装

订

线

同济大学

高级语言程序设计

实验报告

（二维码）

学　号　：1453381

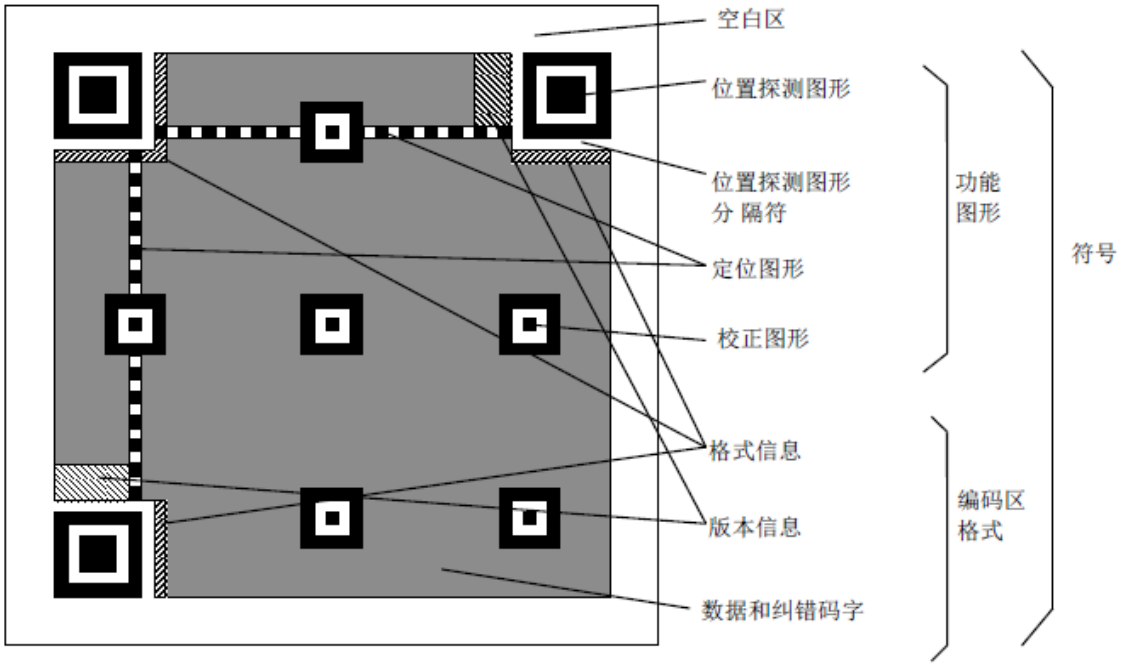
姓　名　：曾鸣

班　级　：计算机三班

完成日期　：2016年6月11日

1. 二维码原理

1.1 QR码结构构成



寻象图形：寻象图形包括三个相同的位置探测图形，可以明确地确定视场中符号的位置和方向。

分隔符：在每个位置探测图形和编码区域之间有宽度为1个模块的分隔符，它全部由浅色模块组成。

定位图形：水平和垂直定位图形分别为一个模块宽的一行和一列，作用是确定符号的密度和版本，提供决定模块坐标的基准位置。

校正图形：每个校正图形可看作是3个重叠的同心正方形，由5×5个的深色模块，3×3个的浅色模块以及位于中心的一个深色模块组成。

编码区域：包括表示数据码字、纠错码字、版本信息和格式信息的符号字符。

空白区：空白区为环绕在符号四周的4个模块宽的区域，其反射率应与浅色模块相同。

1.2 版本与图形关系

二维码一共有40个尺寸。即Version1-Version40,Version 1是21 x 21的矩阵，Version 2是25 x 25的矩阵，每增加一个version，就会增加4的尺寸，公式是：(V-1)*4 + 21（V是版本号）最高Version 40，(40-1)*4+21 = 177，最高是177 x 177 的正方形。

