMBOL;

SYMBOL *right_s = new SYMBOL;

N = ri - le;

switch (N)

{

case 0: {
        root = *SymArr[le - 1];  //there are only one symbol left, set it as the bottom root

    }return;

    case 1:

    {

        left_s = SymArr[le - 1];  //there are two symbol left , set both of them as the bottom

root

        right_s = SymArr[ri - 1];

        root.left = left_s;

        root.right = right_s;

    }return;

    default: {

        for (i = (le); i <= ri; i++)

        {
    }
        dif = 2 * (*(sum_p + i)) - ((*(sum_p + le - 1)) + (*(sum_p + ri)));
        if (abs(dif)<min_dif)
        {
            min_dif = abs(dif);
            pos = i;
        }
    }
    root.left = left_s;             //initial the new left root;
    root.right = right_s;           //initial the new right root;
    Root_find(*left_s, sum_p, le, pos); // Fine new root of the left sides;
    Root_find(*right_s, sum_p, pos + 1, ri); // Find new root of the right sides;
}
}
```

```
    return;

}

/*Generate the shannon CODE according to the binary tree*/

void shannon_generate_code(SYMBOL &root, Sample_Code&scode)  //Generate the

code according to the root

{

    int i;

    if (((root.left) == NULL) && ((root.right) == NULL)) //Achieving the bottom root

    {

        (CODE[root.name]).code = scode; //Send code to the Code vector

        (CODE[root.name]).name = root.name; //Send name to Code vector

        return;

    }
    Sample_Code lcode = scode;  //Hermit the original root to the left

    Sample_Code rcode = scode;  //Hermit the original root to the riht

    lcode.push_back(false);

    rcode.push_back(true);

    shannon_generate_code(*root.left, lcode);

    shannon_generate_code(*root.right, rcode);

}

/*---------------------------------------------------------------------------*/

/*Huffman Function*/

/*---------------------------------------------------------------------------*/

//the structure of Huffman node

class HT_NODE

{
```

```
public:

    HT_NODE* left;  //left node in the tree

    HT_NODE* right; //right node in the tree

    HT_NODE* parent; //parent node in the tree

    int name;  //node name (symbol)

    double weight; //node weight

    int order; //node 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*/ //len is the length of huffman node vector

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

}

/*--------------------------------------------------------------------------*/

/*Adaptive Huffman Function*/

/*--------------------------------------------------------------------------*/

HT_NODE *Tree_node[512]; // The node of huffman  tree

HT_NODE *NY_NODE = new HT_NODE(NULL, NULL, NULL, 0, 0, 512);// The NYT

node
```

```
int NUM = -1;  //initial the node number

int order = 512;//initial the node orderr

/*Generate the adaptive Huffman CODE according to the root tree*/

void AD_generate_code(HT_NODE *root, int name,Sample_Code&scode)

{

   if ((root->parent)->order == 512)

   {

      if ((root->parent)->left == root) scode.push_back(false);

      else scode.push_back(true);

      (CODE[name]).code = scode;

      (CODE[name]).name = name;

      scode.clear();

      return;
```

```
    }
    if ((root->parent)->left == root) scode.push_back(false);
    else scode.push_back(true);
    AD_generate_code(root->parent, name, scode);
}
/*Swap two node of the tree*/
void Swap_node(HT_NODE *Na_parent, HT_NODE *Nb_parent, HT_NODE *Node_a,
HT_NODE *Node_b)
{
    HT_NODE *Temple_left = new HT_NODE;
    HT_NODE *Temple_right = new HT_NODE;
    HT_NODE *Temple_parent = new HT_NODE;
    int order_temple;
```

```
if (Na_parent->left == Node_a)
{
    Na_parent->left = Node_b;
}
else
{
    Na_parent->right = Node_b;
}
if (Nb_parent->left == Node_b)
{
    Nb_parent->left = Node_a;
}
else
```

```
    {
        Nb_parent->right = Node_a;
    }
    Temple_parent = Node_a->parent;
    order_temple = Node_a->order;
    Node_a->parent = Node_b->parent;
    Node_a->order = Node_b->order;
    Node_b->parent = Temple_parent;
    Node_b->order = order_temple;
}
/*Update the huffman tree*/
void Update_tree(HT_NODE *NODE_root)
{
    int i, j;

    int index;

    int ORDER_MAX = 0;

    ORDER_MAX = NODE_root->order;

    if (NODE_root->order == 512)

    {

        NODE_root->weight++;

        return;

    }

    for (i = 1; i <= NUM; i++)

    {

        if ((Tree_node[i]->weight == NODE_root->weight) && (Tree_node[i] !=

NODE_root->parent))//check the max order of the weight class
```

```
            if (Tree_node[i]->order > ORDER_MAX) ORDER_MAX = Tree_node[i]->order,
index = i;
    }
    if (NODE_root->order == ORDER_MAX) //if it is the max order , just add weight
        NODE_root->weight++;
    else  //if not , swap the two node
    {
        Swap_node(Tree_node[index]->parent, NODE_root->parent, Tree_node[index],
NODE_root);
        NODE_root->weight++;
    }
    Update_tree(NODE_root->parent);  //change it to its parent node
    return;
```

```
}

/*add new node to the huffman tree*/

void Add_Root(unsigned symbol)

{

    int i;

    Sample_Code scode;

    HT_NODE *newNY_NODE = new HT_NODE;  //get new NYT node

    HT_NODE *newROOT_NODE = new HT_NODE;

    NY_NODE->left = newNY_NODE;  //change current NYT to the new node

    NY_NODE->right = newROOT_NODE;

    Tree_node[++NUM] = NY_NODE;

