装 订 线

综合题四 二维码生成程序的实现

实验报告

计2 1752528 朱世轩

完成日期 2018 .5. 25

二维码生成程序的实现

1. 题目及基本要求

在cmd窗口下实现二维码的生成程序

题目要求：

1.从键盘输入不超过100字节的字符串作为原始数据，生成二维码；

2.输入内容要求支持中英文及标点符号；

3.在cmd界面下实现二维码的打印

2.整体设计思路

整体设计思路为：

- 取得输入的字符串，选择纠错级别，将字符串转化为utf8编码，用二进制模式（即8bit模式）对其进行处理，即将所有数据模式都作为二进制处理；
- 根据输入字符串的长度和选择的纠错级别计算出版本号等必须信息（实现版本的动态选择）
- 将原串转换为以字符01方式表示的二进制字符串，加入模式编码、补位后得到信息码串；
- 再将该01字符形式表示的码串再转化为十进制，存进字符数组中。再使用现有的通过多项式计算生成纠错码的函数，得到纠错码串；
- 对得到的纠错码串和原数据码串按相应版本和纠错级别按一定规则进行分块再组装，存入总数据数组中；
- 使用二维数组存放对应图形打印的信息；先将定位信息等填充进矩阵，再按填充规则将数据信息填充进矩阵；
- 进行掩模（评分 选择 但因为这一部分我没有完全弄懂，只选择了一种掩模图形直接掩模）
- 将掩模后的信息填入矩阵，再写入版本信息，打印二维码

实现设计思路

- 在实现时，我借鉴了网上比较完善的一个生成二维码程序，在头文件中将生成二维码所需信息定义为一个结构体中，便于管理；并将大部分函数写成结构体成员函数，方便对数据的直接操作；

- 在实现版本动态变化时，我选择了采用打表的形式预置好二维码各个版本和纠错版本对应的各种信息（其实是现成的表）以及纠错块的数据结构等，存放在结构体数组中，要使用时按相应序号调用即可；

如下图所示

装 订 线

结构体中的二维码信息

```
struct QREncode                          //定义结构体存放二维码相关信息
{
public:
    int m_nLevel;       // 纠错等级
    int m_nVersion;     // 版本
    int m_nMaskingNo;   // 纠错等级

    int m_nSymbleSize;                                      //图形大小
    BYTE m_byModuleData[MAX_MODULESIZE][MAX_MODULESIZE]; // 最终需要填充的矩阵

    int m_ncDataCodeWordBit; //输入数据长度
    //以下是二进制串相关数组
    char m_byWordDatapre[MAX_DATACODEWORD * 8];    //二进制01  数据码字转换后的数组
    char m_byWordData[MAX_DATACODEWORD * 8];    //二进制01  数据码字加入模式码 计数码信息后的数组
    char m_byRSData[MAX_DATACODEWORD * 8];    //二进制01  纠错码字转换后的数组
    char m_byAllData[MAX_ALLCODEWORD * 8];     //二进制01  所有数据（数据码字+纠错码）转换后数组
    char m_ModeCode[MODECODE];               //模式指示符
    char m_CountCode[COUNTCODE];               //字符计数符



    //以下是原串相关数组
    char m_byDatepre[MAX_DATACODEWORD];        //原始转utf8后的数据串
    BYTE m_byDataCodeWord[MAX_DATACODEWORD]; //数据码字数组（原串）
    int m_ncDataBlock;                      //数据块数

    //以下是存放相关信息的变量
    int m_ncAllCodeWord; //所有编码数据长度
    BYTE m_byAllCodeWord[MAX_ALLCODEWORD]; //所有数据数组
    BYTE m_byRSWork[MAX_CODEBLOCK]; // RS纠错码数组
    int ncDataCodeWord;//数据分块长度

    //获取编码数据
```