1.3 编码方法

数据分析：分析数据类型确定编码方式,可以进行混编,同时确定数据量,如果没有指定版本,那么按照当前纠正等级所匹配的最小版本来填充数据。

数据编码：根据所定义的规则，将数据字符转换为位流。在当需要进行模式转换时，在新的模式段开始前加入模式指示符进行模式转化，在数据序列后面加入终止符，将位流分为每8位一个码字，必要时加入版本要求的填充字符。

纠错编码：按照标准将数据流分块，通过里德所罗门纠错算法进行校验，生成纠错码字，加入相应数据码字序列的后面。

构造最终信息：按照标准将bit流放入数据块中,未填充完全则加入剩余位

布置模块：放入寻象图形,分隔符,校正图形,定位图形和码字(8bit)一起放入矩阵。

掩模：依次将8种掩模方式用于符号的编码区域，根据4个罚分标准评价结果取掩模。

格式和版本信息：生成格式和版本信息（Version>=7时），形成符号。

1.3.1 数据分析

分析所输入的数据流，确定要进行编码的字符的类型。QR 码支持扩充解释，可以对与缺省的字符集不同的数据进行编码。QR 码包括几种不同的模式，以便高效地将不同的字符子集转换为符号字符。必要时可以进行模式之间的转换更高效地将数据转换，以便为二进制串。

1.3.2 数据编码

Numeric mode 数字编码：从0到9。如果需要编码的数字的个数不是3的倍数，那么，最后剩下的1或2位数会被转成4或7bits，则其它的每3位数字会被编成 10，12，14bits。

Alphanumeric mode 字符编码：包括 0-9，大写的A到Z（没有小写），以及符号$ % * + - . / : 包括空格。这些字符会映射成一个字符索引表。编码的过程是把字符两两分组，然后转成下表的45进制，然后转成11bits的二进制，如果最后有一个落单的，那就转成6bits的二进制。而编码模式和字符的个数需要根据不同的Version尺寸编成9, 11或13个二进制。

Byte mode 字节编码：可以是0-255 的 ISO-8859-1 字符。有些二维码的扫描器可以自动检测是否是 UTF-8 的编码。

Kanji mode日文编码：双字节编码，也可以用于中文编码。日文和汉字的编码会减去一个值。如：在 0X8140 to 0X9FFC 中的字符会减去 8140，在 0XE040 到 0XEBBF 中的字符要减去 0XC140，然后把前两位拿出来乘以 0XC0，然后再加上后两位，最后转成 13bit 的编码

1.3.3 纠错编码

纠错一共有4个等级,恢复容量分别是L 7% M 15% Q 25 % H 30%

纠错码字可以纠正两种类型的错误，拒读错误（错误码字的位置已知,既无法扫描到或者无法译码,例如缺失了某一角）和替代错误（错误码字位置未知,既读取到了该模块,但是由于奇奇怪怪的问题导致黑读成了白,白读成了黑）。二维码的纠错码主要是通过Reed-Solomon error correction（里德-所罗门纠错算法）来实现的

1.3.4 构造最终信息

构造最终数据信息：在规格确定的条件下，将上面产生的序列按次序放如分块中，按规定把数据分块，然后对每一块进行计算，得出相应的纠错码字区块，把纠错码字区块 按顺序构成一个序列，添加到原先的数据码字序列后面。如：D1, D12, D23, D35, D2, D13, D24, D36, ... D11, D22, D33, D45, D34, D46, E1, E23,E45, E67, E2, E24, E46, E68，...构造矩阵：将探测图

形、分隔符、定位图形、校正图形和码字模块放入矩阵中。

1.3.5 布置模块

按照与使用的版本相对应的模块数构成空白的正方形矩阵。在寻象图形、分隔符、定位图形以及校正图形相应的位置，填入适当的深色浅色模块。格式信息和版本信息的模块位置暂时空置，在QR符号的编码区域中，符号字符以2个模块宽的纵列从符号的右下角开始布置，并自右向左，且交替地从下向上或从上向下安排。