    Tree_node[NUM]->weight++;  //add weight to original node

    newROOT_NODE->name = symbol;
    newROOT_NODE->order = --order;

    newROOT_NODE->weight = 1;

    newROOT_NODE->parent = Tree_node[NUM];

    Tree_node[++NUM] = newROOT_NODE; //new symbol weight++

    NY_NODE = newNY_NODE;

    NY_NODE->parent = Tree_node[NUM - 1];

    NY_NODE->order = --order;

    AD_generate_code(Tree_node[NUM], symbol, scode);//generate code for the new
symbol

    for (i = CODE[(symbol)].code.size() - 1; i >= 0; i--)

    {

        OutputBits(bit_file, symbol,8);

        countnum+=8;
    }

    if (Tree_node[NUM - 1]->order == 512) //the root node

    {

        return;

    }

    else Update_tree(NY_NODE->parent->parent); //update the tree

}

/*--------------------------------------------------------------------------*/

/*Run_length Function*/

/*--------------------------------------------------------------------------*/

//the structure of Run_length root node

class RL_NODE

{
```

```
public:

    unsigned char lenth; //the code length

    unsigned char code; //the code name

    RL_NODE() { lenth = 0; code = 0; };

    RL_NODE(unsigned char l, unsigned char c) { lenth = l; code = c; }

};

typedef vector<RL_NODE*> CODEVector;

CODEVector RLCode_arr;

typedef vector<int> BUFFVector;

BUFFVector buffvector; //Input buff vector

BUFFVector decovector; //Output buff vector

/*--------------------------------------------------------------------------*/

/*Modified Run_length Function*/
```

```
/*---------------------------------------------------------------------------*/

typedef vector<unsigned char> MLCODEVector;

MLCODEVector MLCode_arr;

BUFFVector SECbuffvector;

/*---------------------------------------------------------------------------*/

/*Move to front HUffman Function*/

/*---------------------------------------------------------------------------*/

class CODE_TABLE

{

public:

    int index;//code index

    int code;

    CODE_TABLE() { index = 0; code = 0; };
    CODE_TABLE(unsigned char I, unsigned char c) { index = I; code = c; }

}code_table[256];

typedef vector<unsigned char> MVCOVector;

MVCOVector MvCode_arr;

/*----------------------------------------------------------------------*/

// CAboutDlg dialog used for App About

class CAboutDlg : public CDialogEx

{

public:

    CAboutDlg();

// Dialog Data

    enum { IDD = IDD_ABOUTBOX };

    protected:
```

```
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

// Implementation

protected:

    DECLARE_MESSAGE_MAP()

};

CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)

{

}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)

{

    CDialogEx::DoDataExchange(pDX);

}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
```

```
END_MESSAGE_MAP()

// CMFCApplication2Dlg dialog

CMFCApplication2Dlg::CMFCApplication2Dlg(CWnd* pParent /*=NULL*/)

    : CDialogEx(CMFCApplication2Dlg::IDD, pParent)

{

    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);

}

void CMFCApplication2Dlg::DoDataExchange(CDataExchange* pDX)

{

    CDialogEx::DoDataExchange(pDX);

}

BEGIN_MESSAGE_MAP(CMFCApplication2Dlg, CDialogEx)

    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()

    ON_WM_QUERYDRAGICON()

ON_BN_CLICKED(IDC_BUTTON1, &CMFCApplication2Dlg::OnBnClickedButton1)

ON_BN_CLICKED(IDC_BUTTON2, &CMFCApplication2Dlg::OnBnClickedButton2)

ON_BN_CLICKED(IDC_BUTTON4, &CMFCApplication2Dlg::OnBnClickedButton4)

ON_BN_CLICKED(IDC_BUTTON5, &CMFCApplication2Dlg::OnBnClickedButton5)

ON_BN_CLICKED(IDC_BUTTON6, &CMFCApplication2Dlg::OnBnClickedButton6)

ON_BN_CLICKED(IDC_BUTTON8, &CMFCApplication2Dlg::OnBnClickedButton8)

ON_BN_CLICKED(IDC_BUTTON10, &CMFCApplication2Dlg::OnBnClickedButton10)

ON_BN_CLICKED(IDC_BUTTON11, &CMFCApplication2Dlg::OnBnClickedButton11)

ON_BN_CLICKED(IDC_RADIO1, &CMFCApplication2Dlg::OnBnClickedRadio1)

ON_BN_CLICKED(IDC_RADIO2, &CMFCApplication2Dlg::OnBnClickedRadio2)

ON_BN_CLICKED(IDC_RADIO3, &CMFCApplication2Dlg::OnBnClickedRadio3)
```

```
ON_BN_CLICKED(IDC_RADIO4, &CMFCApplication2Dlg::OnBnClickedRadio4)

ON_BN_CLICKED(IDOK, &CMFCApplication2Dlg::OnBnClickedOk)

ON_EN_CHANGE(IDC_EDIT5, &CMFCApplication2Dlg::OnEnChangeEdit5)

ON_EN_CHANGE(IDC_EDIT7, &CMFCApplication2Dlg::OnEnChangeEdit7)

ON_BN_CLICKED(IDC_BUTTON15, &CMFCApplication2Dlg::OnBnClickedButton15)

ON_BN_CLICKED(IDC_RADIO7, &CMFCApplication2Dlg::OnBnClickedRadio7)

ON_BN_CLICKED(IDC_RADIO8, &CMFCApplication2Dlg::OnBnClickedRadio8)

ON_EN_CHANGE(IDC_EDIT8, &CMFCApplication2Dlg::OnEnChangeEdit8)

END_MESSAGE_MAP()

// CMFCApplication2Dlg message handlers

BOOL CMFCApplication2Dlg::OnInitDialog()

{

   CDialogEx::OnInitDialog();
```

```
// Add "About..." menu item to system menu.