结构体中的操作函数

```
    //获取编码数据
    /*************************************************************************************************/
    int GetVersion(int nLevel, const char* lpsSource, int ncLength);//通过输入字符长度和选择的纠错等级
    void GetCountCode(const char*data);        //获取字符计数符
    void GetEncodeInfo();                     //整合编码信息
    void InsertBit();                        //补位
    void prt_binary();                //将二进制串转成十进制串
    //void prt_dec();                   //将十进制串转成二进制串
    //void GetRSCodeWord(LPBYTE lpbyRSWork, int ncDataCodeWord, int ncRSCodeWord);//纠错码
    void BlockstoCode();                              //将数据和纠错码分块并存放至m_byAllCodeWord数组中
    /*************************************************************************************************/

    //填充矩阵
    /*************************************************************************************************/
    void FillMartrixCode();//编码信息
    void FillMartrixFunction();//功能模块 将定位信息和数据信息等填入矩阵 （调用下面的函数）
    void SetFinderPattern(int x, int y);//填充寻像图形
    void SetAlignmentPattern(int x, int y);//填充校正图形
    void SetVersionPattern();//版本信息
    /*************************************************************************************************/

    //掩模
    /*************************************************************************************************/
    void Masking();//设置掩模图形
    void SetFormatInfoPattern();
    void FilltheMatrix();  //将最终结果写入矩阵
    /*************************************************************************************************/

    //打印二维码
    /*************************************************************************************************/
    int printQR(); //打印二维码函数
    /*************************************************************************************************/
;
```

部分预置数据表

```
//QRcode版本相关信息
typedef struct tagQR_VERSIONINFO
{
    int nVersionNo;     // 版本号1-40
    int ncAllCodeWord; // 总码字数
    int ncDataCodeWord[4];  // 数据码字（总码字-RS码字）

    int ncAlignPoint;   // 校正点数
    int nAlignPoint[6]; // 校正图形中心坐标

    RS_BLOCKINFO RS_BlockInfo1[4]; // 纠错块部分信息
    RS_BLOCKINFO RS_BlockInfo2[4]; // 纠错块部分信息

} QR_VERSIONINFO;


/*二维码各个版本与各种信息量的关系*/          //匹配版本号的时候要用到
const int DataNum[41][4] = { ... }
//不同版本和纠错级别下二维码相关信息
const QR_VERSIONINFO QR_VersonInfo[] = { { 0 }, // Ver.0
{ 1, // Ver.1
26,    19,    16,    13,     9,
0,     0,     0,     0,     0,     0,     0,
1,    26,    19,
1     26     16
```