1.3.6 掩模

将掩摸图形用于符号的编码区域，使得二维码图形中的深色和浅色（黑色和白色）区域能够比率最优的分布。掩模不用于功能图形用多个矩阵图形连续地对已知的编码区域的模块图形（格式信息和版本信息除外）进行XOR操作。XOR操作将模块图形依次放在每个掩模图形上，并将对应于掩模图形的深色模块的模块取反（浅色变成深色，或相反）。对每个结果图形的不合要求的部分记分，以评估这些结果。选择得分最低的图形。

1.3.7 格式和版本信息

版本信息为15位，5个数据位，10个用BCH(15,5)编码计算得到的纠错位。将10位纠错数据放到5位数据位之后，与掩模图形进行XOR运算；版本信息18位，6位数据位，通过用BCH(18,6)编码计算出12个纠错位，将12位纠错信息放在6位数据之后，不必对版本信息进行掩模，只有Version7-Version40包含版本信息。

2. 整体设计

2.1 界面设计

输入说明：理论上随意输入任意字符及长度，但是实际由于控制台字体的大小，会有一定的限制。输入任意字符（含中英文、数字、空格及符号等）和网址后按回车即可。

输出说明：根据输入的字符，产生相应二维码输出（二维码扫描工具能够识别）。

2.2 整体程序设计

从二维码的编码原理中可以知道，生成一个二维码并不是一两个步骤的事情，而是由数据分析、数据编码、纠错编码、构造最终信息、布置模块、掩模、填写格式和版本信息七大步骤实现，七个步骤缺一不可，并且任一步骤均不能有半点差错，稍有不慎形成错误就会造成错误累加，所以，对于整个程序的完成，最好分为不同模块功能，逐次校验，然后拼接调用，最后完成整个任务。

2.2.1 必要数据的存储

各个版本信息的存储：二维码共有40个不同版本，每个版本又有四个不同的纠错等级，而每

个版本和纠错格式都包含不同的信息，并且大多没有什么规律可循，所以可以定义一个全局常量结构体，用于存储版本的不同信息。定义如下：

```
struct QRVersion
{
    unsigned char Version;                  //版本
    unsigned char Level;                    //纠错等级
    unsigned char SideSize;                 //边界大小
    short DataCodeWordsNo;                  //数据码字数
    unsigned char RemainBit;                //剩余位
    short ECCodeWordsNo;                    //每块纠错码字数
    unsigned char Block1No;                 //包含块1数量
    unsigned char Block1DataCodeWordsNo;    //块1中数据码字数
    unsigned char Block2No;                 //包含块2数量
    unsigned char Block2DataCodeWordsNo;    //块2中数据码字数
};
```

校正图形坐标存储：用于提供每个版本的校正图形坐标位置的结构体。

```
struct AlignLocation
{
    unsigned char Version;          //版本
    unsigned char ncoord;           //校正图形坐标数字的个数
    unsigned char coord[7];         //存储校正图形坐标的数组，最多有七个校正数
};
```

用于纠错算法中的纠错码字数和其对应的生成多项式表可分别用一个一维数组和序号对应的二维数组存储。纠错码字表数据为纠错码字数，生成多项式表中按系数由高到低或由低到高存储。

2.2.2 模块功能及实现设计

首先，需要对输入的字符串进行分析，这个字符串包含各种信息，有不同的排列方法，编码描述中建议选取能够容纳编码字符串的最小版本，对于汉字字符外的字符所需最长比特串应该是按照8位编码模式所需的比特串最长，而汉字的比特串长度由汉字个数确定，另外将各个编码模式的计数串算作取做最大长度估计，最后加上每个编码模式串长度可以估计出数据串编码后所需最长位串，根据所需纠错等级，在版本信息表中找到合适的最小版本。