// IDM_ABOUTBOX must be in the system command range.

ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);

ASSERT(IDM_ABOUTBOX < 0xF000);

CMenu* pSysMenu = GetSystemMenu(FALSE);

if (pSysMenu != NULL)

{

    BOOL bNameValid;

    CString strAboutMenu;

    bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);

    ASSERT(bNameValid);

    if (!strAboutMenu.IsEmpty())

    {
}

            pSysMenu->AppendMenu(MF_SEPARATOR);

            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);

        }

    }

    // Set the icon for this dialog.  The framework does this automatically

    //  when the application's main window is not a dialog

    SetIcon(m_hIcon, TRUE);            // Set big icon

    SetIcon(m_hIcon, FALSE);        // Set small icon

    // TODO: Add extra initialization here

    OnBnClickedRadio2();

    CheckDlgButton(IDC_RADIO2, 1);

    return TRUE;  // return TRUE  unless you set the focus to a control

}
```

```
void CMFCApplication2Dlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialogEx::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,

//  this is automatically done for you by the framework.

void CMFCApplication2Dlg::OnPaint()

{

    if (IsIconic())

    {

        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND,

reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

        // Center icon in client rectangle

        int cxIcon = GetSystemMetrics(SM_CXICON);

        int cyIcon = GetSystemMetrics(SM_CYICON);

        CRect rect;
        GetClientRect(&rect);

        int x = (rect.Width() - cxIcon + 1) / 2;

        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon

        dc.DrawIcon(x, y, m_hIcon);

    }

    else

    {

        CDialogEx::OnPaint();

    }

}

// The system calls this function to obtain the cursor to display while the user drags

//  the minimized window.
```

```
HCURSOR CMFCApplication2Dlg::OnQueryDragIcon()

{

    return static_cast<HCURSOR>(m_hIcon);

}

// Choose the Input files name for the input

// This is the first step

void CMFCApplication2Dlg::OnBnClickedRadio1()

{

    // TODO: Add your control notification handler code here

    CheckDlgButton(IDC_RADIO2, 0);

    CheckDlgButton(IDC_RADIO3, 0);

    CheckDlgButton(IDC_RADIO4, 0);

    GetDlgItem(IDC_BUTTON10)->EnableWindow(FALSE);
    GetDlgItem(IDC_BUTTON1)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON4)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON2)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON5)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON6)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON15)->EnableWindow(FALSE);

    FILE_name = "audio.dat";

}

void CMFCApplication2Dlg::OnBnClickedRadio2()

{

    // TODO: Add your control notification handler code here

    CheckDlgButton(IDC_RADIO1, 0);

    CheckDlgButton(IDC_RADIO3, 0);
    CheckDlgButton(IDC_RADIO4, 0);

    GetDlgItem(IDC_BUTTON10)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON1)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON4)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON2)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON5)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON6)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON15)->EnableWindow(FALSE);

    FILE_name = "text.dat";

}

void CMFCApplication2Dlg::OnBnClickedRadio3()

{

    // TODO: Add your control notification handler code here
    CheckDlgButton(IDC_RADIO1, 0);

    CheckDlgButton(IDC_RADIO2, 0);

    CheckDlgButton(IDC_RADIO4, 0);

    GetDlgItem(IDC_BUTTON10)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON1)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON4)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON2)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON5)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON6)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON15)->EnableWindow(FALSE);

    FILE_name = "binary.dat.raw";

}

void CMFCApplication2Dlg::OnBnClickedRadio4()
```

```
{

    // TODO: Add your control notification handler code here

    CheckDlgButton(IDC_RADIO2, 0);

    CheckDlgButton(IDC_RADIO3, 0);

    CheckDlgButton(IDC_RADIO1, 0);

    GetDlgItem(IDC_BUTTON10)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON1)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON4)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON2)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON5)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON6)->EnableWindow(FALSE);

    GetDlgItem(IDC_BUTTON15)->EnableWindow(FALSE);

    FILE_name = "image.dat.raw";
}

void CMFCApplication2Dlg::OnBnClickedOk()

{

    // TODO: Add your control notification handler code here

    CDialogEx::OnOK();

    //system("pause");

}

void CMFCApplication2Dlg::OnCbnSelchangeCombo2()

{

    // TODO: Add your control notification handler code here

}

/*Output the information*/

void CMFCApplication2Dlg::OnEnChangeEdit5()
```

```
{
    // TODO:  If this is a RICHEDIT control, the control will not
    // send this notification unless you override the CDialogEx::OnInitDialog()
    // function and call CRichEditCtrl().SetEventMask()
    // with the ENM_CHANGE flag ORed into the mask.
    // TODO:  Add your control notification handler code here
}
void CMFCApplication2Dlg::OnEnChangeEdit6()
{
    // TODO:  If this is a RICHEDIT control, the control will not
    // send this notification unless you override the CDialogEx::OnInitDialog()
    // function and call CRichEditCtrl().SetEventMask()
    // with the ENM_CHANGE flag ORed into the mask.
    // TODO:  Add your control notification handler code here

}

void CMFCApplication2Dlg::OnEnChangeEdit7()

{

    // TODO:  If this is a RICHEDIT control, the control will not

    // send this notification unless you override the CDialogEx::OnInitDialog()

    // function and call CRichEditCtrl().SetEventMask()

    // with the ENM_CHANGE flag ORed into the mask.