装

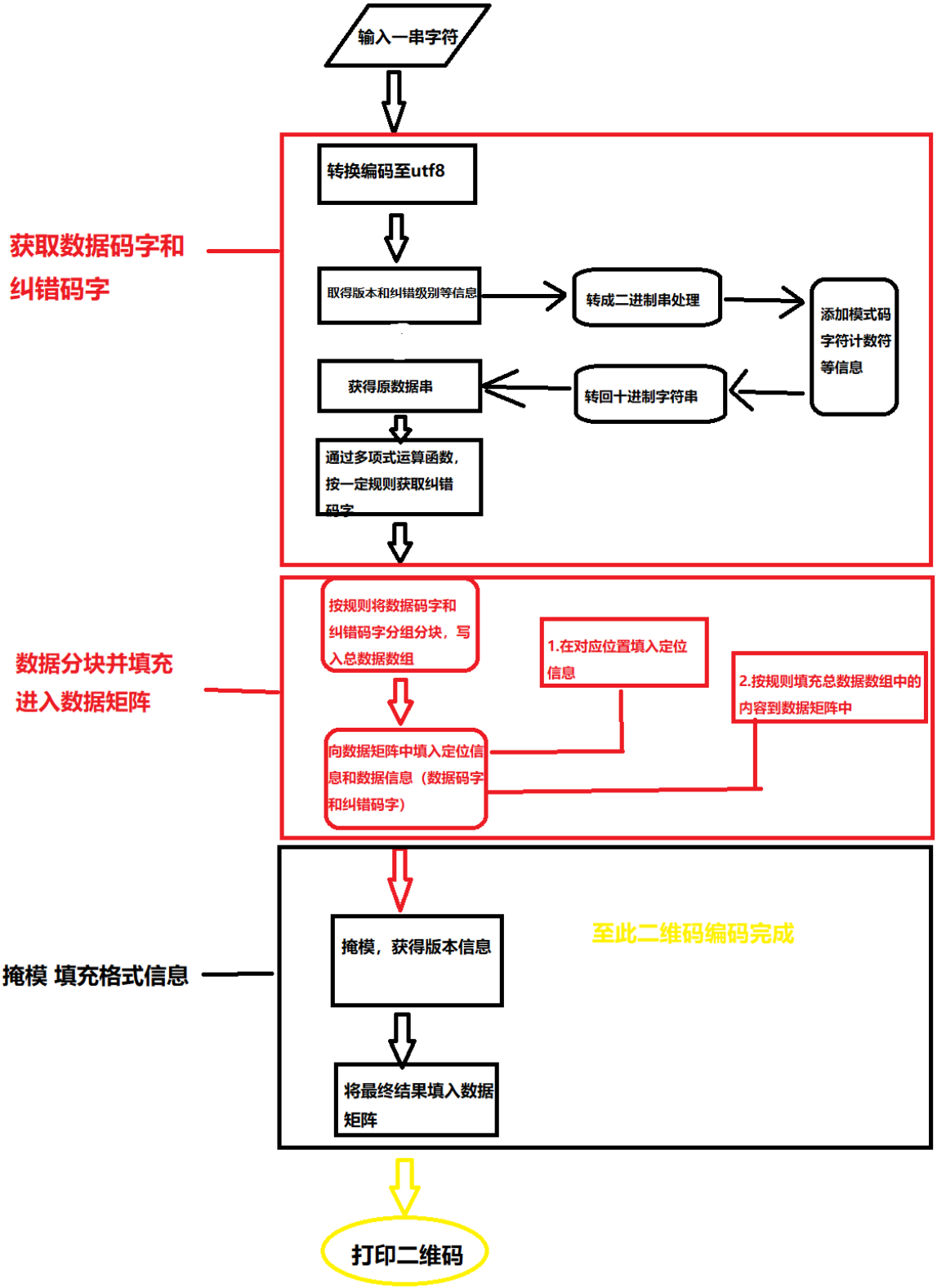
订

线

3. 主要功能的实现

1. 二维码的编码部分我使用了utf8编码方式和二进制数据模式的“万能方法”，来处理，实际效果也比较理想；
2. 对于实现二维码版本的动态适应所需要的二维码各版本的各种数据，我使用了在头文件中预置数据表的方法；例如，要取得在某版本Version和某纠错级别Level所对应的数据码字的容量，
   只需要按如下方式取得结构体数组QR_VersonInfo中的元素即可得到需要的数据
   QR_VersonInfo[Version].ncDataCodeWord[Level]
3. 生成纠错码和部分填充格式信息函数我使用了网上现成的函数
4. 掩模部分我只使用了000一种掩模图形，没有对八种掩模模式进行评分选择，最终生成的二维码在大多数情况下识别正常
5. 实现过程的流程图大致如下