对于数据编码成的比特串，如果直接用整型数据或其数组存储，对其后的分析不易操作，所以，可以将数据编码后的0、1比特串依次存储在一个char型一维数组中，将数据转化完毕后，添加调整四个尾零，若还不足数据串长度，循环用11101100和00010001串补足，最后将形成的比特串八个一组转为unsigned char 存储到数组中，方便进行数据分块和进行后续的生成纠错码。

根据上一步骤完成的数据编码后的unsigned char 数组，根据版本信息进行数据分块，由纠错码字得到对应的生成多项式，对每一块应用里德所罗门纠错算法生成纠错码。由于不同的版本会有不同的纠错码字和数据分块，所以为方便统一完成，可以将数据码和纠错码均存储为一维unsigned char 型数组，然后由版本信息生成一个数组用于存储每个块的第一个数据的偏移数据，方便进行数据块的位串流生成。然后将数据块中交错将unsigned char 转化为 char 型位串，可以用‘0’表示浅色，‘1’表示深色，采用同样的方式将纠错块数据转化为位串加到数据串后。

便可进行模块的布置了。

接下来布置模块，布置模块会涉及将数据填入到最终的图形矩阵中，但是不同的版本大小不同，为方便进行函数间的参数传递和进行后续的数据填充，可以采取适当的浪费空间的做法，将图形矩阵设置为char matrix[177][177]以使用所用版本。布置模块先布置寻象图形、分隔符和定位图形，因为其位置相对较固定，不会有太多版本的不同要求，然后由相应的版本布置校正图形，其中需要避开寻象图形，不能有模块位置的重复，在上面三个模块布置时，为方便后续数据的位置填充，可以用1代表深色模块，2代表浅色模块，而数据位的填充可以用3代表深色模块，4代表浅色模块，0代表未填充的剩余位来区分。最后将数据块按照两列一组从矩阵右下角开始Z型填充，需跳过已填充信息，垂直定位列需要跳过。