    // TODO:  Add your control notification handler code here

}

void CMFCApplication2Dlg::OnEnChangeEdit8()

{

    // TODO:  If this is a RICHEDIT control, the control will not
    // send this notification unless you override the CDialogEx::OnInitDialog()

    // function and call CRichEditCtrl().SetEventMask()

    // with the ENM_CHANGE flag ORed into the mask.

    // TODO:  Add your control notification handler code here

}

/*Input the files*/

void CMFCApplication2Dlg::OnBnClickedButton8()

{

    // TODO: Add your control notification handler code here

    int i;

    CString s; //Temple string

    CString strshow;

    int *Buff;
```

```
double sam_weight[256] = { 0 };

double sum_weight = 0;

double entro = 0;

countnum = 0;

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

//Input the file

ifstream infile(FILE_name, ios::binary);

if (!infile)

{

    cerr << "open error!" << endl;

    abort();

}

while (infile.peek() != EOF)
```

```
{
    infile.read((char*)Buff, sizeof(char));
    sam_weight[*Buff]++; //calculate each weight of the symbol
    sum_weight++;
}
infile.close();
delete Buff;
input_size = sum_weight;
s.Format(_T("%lf"), input_size);
strshow = s;
SetDlgItemText(IDC_EDIT6, strshow);
/*Active the compress process*/
GetDlgItem(IDC_BUTTON10)->EnableWindow(TRUE);
```

```
    GetDlgItem(IDC_BUTTON1)->EnableWindow(TRUE);

    GetDlgItem(IDC_BUTTON4)->EnableWindow(TRUE);

    GetDlgItem(IDC_BUTTON2)->EnableWindow(TRUE);

    GetDlgItem(IDC_BUTTON5)->EnableWindow(TRUE);

    GetDlgItem(IDC_BUTTON6)->EnableWindow(TRUE);

    if(FILE_name == "image.dat.raw")

    GetDlgItem(IDC_BUTTON15)->EnableWindow(TRUE);

}

/*Calculate the entropy for the inpputfiles*/

void CMFCApplication2Dlg::OnBnClickedButton11()

{

    // TODO: Add your control notification handler code here

    // TODO: Add your control notification handler code here
```

```
int i;

CString s;

CString strWebsiteSel;

int *Buff;

double sam_weight[256] = { 0 };

double sum_weight = 0;

double entro = 0;

countnum = 0;

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));

        sam_weight[*Buff]++;

        sum_weight++;

    }

    /*Calculate the Entropy for the file*/

    for (i = 0; i<256; i++)

    {   if (sam_weight[i])

        entro = entro +( sam_weight[i] / (sum_weight))*(log(sam_weight[i] / (sum_weight))
/ log(2.0));
```

```
    }

    infile.close();

    delete Buff;

    s.Format(_T("%lf"), -entro);

    strWebsiteSel = s;

    SetDlgItemText(IDC_EDIT5, strWebsiteSel);

}

/*Compress the files with shannon algorism*/

void CMFCApplication2Dlg::OnBnClickedButton1()

{

    double sam_weight[256] = { 0 };  //Record the weight of each sample

    double sum_weight = 0; //Record the total weight of the files

    double sum_pro[257] = { 0 }; //Record the sum of the sample weight
```

```
int i, j;

int *Buff; //Input Buff

Sample_Code scode; //Code vector

SYMBOL *root = new SYMBOL;

CString s;

CString strWebsiteSel;

countnum = 0;

Buff = (int*)calloc(sizeof(int), 1);  //Initial the Buff space

ifstream infile(FILE_name, ios::binary);

if (!infile)

{

    cerr << "open error!" << endl;

    abort();
    }

    while (infile.peek() != EOF)

    {

        infile.read((char*)Buff, sizeof(char));  //Input the datas to the buff

        sam_weight[*Buff]++; //Compute the weight of the each sample

        sum_weight++;

    }

    for (i = 0, j = 0; i<256; i++)

    {

        if (sam_weight[i]) {

            SymArr.push_back(new SYMBOL(NULL, NULL, sam_weight[i], (char)i, NULL));

//put sample into the symbol array

        }
```

```
}

SYsort_ARR(SymArr, SymArr.size());

for (i = 1; i <= SymArr.size(); i++)

{

    sum_pro[i] = sum_pro[i - 1] + SymArr[i - 1]->pro; //Calculate the sum_weight

}

Root_find(*root, sum_pro, 1, SymArr.size());  //Build the tree

shannon_generate_code(*root, scode); //Generate the code

scode.clear();

SymArr.clear();

strWebsiteSel += _T("Shan_com_");

strWebsiteSel += FILE_name;

string str;
```

```
str = CT2A(strWebsiteSel.GetBuffer());

const char * outFILE_name = str.c_str();

bit_file = OpenOutputBitFile(outFILE_name);

infile.clear();

infile.seekg(0);

if (!infile)

{

    cerr << "open error!" << endl;

    abort();

}

while (infile.peek() != EOF)

{

    infile.read((char*)Buff, sizeof(char));
    for (i = 0; i < CODE[(*Buff)].code.size(); i++)

    {

        OutputBit(bit_file, CODE[(*Buff)].code[i]);

        countnum++;

    }

}

infile.close();

CloseOutputBitFile(bit_file);

output_size = countnum / 8;

s.Format(_T("%lf"), output_size);

strWebsiteSel = s;

SetDlgItemText(IDC_EDIT7, strWebsiteSel);

s.Format(_T("%lf"), (output_size / input_size) * 100);
```

```
    strWebsiteSel = s;

    SetDlgItemText(IDC_EDIT8, strWebsiteSel);

}

/*Compress the files with Huffman algorism*/

void CMFCApplication2Dlg::OnBnClickedButton10()