装 订 线


输入一串字符
转换编码至utf8
获取数据码字和纠错码字
取得版本和纠错级别等信息
转成二进制串处理
添加模式码字符计数符等信息
获得原数据串
转回十进制字符串
通过多项式运算函数，按一定规则获取纠错码字
数据分块并填充进入数据矩阵
按规则将数据码字和纠错码字分组分块，写入总数据数组
1.在对应位置填入定位信息
2.按规则填充总数据数组中的内容到数据矩阵中
向数据矩阵中填入定位信息和数据信息（数据码字和纠错码字）
掩模 填充格式信息
掩模，获得版本信息
至此二维码编码完成
将最终结果填入数据矩阵
打印二维码

```
/***第一部分 获取数据编码***/
char*UTF8input = G2U(input);
inputLength = strlen(UTF8input);
QRdata.m_nVersion = QRdata.GetVersion(level, UTF8input, inputLength); //给结构体中版本号赋值
QRdata.m_nLevel = level;                                              //给结构体中纠错级别赋值
QRdata.m_ncDataCodeWordBit = inputLength;                             //给结构体中输入数据长度赋值
strcpy(QRdata.m_byDatepre, UTF8input);                                //给结构体中待处理字符串赋值
QRdata.GetCountCode(UTF8input);                                       //取得编码信息
StrToBinary(UTF8input, QRdata.m_byWordDatapre);                       //获得最初的二进制数据串
QRdata.GetEncodeInfo();                                               //加入编码信息 处理后的数据都保存在m_byWordData
QRdata.InsertBit();                                                   //补位          处理后的数据都保存在m_byWordData中

/***第二部分 获取纠错（RS）编码***/
QRdata.prt_binary();                                                  //将二进制数据串转成十进制
for (int i = 0; i < QR_VersonInfo[QRdata.m_nVersion].ncDataCodeWord[QRdata.m_nLevel]; i++){ ... }
cout << endl;
//调用多项式计算函数 生成纠错码字赋值给data.m_byRSWork
strcpy((char*)QRdata.m_byRSWork, (char*)(RS_Code(QRdata.m_ncAllCodeWord, QR_VersonInfo[QRdata.m_nVersion].ncDataCodeWord[QRdata.m_nLevel], QRdata.m_byDataCodeWord)));
cout << "纠错码：";                                                   //output test 按十进制打印出纠错码
for (int i = 0; i < QRdata.m_ncAllCodeWord - QR_VersonInfo[QRdata.m_nVersion].ncDataCodeWord[QRdata.m_nLevel]; i++){ ... }
cout << endl;
QRdata.BlockstoCode();                                                //对数据与纠错码进行分块
cout << "所有数据数组";                                               //output test   按十进制打印出所有数据数组
for (int i = 0; i < QRdata.m_ncAllCodeWord; i++){ ... }
cout << endl;

/***第三部分 填充 掩模***/
  //prt((char*)data.m_byAllCodeWord, data.m_byAllData);               //将所有数据的十进制码字串转为二进制串
QRdata.m_nSymbleSize = QRdata.m_nVersion * 4 + 17;                    // 通过版本号计算图形边长
//cout << QRdata.m_nSymbleSize;                                       //output test
QRdata.FillMartrixFunction();                                         //将定位信息和数据等填充进入矩阵
QRdata.FillMartrixCode();
//开始掩模
QRdata.Masking();                //使用000掩模图形进行掩模
QRdata.SetFormatInfoPattern(); // 设置格式信息
QRdata.FilltheMatrix();          //将最终结果写入矩阵

/***第四部分 打印二维码***/
QRdata.printQR();
```

主函数中反映的大致流程

4. 调试过程碰到的问题及解决方法

问题一：在开始对转换完成的utf8串进行加入信息、补位等处理时，尝试直接添加进原字符数组，

后发现，模式码是四位而其他数据是8位，难以操作；

解决方法：先将utf8数据串的每一个元素都转换为01形式的8位二进制表示，存入另一个字符数组中，进行加入信息补位等操作完成后，再转换回十进制形式

问题二：填充信息进入数据矩阵时，若一开始就以01形式填充，后面填充数据信息判断是否需要填充时会造成不便，掩模也会增大难度；

解决方法：填充信息时使用不同值进行区分，将定位信息等填充过的地方先用字符0和空格等标记，填充数据信息时遇到这些标记不进行填充；

问题三：最终打印后扫描识别不成功；

解决方法：逐步打印每一步编码，先确认到获得原数据串均是正确的，再打印出纠错码，和别人的正确结果比对，确认无误后，再逐个排查填充信息，掩模等的问题；最终发现是填充时顺序出错。