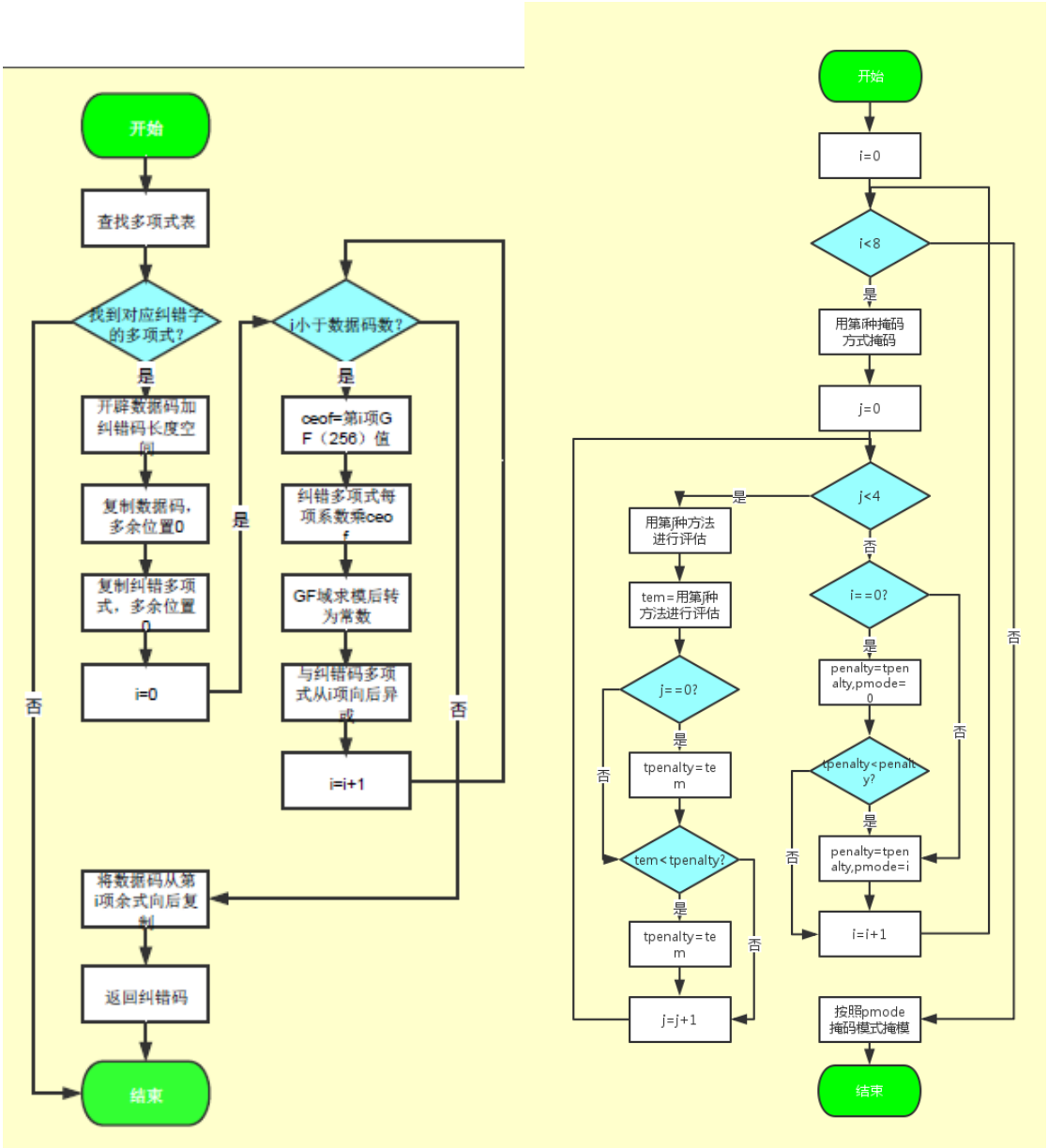
然后进行掩模操作，同样可以在定义一个 char mask[177][177]的数组进行掩模后的数据存储。掩模的评价需要包含寻象图形、分隔符和定位图形，同时如果有剩余位和保留位当做浅色模块处理，在进行每种方式掩模是，可以将矩阵中的浅色模块统一用0表示，深色模块统一用1表示，方便进行掩模方式的评价。

最后，在掩模后的矩阵中添加格式和版本信息，形成最终的矩阵。根据版本大小信息和矩阵中的深浅模块信息，调整控制台屏幕大小，设置前景背景色输出，形成最终的二维码。

3. 主要功能的实现

装 订 线

里德所罗门纠错　　选择掩模方式



4. 调试过程碰到的问题

4.1 版本和格式串错误

在刚刚写好二维码时，随便输入了几个字符测试，然后用扫码器可以正常识别，但是再多测试几次的时候，就会出现某些时候不能正常识别，细心分析思考，既然可以识别，不可能是数据编码、纠错码生成、数据模块的放置、其它信息的放置以及最后的输出出现问题，所以问题只可能是在格式和版本串生成各自的纠错信息时可能出现问题，而这一模块是直接看教程用算法自己生成的，所以很可能出错，于是再次看了下教程，发现可以用查表方式，这样准确简单，于是重

新用查表方式生成格式和版本信息，再次测试时问题已解决。

4.2 汉字不能识别

二维码写好了，但是对于字符、数字、字母可以正常识别，但是一旦输入汉字就会识别错误，明明是按照二维码说明文件一步一步规范来做的，应该不会出问题吧，即使有问题，在进行了单步调试看看了自己汉字编码的具体过程，也是严格按照说明文档的。百思不得其解，于是上网搜了一下资料，发现至少有三个以上网页都是和文档的说明一样，中国汉字模式指示符1101,，再加上计数位和汉字13位编码，还是没有解决问题。最后还是找了一份国家QRCode标准说明文档，发现前面的文档都有一个问题：汉字模式指示符没有错，编码没问题，但是指示符后少了一个字符集指示符0001（GB2312子集指示符）。加上这个后，一切问题解决。