{

    // TODO: Add your control notification handler code here

    int i;

    int k = 0;

    int *Buff;

    double sam_weight[256] = { 0 };

    double sum_weight = 0;

    Sample_Code scode;
```

```
CString str_filename;

CString s;

string str;

countnum = 0; //number of bit;

    /*Input the sample date*/

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));
        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();

/*Generate the outfile name*/

str_filename += _T("HUFF_comp_");

str_filename += FILE_name;

str = CT2A(str_filename.GetBuffer());

const char * outFILE_name = str.c_str();

bit_file = OpenOutputBitFile(outFILE_name);

/*Rescan the input files*/

infile.clear();

infile.seekg(0);
```

```
if (!infile)
{
    cerr << "open error!" << endl;
    abort();
}
while (infile.peek() != EOF)
{
    infile.read((char*)Buff, sizeof(char));
    for (i = 0; i < CODE[(*Buff)].code.size(); i++)
    {
        OutputBit(bit_file, CODE[(*Buff)].code[i]);
        countnum++;
    }
    }

    infile.close();

    CloseOutputBitFile(bit_file);

    output_size = countnum / 8;

    s.Format(_T("%lf"), output_size);

    str_filename = s;

    SetDlgItemText(IDC_EDIT7, str_filename);

    s.Format(_T("%lf"), (output_size / input_size) * 100);

    str_filename = s;

    SetDlgItemText(IDC_EDIT8, str_filename);

}

/*Compress the files with Adaptive Huffman algorism*/

void CMFCApplication2Dlg::OnBnClickedButton4()
```

```
{

    // TODO: Add your control notification handler code here

    int i;

    int j;

    int k;

    int temple_index = 0;

    int *Buff;

    double sam_weight[256] = { 0 };

    double sum_weight = 0;

    Sample_Code scode;

    CString s;

    CString strWebsiteSel;

    HT_NODE *address;
```

```
countnum = 0;

/*Initial data*/

NUM = -1;

order = 512;

for (i = 0; i < 512; i++)

    Tree_node[i] = new HT_NODE;

NY_NODE = new HT_NODE(NULL, NULL, NULL, 0, 0, 512);

for (i = 0; i < 256; i++)

{

    code_table[i].code = i;

    code_table[i].index = i;