5.心得体会与经验教训

5.1．心得体会

心得体会：通过这次作业我体会到了程序框架搭建的重要性。前几次大作业例如消灭星星和俄罗斯方块，我没有构思清楚就急急忙忙开始实现，导致有很多问题没有考虑到，在实现过程中发现这些问题时才开始反复修改添加，吃了很多苦头；前一次工具函数集作业就更是如此，没有想写好需要的工具函数，就匆忙开始写，导致了很多重复的工作和问题；在这次实现二维码生成程序的过程中，一方面因为事先学习了大量资料，对二维码生成过程有一个完整的认识，也就了解了程序完成的大致框架；另一方面参考了网上一些比较完善的程序，预先在头文件中大致预备好了需要的变量，数组，需要用的数据表等，然后在主函数中搭好顺序流程的框架，再一个一个函数补充，整个过程就比较清晰了

经验教训：开始在实现中我使用的一些现成的函数，找到后没有细看就加入了源文件并直接进行调用，然后继续之后的实现，最终测试的时候才发现有问题，又回过头来检查那个函数，并进行修改，浪费了大量的时间；本来应该把那个函数具体实现看懂再使用的，想要省功夫的想法本来就是不切实际的

5.2．总结

这几次大作业，前几次合成十和消灭星星，我认为我考虑了前后小题的关系，并尽量让后面小题使用前面的代码。但我认为我这种考虑是出于减小工作量的目的，并没有考虑到重复代码或者说公共函数的通用性并进行有效利用，这一点上我做的不好，也导致了后面工具及函数作业没有做到让前两个大作业读取配置文件；在二维码大作业中我体会到了先搭好框架再填充函数对于程序实现的好处，我以后要养成这种习惯。