5. 心得体会

5.1 文档资料很重要

通过上次LED模拟显示的作业和本次的二维码作业，我发现其实是否能够快速正确完成的关键并不在于编程能力，而很大程度取决于查找资料和分析资料的能力。LED耗时在查找一份GB18030编码的标准说明文档，二维码究其根本还是耗时在一份正确标准的说明文档。做LED时没有找一份国标文档之前找了不少的关于字库的偏移地址的计算算法，但拿来一用根本不能成功，如果没有找一份国家标准文档，根本不可能分析出字库的问题，也不能正确完成加分题要求。而这次二维码最先找了两份看似非常标准的文档，一份英文版一百多页和一份中文版六十余页，心想应该不会有问题，但是在汉字编码部分都有问题，导致了后面的汉字无法正常识别。无数网页都没有解决这个问题，如果不是国家标准文档，这个问题还是无法解决。另外在二维码编写的时候有一个英文网页将其编写方法说的很详尽透彻，包括里德所罗门算法，使得后面的编写较为容易，否则在纠错码算法部分可能会花很多时间还会没有结果。所以对于很多不知道的知识，查找文档应该尽量找国家标准规范文档，文档对于未知任务的完成很重要。

6. 附件：源程序

```
/*   97-b5-main.cpp   */
#include "97-b5.h"

int main()
{
	char data[2048];
	QRVersion Q;
	char m[177][177] = { 0 };
	char mask[177][177] = { 0 };
	int pmode;

	cout << "请输入文本或网址信息（按回车结束）：" << endl;
	gets_s(data);
```

```
	DataAnalysis(data, Q);

	unsigned char *DataCodeWords = new unsigned char[Q.DataCodeWordsNo];//记得释放
	DataEnCoding(Q, data, DataCodeWords);

	char *bitString = new char[(Q.DataCodeWordsNo + Q.ECCodeWordsNo*(Q.Block1No + Q.Block2No)) * 8
+ 1];//记得释放
	StructMes(Q, DataCodeWords, bitString);

	ModulePlacement(Q, m, bitString);

	Masking(Q, m, mask, pmode);

	SetFormatVersion(Q, mask, pmode);

	ScreenOutput(Q, mask);

	delete DataCodeWords;//释放空间
	delete bitString;

	return 0;
}

/*   97-b5-myself.cpp        */


#include "97-b5.h"

/*********************************
	数据分析+数据编码部分
*********************************/

/*  获得相应版本编码模式计数bit数  */
int GetCountIndiBit(int Version, int Mode)
{
	if (Version >= 1 && Version <= 9)
		Version = 0;
	else if (Version <= 26)
		Version = 1;
	else if (Version <= 40)
		Version = 2;

	return CountIndiBit[Version][Mode];
}

/*  将整数转为指定位数的‘0’‘1’位串  */
void ToBit(int I, char *bitChar, int NBit)
{
	int i;
	for (i = 0; i < NBit; i++)
		if (I & 1 << i)
			bitChar[NBit - i - 1] = '1';
		else
			bitChar[NBit - i - 1] = '0';
}

void BitToChar(unsigned char &result, char *BitChar)
{
	int i;
```

```
    for (i = 0; i < 8; i++)
        if (BitChar[i] == '1')
            result |= 0x80 >> i;
        else
            result &= ~(0x80 >> i);
}
void ByteMode(char *&data, char*& bitChar, int CountBit)
{
    int CByte;
    int i;
    for (CByte = 0; (unsigned char)data[CByte] > 0 && (unsigned char)data[CByte] < 0x7f; CByte++)
        ;

    if (CByte)
    {
        strcpy(bitChar, ModeIndi[Byte]);
        ToBit(CByte, bitChar + 4, CountBit);

        bitChar += (4 + CountBit);
        for (i = 0; i < CByte; i++, bitChar += 8)
            ToBit(data[i], bitChar, 8);

        data += CByte;
        *bitChar = '\0';
    }
}

void ChineseMode(char *&data, char *&bitChar, int CountBit)
{
    int CChinese;
    int i;
    unsigned char c1, c2;
    for (CChinese = 0; data[CChinese * 2] < 0; CChinese++)
        ;

    if (CChinese)
    {
        strcpy(bitChar, "11010001");
        ToBit(CChinese, bitChar + 8, CountBit);

        bitChar += (8 + CountBit);
        for (i = 0; i < CChinese; i++, bitChar += 13)
        {
            c1 = data[i * 2];
            c2 = data[i * 2 + 1];
            if (c1 < 0xB0)
                ToBit((c1 - 0xA1) * 0x60 + (c2 - 0xA1), bitChar, 13);
            else
                ToBit((c1 - 0xA6) * 0x60 + (c2 - 0xA1), bitChar, 13);
        }

        data += (CChinese * 2);
        *bitChar = '\0';
    }
}

void DataAnalysis(const char *data, QRVersion &Q)
{
    int CByteMode = 0;
    int Chinese = 0;
```