}

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

```
}

ifstream infile(FILE_name, ios::binary);

/*Generate the outfile name*/

strWebsiteSel += _T("AT_Huff_com");

strWebsiteSel += FILE_name;

string str;

str = CT2A(strWebsiteSel.GetBuffer());

const char * outFILE_name = str.c_str();

bit_file = OpenOutputBitFile(outFILE_name);

if (!infile)

{

    cerr << "open error!" << endl;

    abort();

}
```

```
    if (flag_MTF == 1) //using MTF algorism

    {

        while (infile.peek() != EOF)

        {

            infile.read((char*)Buff, sizeof(char));

            buffvector.push_back(*Buff);//PUSH data to the input buff vector

            MvCode_arr.push_back(code_table[*Buff].index);//PUSH code to the code buff
vector

            temple_index = code_table[*Buff].index;

            for (i = 0; i < 256; i++) //update table

            {

                if (code_table[i].index < temple_index) code_table[i].index++;

            }
            code_table[*Buff].index = 0;
        }
        for (j = 0; j < MvCode_arr.size(); j++)
        {
            *Buff = MvCode_arr[j]; //Using new code vector as input
            if (!sam_weight[*Buff])
            {
                Add_Root(*Buff);
            }
            else
            {
                for (i = 1; i <= NUM; i++)
                {
                }
                  if (Tree_node[i]->name == *Buff)
                  {
                      address = Tree_node[i];
                      AD_generate_code(Tree_node[i], *Buff, scode);
                      for (k = CODE[(*Buff)].code.size() - 1; k >= 0; k--)
                      {
                          OutputBit(bit_file, CODE[(*Buff)].code[k]);
                          countnum++;
                      }
                  }
              }
              Update_tree(address);
          }
```

```
        sam_weight[*Buff]++;

        sum_weight++;

    }

}

else{

while (infile.peek() != EOF)

{

    infile.read((char*)Buff, sizeof(char));

    if (!sam_weight[*Buff]) //new symbol

    {

        Add_Root(*Buff);

    }

    else
```

```
        {
            for (i = 1; i <= NUM; i++)
            {
                if (Tree_node[i]->name == *Buff)
                {
                    address = Tree_node[i]; //get the address of existing symbol
                    AD_generate_code(Tree_node[i], *Buff, scode); //generate code based on
old tree
                    for (j = CODE[(*Buff)].code.size() - 1; j >= 0; j--)
                    {
                        OutputBit(bit_file, CODE[(*Buff)].code[j]);
                        countnum++;
                    }
                }

            }

            Update_tree(address);

        }

        sam_weight[*Buff]++;

        sum_weight++;

    }

}

    infile.close();

    CloseOutputBitFile(bit_file);

    node_arr.clear();

    output_size = countnum/8;

    MvCode_arr.clear();
    buffvector.clear();

    /*Output message*/

    s.Format(_T("%lf"), output_size);

    strWebsiteSel = s;

    SetDlgItemText(IDC_EDIT7, strWebsiteSel);

    s.Format(_T("%lf"), (output_size / input_size) * 100);

    strWebsiteSel = s;

    SetDlgItemText(IDC_EDIT8, strWebsiteSel);

}

/*Compress the files with Run_length algorism*/

void CMFCApplication2Dlg::OnBnClickedButton6()

{

    // TODO: Add your control notification handler code here
```

```
int i;

int *Buff;

int buff_pre = INF;  //The pre symbol

double sam_weight[256] = { 0 };

double sum_weight = 0;

double count = 0;

unsigned long code = 0;

countnum = 0;

CString s;

CString strWebsiteSel;

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));
    buffvector.push_back(*Buff); //Push data to the input buff
    if (sum_weight == 0)
    {
        buff_pre = *Buff; //The first input
        count++;
    }
```

```
        else {

            if (*Buff == buff_pre&&count < 255) count++;

            else {

                RLCode_arr.push_back(new  RL_NODE(count, buff_pre));//Each time input

count and symbol

                count = 1;

                buff_pre = *Buff;

            }

        }

        sam_weight[*Buff]++;

        sum_weight++;

    }

    RLCode_arr.push_back(new  RL_NODE(count, buff_pre)); //the last input
```

```
}

/*Generate the outfile name*/

strWebsiteSel += _T("Runl_com_");

strWebsiteSel += FILE_name;

string str;

str = CT2A(strWebsiteSel.GetBuffer());

const char * outFILE_name = str.c_str();

bit_file = OpenOutputBitFile(outFILE_name);

for (i = 0; i < RLCode_arr.size(); i++)

{

    OutputBits(bit_file, RLCode_arr[i]->lenth, 8);

    OutputBits(bit_file, RLCode_arr[i]->code, 8);

    countnum += 16;

}
```

```
infile.close();

CloseOutputBitFile(bit_file);

delete Buff;

/*Decode the copressed ouputfiles */

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

ifstream deofile(outFILE_name, ios::binary);

if (!deofile)

{

    cerr << "open error!" << endl;

    abort();

}

while (deofile.peek() != EOF)

{
    deofile.read((char*)Buff, sizeof(char));

    count = *Buff;

    deofile.read((char*)Buff, sizeof(char));

    code = *Buff;

    for (i = 0; i < count; i++)

    {

        decovector.push_back(code);

    }

}

/*Generate the outfile name for decode*/

strWebsiteSel = _T("DecoRunl_com_");

strWebsiteSel += FILE_name;

str = CT2A(strWebsiteSel.GetBuffer());
```

```
const char * deFILE_name = str.c_str();

bit_file = OpenOutputBitFile(deFILE_name);

for (i = 0; i < decovector.size(); i++)

{

    OutputBits(bit_file, decovector[i], 8);

}

deofile.close();

CloseOutputBitFile(bit_file);

delete Buff;

/*Free the RL_code*/

for (i = 0; i < RLCode_arr.size(); i++)

{

    delete RLCode_arr[i];
    }

    RLCode_arr.clear();

    buffvector.clear();

    decovector.clear();

    output_size = countnum/8;

    s.Format(_T("%lf"), output_size);

    strWebsiteSel = s;

    SetDlgItemText(IDC_EDIT7, strWebsiteSel);

    s.Format(_T("%lf"), (output_size / input_size) * 100);

    strWebsiteSel = s;

    SetDlgItemText(IDC_EDIT8, strWebsiteSel);

}

/*Compress the files with Modified Run_length algorism*/
```

```
void CMFCApplication2Dlg::OnBnClickedButton5()

{