附件：源程序

部分工具函数

```
/*工具函数*/
/*********************************************************************/
//将UTF8串转化为二进制串
void StrToBinary(char c[], char binary[])
{
    int count = 0;
    for (int i = 0; c[i]; i++)
    {
        for (int j = 7; j >= 0; j--)
        {
            cout << ((c[i] & 1 << j) != 0);
            binary[count] = ((c[i] & 1 << j) != 0) + '0';
            count++;
```

```
        }
        cout << " ";
    }
    binary[count] = 0;
}

//将二进制串转成十进制码字
void QREncode::prt_binary()
{
    unsigned char *code = new unsigned char[QR_VersonInfo[m_nVersion].ncAllCodeWord];
    int i, j;
    for (i = 0; i < QR_VersonInfo[m_nVersion].ncAllCodeWord; i++)
    {
        code[i] = 0;
        for (j = 8 * i; j < 8 * i + 8; j++)
            code[i] = code[i] + (m_byWordData[j] - '0')*(1 << (7 - j % 8));
    }
    for (i = 0; i < QR_VersonInfo[m_nVersion].ncDataCodeWord[m_nLevel]; i++)
    {
        m_byDataCodeWord[i] = code[i];
    }
    //加尾零
    m_byDataCodeWord[QR_VersonInfo[m_nVersion].ncAllCodeWord] = '\0';
    delete code;
}

//将十进制串转成二进制码字
/*void QREncode::prt_dec()
{
    int count = 0;
    for (int i = 0;i<nc; i++)
    {
        for (int j = 7; j >= 0; j--)
        {
            cout << ((c[i] & 1 << j) != 0);
            binary[count] = ((c[i] & 1 << j) != 0) + '0';
            count++;
        }
        cout << " ";
    }
    binary[count] = 0;
}*/

/*取得编码信息 计算纠错码*/
/*****************************************************************/
```

```
//取得字符计数码
void QREncode::GetCountCode(const char*data)
{
    int countdata = strlen(data);
    cout << countdata << endl;     //test
    for (int i = 7; i >= 0; i--)
    {
        m_CountCode[7 - i] = ((countdata >> i) & 1) + '0';
    }
    m_CountCode[8] = 0;
}

//整合编码信息
void QREncode::GetEncodeInfo()
{
    int count = 0;
    int datalength = strlen((char*)(m_byWordData));
    m_ModeCode[0] = '0';
    m_ModeCode[1] = '1';
    m_ModeCode[2] = '0';
    m_ModeCode[3] = '0';
    for (int i = 0; i < 4; i++)
    {
        m_byWordData[count] = m_ModeCode[i];
        count++;
    }
    for (int i = 0; i < 8; i++)
    {
        m_byWordData[count] = m_CountCode[i];
        count++;
    }
    for (int i = 0; i < datalength; i++)
    {
        m_byWordData[count] = m_byWordDatapre[i];
        count++;
    }
}

//对编码信息进行补位  待处理数据在m_byWordData中
void QREncode::InsertBit()
{
    int count = 0;
    int count_insert = 0;
    int flag = 0;
    int datalength = strlen(m_byWordData);                    //取得未补位时的长度
```

```
    cout << datalength << endl;                                    //test
    int Maxlength = QR_VersonInfo[m_nVersion].ncDataCodeWord[m_nLevel];      //取得应该补到的长
度
    cout << Maxlength << endl;
    m_ncAllCodeWord = QR_VersonInfo[m_nVersion].ncAllCodeWord;
    char Bit[2][10] = {                                             //预置补位数组
        { "11101100" } ,
        { "00010001" },
    };
    for (int i = datalength; i < Maxlength * 8; i++)            //补0至被8整除
    {
        m_byWordData[i] = '0';
        count++;
        if ((i + 1) % 8 == 0)
            break;
    }
    datalength = datalength + count;                           //更新数据长度
    cout << datalength << endl;
    for (int i = datalength; i < Maxlength * 8; i = i + 8)   //交替补码串直到最大长度
    {
        for (int j = 0; j < 8; j++)
        {
            if (flag % 2 == 0)
                m_byWordData[i + j] = Bit[0][j];
            else
                m_byWordData[i + j] = Bit[1][j];

        }
        count_insert = count_insert + 8;
        flag++;
    }
    datalength = datalength + count_insert;                     //更新数据长度
    cout << datalength << endl;
    m_byWordData[datalength] = 0;
}

//通过转化后的字符串大小 获取对应版本号
int QREncode::GetVersion(int nLevel, const char* lpsSource, int ncLength)
{

    for (int i = 1; i < 41; i++)
    {
        if (DataNum[i][nLevel] > ncLength)
        {
            m_nVersion = i;
```

```
                break;
            }
        }
        return m_nVersion;
}

//对数据和纠错码进行分块填入总数据数组
void QREncode::BlockstoCode()
{
        //对数据码字分块 写入总数组
        int i, j;
        int count = 0;
        //两组分块中每组的块数
        int Block1 = QR_VersonInfo[m_nVersion].RS_BlockInfo1[m_nLevel].ncRSBlock;
        int Block2 = QR_VersonInfo[m_nVersion].RS_BlockInfo2[m_nLevel].ncRSBlock;
        int BlockSum = Block1 + Block2;
        int BlockNum = 0; // 处理块数总和

        int DataClass1 = QR_VersonInfo[m_nVersion].RS_BlockInfo1[m_nLevel].ncDataCodeWord;
        int DataClass2 = QR_VersonInfo[m_nVersion].RS_BlockInfo2[m_nLevel].ncDataCodeWord;

        for (i = 0; i < Block1; i++)
        {
            for (j = 0; j < DataClass1; j++)
            {
                m_byAllCodeWord[(BlockSum * j) + BlockNum] = m_byDataCodeWord[count];
                count++;
            }
            BlockNum++;
        }

        for (i = 0; i < Block2; i++)
        {
            for (j = 0; j < DataClass2; j++)
            {
                if (j < DataClass1)
                {
                    m_byAllCodeWord[(BlockSum * j) + BlockNum] = m_byDataCodeWord[count];
                    count++;
                }
                else
                {
                    m_byAllCodeWord[(BlockSum * DataClass1) + i] = m_byDataCodeWord[count];
                    count++;
                }
```

```
        }

        BlockNum++;
    }
    ////////////////////////////////////////////////////////////////////////
        //对纠错码字分块 写入总数组

    int RSClass1 = QR_VersonInfo[m_nVersion].RS_BlockInfo1[m_nLevel].ncAllCodeWord - DataClass1;
    int RSClass2 = QR_VersonInfo[m_nVersion].RS_BlockInfo2[m_nLevel].ncAllCodeWord - DataClass2;

    count = 0;
    BlockNum = 0;

    for (i = 0; i < Block1; i++)
    {
        for (j = 0; j < RSClass1; j++)
            m_byAllCodeWord[ncDataCodeWord + (BlockSum * j) + BlockNum] = m_byRSWork[j];

        count = count + DataClass1;
        BlockNum++;
    }

    for (i = 0; i < Block2; i++)
    {
        for (j = 0; j < RSClass2; j++)
            m_byAllCodeWord[ncDataCodeWord + (BlockSum * j) + BlockNum] = m_byRSWork[j];

        count = count + DataClass2;
        BlockNum++;
    }
}

/*向矩阵填充信息 掩模*/
/**********************************************************************/
//向矩阵内写入数据信息
void QREncode::FillMartrixCode()
{
    //填入功能信息后 矩阵内现在有信息部分为0 无信息部分为空格
    int x = m_nSymbleSize;          //定位坐标到矩阵右下角
    int y = m_nSymbleSize - 1;
    int x_flag = 1;                  //坐标状态标记 设置前进状态 初始都是前进状态
    int y_flag = 1;
    int length_of_code = m_ncAllCodeWord;  //要写入矩阵所有数据的长度
    for (int i = 0; i < length_of_code; i++)
    {
```

```
        for (int j = 0; j < 8; j++)
        {

            do   //如果是空格循环填充
            {
                x = x + x_flag;       //x坐标移动一格
                x_flag = x_flag* -1;//标记改变符号 下次反向
                if (x_flag < 0)
                {
                    y = y + y_flag;  //x坐标移动一格
                    if (y < 0 || y == m_nSymbleSize) //如果前进到边界 y坐标置0 到左边一列填充
否则继续前进
                    {
                        if (y < 0)
                            y = 0;
                        else
                            y = m_nSymbleSize - 1;
                        x = x - 2;
                        y_flag = y_flag * -1;//标记改变符号 下次反向

                        if (x == 6)
                            x--;
                    }
                }
            } while (m_byModuleData[x][y] & 0x20);//while
            unsigned char codeword = 0;
            if (m_byAllCodeWord[i] & (1 << (7 - j))) //获取每个码字8位的二进制码 1赋真0赋假
                codeword = 2;
            else
                codeword = 0;
            m_byModuleData[x][y] = codeword;
        }//for
    }//for 直到填完所有数据 跳出循环
}

//掩模 选择第一种掩模图形
void QREncode::Masking()
{
    for (int i = 0; i < m_nSymbleSize; i++)
    {
        for (int j = 0; j < m_nSymbleSize; j++)
        {
            if (!(m_byModuleData[j][i] == ' '))
            {
                int Maskflag;
```

```
                    Maskflag = ((i + j) % 2 == 0);      //满足反转条件置1 否则置0
                    //将编码区域数据与掩模异或
                    m_byModuleData[j][i] = (BYTE)((m_byModuleData[j][i] & 0xfe) |
(((m_byModuleData[j][i] & 0x02) > 1) ^ Maskflag));
                }
            }
        }
}
```

注意排版格式，对齐方式，可以采用两列排版，正文采用宋体，小五，行距为单倍行距

装

订

线