    // TODO: Add your control notification handler code here

    int i;

    int j;

    int k;

    int temple_index = 0;

    string file_name;

    int *Buff;

    int buff_pre = INF;

    double sam_weight[256] = { 0 };

    double sum_weight = 0;

    unsigned long code = 0;
```

```
double count = 0;

CString s;

CString strWebsiteSel;

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

ifstream infile(FILE_name, ios::binary);

/*Generate the outfile name for decode*/

strWebsiteSel += _T("MORunl_com_");

strWebsiteSel += FILE_name;

string str;

str = CT2A(strWebsiteSel.GetBuffer());

const char * outFILE_name = str.c_str();

bit_file = OpenOutputBitFile(outFILE_name);

countnum = 0;
```

```
for (i = 0; i < 256; i++)

{

    code_table[i].code = i;

    code_table[i].index = i;

}

if (!infile)

{

    cerr << "open error!" << endl;

    abort();

}

if (flag_MTF == 1)

{

    while (infile.peek() != EOF)
```

```
{
    infile.read((char*)Buff, sizeof(char));
    buffvector.push_back(*Buff);
    MvCode_arr.push_back(code_table[*Buff].index);
    temple_index = code_table[*Buff].index;
    for (i = 0; i < 256; i++)
    {
        if (code_table[i].index < temple_index) code_table[i].index++;
    }
    code_table[*Buff].index = 0;
}
for (j = 0; j < MvCode_arr.size(); j++)
{
        buffvector.push_back(*Buff);

        if (sum_weight == 0)

        {

            buff_pre = *Buff;

            count++;

        }

        else

        {

            *Buff = MvCode_arr[j];

            if (*Buff == buff_pre&&count < 127) count++; //repeat and Not the MSR

symbol

            else                                        //MSR symbol or count =1

            {
```

```
if (count == 1 && !(buff_pre & 0x80)) //not MSR symbol
{
    MLCode_arr.push_back(buff_pre);
    count = 1;
    buff_pre = *Buff;
}
else
{
    count += 128;    // MSR symbol
    MLCode_arr.push_back(count);
    MLCode_arr.push_back(buff_pre);
    count = 1;
    buff_pre = *Buff;
```

```
                }
            }
        }
        sam_weight[*Buff]++;
        sum_weight++;
    }
}
else
{
    while (infile.peek() != EOF)
    {
        infile.read((char*)Buff, sizeof(char));
        buffvector.push_back(*Buff);
```

```
if (sum_weight == 0)
{
    buff_pre = *Buff;
    count++;
}
else
{
    if (*Buff == buff_pre&&count <127) count++;
    else
    {
        if (count == 1 && !(buff_pre & 0x80))
        {
            MLCode_arr.push_back(buff_pre);
    }

            count = 1;

            buff_pre = *Buff;

        }

        else

        {

            count += 128;

            MLCode_arr.push_back(count);

            MLCode_arr.push_back(buff_pre);

            count = 1;

            buff_pre = *Buff;

        }

    }

}
```

```
            }

        sam_weight[*Buff]++;

        sum_weight++;

    }

}

if (count == 1)

{

    MLCode_arr.push_back(buff_pre);

}

else

{

    MLCode_arr.push_back(count+=128);

    MLCode_arr.push_back(buff_pre);

}
```

```
for (i = 0; i < MLCode_arr.size(); i++)

{

    OutputBits(bit_file, MLCode_arr[i], 8);

    countnum++;

}

infile.close();

CloseOutputBitFile(bit_file);

delete Buff;

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

ifstream deofile(outFILE_name, ios::binary);

if (!deofile)

{

    cerr << "open error!" << endl;
    abort();

}

while (deofile.peek() != EOF)

{

    deofile.read((char*)Buff, sizeof(char));

    if (*Buff & 0x80)

    {

        count = *Buff - 128;

        deofile.read((char*)Buff, sizeof(char));

        code = *Buff;

        for (i = 0; i < count; i++)

        {

            decovector.push_back(code);
        }
    }
    else
    {
        code = *Buff;
        decovector.push_back(code);
    }
}
/*Generate the outfile name for decode*/
strWebsiteSel = _T("DecoMODRL_com_");
strWebsiteSel += FILE_name;
str = CT2A(strWebsiteSel.GetBuffer());
const char * deFILE_name = str.c_str();
```

```
bit_file = OpenOutputBitFile(deFILE_name);

for (i = 0; i < decovector.size(); i++)

{

    OutputBits(bit_file, decovector[i], 8);

}

deofile.close();

CloseOutputBitFile(bit_file);

delete Buff;

output_size = countnum ;

s.Format(_T("%lf"), output_size);

strWebsiteSel = s;

SetDlgItemText(IDC_EDIT7, strWebsiteSel);

s.Format(_T("%lf"), (output_size / input_size) * 100);
```

```
    strWebsiteSel = s;

    SetDlgItemText(IDC_EDIT8, strWebsiteSel);

    MLCode_arr.clear();

    MvCode_arr.clear();

    buffvector.clear();

    decovector.clear();

}

/*Compress the files with AD_Huffan algorism using Move_to front pre_processing*/

void CMFCApplication2Dlg::OnBnClickedButton2()

{

    // TODO: Add your control notification handler code here

    int i;

    int j;
int k;

HT_NODE node_root[512];

int *Buff;

int buff_pre = INF;

double sam_weight[256] = { 0 };

double sum_weight = 0;

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

int temple_index;

Sample_Code scode;

CString s;

CString strWebsiteSel;

HT_NODE *address;

countnum = 0;
```

```
NUM = -1;

order = 512;

buffvector.clear();

for (i = 0; i < 512; i++)

    Tree_node[i] = new HT_NODE;

NY_NODE = new HT_NODE(NULL, NULL, NULL, 0, 0, 512);

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

ifstream infile(FILE_name, ios::binary);

/*Generate the outfile name*/

strWebsiteSel += _T("MT_AT_com_");

strWebsiteSel += FILE_name;

string str;

str = CT2A(strWebsiteSel.GetBuffer());
```

```
}

const char * outFILE_name = str.c_str();

bit_file = OpenOutputBitFile(outFILE_name);

for (i = 0; i < 256; i++)

{

    code_table[i].code = i;

    code_table[i].index = i;

}

/*Move to front process*/

if (!infile)

{

    cerr << "open error!" << endl;

    abort();

}
```

```
while (infile.peek() != EOF)

{

    infile.read((char*)Buff, sizeof(char));

    buffvector.push_back(*Buff);

    MvCode_arr.push_back(code_table[*Buff].index);

    temple_index = code_table[*Buff].index;

    for (i = 0; i < 256; i++)

    {

        if (code_table[i].index < temple_index) code_table[i].index++;

    }

    code_table[*Buff].index = 0;

}

infile.close();
```

```
/*Using new symbol as input to do adp_huffman compress*/

for (j = 0; j < MvCode_arr.size(); j++)

{

    *Buff = MvCode_arr[j];

    if (!sam_weight[*Buff])

    {

        Add_Root(*Buff);

    }

    else

    {

        for (i = 1; i <= NUM; i++)

        {

            if (Tree_node[i]->name == *Buff)
            {
                address = Tree_node[i];
                AD_generate_code(Tree_node[i], *Buff, scode);
                for (k = CODE[(*Buff)].code.size() - 1; k >= 0; k--)
                {
                    OutputBit(bit_file, CODE[(*Buff)].code[k]);
                    countnum++;
                }
            }
        }
        Update_tree(address);
    }
    sam_weight[*Buff]++;
```

```
    sum_weight++;

}

delete Buff;

CloseOutputBitFile(bit_file);

node_arr.clear();

/*Decode process*/

for (i = 0; i < 256; i++)

{

    code_table[i].code = i;

    code_table[i].index = i;

}

int tem = 0;

for (j = 0; j < MvCode_arr.size(); j++)
```

```
{
    for (i = 0; i < 256; i++)
    {
        if (MvCode_arr[j] == code_table[i].index)
        {
            decovector.push_back(i);
            temple_index = code_table[i].index;
            tem = i;
        }
    }
    for (i = 0; i < 256; i++)
    {
        if (code_table[i].index < temple_index) code_table[i].index++;
    }
    code_table[tem].index = 0;
}
output_size = countnum / 8;
strWebsiteSel = _T("DecoMTF_com_");
strWebsiteSel += FILE_name;
str = CT2A(strWebsiteSel.GetBuffer());
const char * deFILE_name = str.c_str();
bit_file = OpenOutputBitFile(deFILE_name);
for (i = 0; i < decovector.size(); i++)
{
    OutputBits(bit_file, decovector[i], 8);
}
```

```
    CloseOutputBitFile(bit_file);

    s.Format(_T("%lf"), output_size);

    strWebsiteSel = s;

    SetDlgItemText(IDC_EDIT7, strWebsiteSel);

    s.Format(_T("%lf"), (output_size / input_size) * 100);

    strWebsiteSel = s;

    SetDlgItemText(IDC_EDIT8, strWebsiteSel);

    decovector.clear();

    MvCode_arr.clear();

}

/*Compress the files with Modified Run_length algorism using zigzag pre_processing*/

void CMFCApplication2Dlg::OnBnClickedButton15()

{
```

```
int i;

int j;

int N;

int *Buff;

int buff_pre = INF;

double sam_weight[256] = { 0 };

double sum_weight = 0;

unsigned long code = 0;

double count = 0;

//int im_data[512][512] = { 0 };

string str;

CString s;

CString strshowonboard;
```

```
}

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

ifstream infile(FILE_name, ios::binary);

strshowonboard += _T("zizMORunl_com_");

strshowonboard += FILE_name;

str = CT2A(strshowonboard.GetBuffer());

const char * outFILE_name = str.c_str();

bit_file = OpenOutputBitFile(outFILE_name);

countnum = 0;

if (!infile)

{

    cerr << "open error!" << endl;

    abort();

}
```

```
while (infile.peek() != EOF)

{

    infile.read((char*)Buff, sizeof(char));

    buffvector.push_back(*Buff);

}

/*Zigzag process*/

for (N = 0; N < 512; N++)

{

    for (i = 0; i <= N; i++)

    {

        SECbuffvector.push_back(buffvector[i * 512 + (N - i)]);

    }

    N++;
```

```
        for (i = N; i >= 0; i--)

        {

            SECbuffvector.push_back(buffvector[i * 512 + (N - i)]);

        }

    }

    for (N = 512; N <1024; N++)

    {

        for (i = N - 511; i <= 511; i++)

        {

            SECbuffvector.push_back(buffvector[i * 512 + (N - i)]);

        }

        N++;

        for (i = 511; i >= N - 511; i--)
        {
            SECbuffvector.push_back(buffvector[i * 512 + (N - i)]);
        }
    }
/*------------------------------------------------------------*/
    for (i = 0; i < 512; i++)
    {
        for (j = 0; j < 512; j++)
        {
            *Buff = SECbuffvector[i * 512 + j]; /*new input */
            if (sum_weight == 0)
            {
                buff_pre = *Buff;
    count++;
}
else
{
    if (*Buff == buff_pre&&count <= 127) count++;
    else
    {
        if (count == 1 && !(buff_pre & 0x80))
        {
            MLCode_arr.push_back(buff_pre);
            count = 1;
            buff_pre = *Buff;
        }
```

```
            else

            {

                count += 128;

                MLCode_arr.push_back(count);

                MLCode_arr.push_back(buff_pre);

                count = 1;

                buff_pre = *Buff;

            }

        }

    }

}

sam_weight[*Buff]++;

sum_weight++;
```

```
}
if (count == 1)
{
    MLCode_arr.push_back(buff_pre);
}
else
{
    MLCode_arr.push_back(count);
    MLCode_arr.push_back(buff_pre);
}
for (i = 0; i < MLCode_arr.size(); i++)
{
    OutputBits(bit_file, MLCode_arr[i], 8);
    countnum++;

}

infile.close();

CloseOutputBitFile(bit_file);

delete Buff;

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

ifstream deofile(outFILE_name, ios::binary);

if (!deofile)

{

    cerr << "open error!" << endl;

    abort();

}

while (deofile.peek() != EOF)
```

```
{
    deofile.read((char*)Buff, sizeof(char));
    if (*Buff & 0x80)
    {
        count = *Buff - 128;
        deofile.read((char*)Buff, sizeof(char));
        code = *Buff;
        for (i = 0; i < count; i++)
        {
            decovector.push_back(code);
        }
    }
    else
    {
        code = *Buff;
        decovector.push_back(code);
    }
}
strshowonboard = _T("DecoMODRL_com_");
strshowonboard += FILE_name;
str = CT2A(strshowonboard.GetBuffer());
const char * deFILE_name = str.c_str();
bit_file = OpenOutputBitFile(deFILE_name);
for (i = 0; i < decovector.size(); i++)
{
    OutputBits(bit_file, decovector[i], 8);
```

```
}

deofile.close();

CloseOutputBitFile(bit_file);

delete Buff;

output_size = countnum;

s.Format(_T("%lf"), output_size);

strshowonboard = s;

SetDlgItemText(IDC_EDIT7, strshowonboard);

s.Format(_T("%lf"), (output_size / input_size) * 100);

strshowonboard = s;

SetDlgItemText(IDC_EDIT8, strshowonboard);

MLCode_arr.clear();

buffvector.clear();
```

```
    decovector.clear();
}
/*Using the MTF*/
void CMFCApplication2Dlg::OnBnClickedRadio7()
{
    // TODO: Add your control notification handler code here
    flag_MTF = 1;
}
/*Not Using the MTF*/
void CMFCApplication2Dlg::OnBnClickedRadio8()
{
    // TODO: Add your control notification handler code here
    flag_MTF = 0;
```

```
}
```