装 订 线

```
	int TotalDataBit = 0;

	int i;
	for (i = 0; data[i];)
		if (data[i] >0 && data[i] < 0x7f)
		{
			TotalDataBit += 8;
			if (i == 0)
				CByteMode++;
			else if (data[i - 1]<0)
				CByteMode++;
			i++;
		}
		else
		{
			TotalDataBit += 13;
			if (i == 0)
				Chinese++;
			else if (data[i - 1] >0&&data[i - 1] < 0x7f)
				Chinese++;
			i += 2;
		}
	TotalDataBit += (CByteMode * 4 + Chinese * 8);

	for (i = 1; i <= 40; i++)
		if (TotalDataBit + CByteMode*GetCountIndiBit(i, Byte) + Chinese*GetCountIndiBit(i, KanJi) <= 8 *
AllVersion[(i - 1) * 4 + VersionLevel].DataCodeWordsNo)
			break;

	Q = AllVersion[(i - 1) * 4 + VersionLevel];
}

void DataEnCoding(const QRVersion &Q, char *data, unsigned char *result)
{
	int TotalBit = Q.DataCodeWordsNo * 8;
	char *DataBit = new char[TotalBit + 1 + 4];//记得释放

	char *pData = data;
	char *pDataBit = DataBit;

	for (; *pData != '\0';)
	{
		ByteMode(pData, pDataBit, GetCountIndiBit(Q.Version, Byte));
		ChineseMode(pData, pDataBit, GetCountIndiBit(Q.Version, KanJi));
	}

	strcpy(pDataBit, "0000");
	DataBit[TotalBit] = '\0';

	int CurDataBit = strlen(DataBit);
	int RemainDataBit = TotalBit - CurDataBit;
	int Set0Bit = RemainDataBit % 8;

	int i;
	for (i = 0; i < Set0Bit; i++)
		strcat(pDataBit, "0");

	RemainDataBit -= Set0Bit;
	for (i = 0; i < RemainDataBit; i += 8)
		if (i / 8 % 2 == 0)
```

```
                strcat(pDataBit, "11101100");
            else
                strcat(pDataBit, "00010001");

        for (i = 0; i < Q.DataCodeWordsNo; i++)
            BitToChar(result[i], DataBit + i * 8);
}

/*********************************
        生成纠错码部分
*********************************/

/*    *********************************
函数功能：生成一块纠错码
Mes：数据码首地址
MesNum：一块数据码字节数
result：纠错码结果首地址
ErrCorCodeNum：纠错码字节数
*/
void GenerErrorCorCode(const unsigned char *Mes, int MesNum, unsigned char *result, int ErrCorCodeNum)
{
        int pGenPoly;
        for (pGenPoly = 0; pGenPoly < GenerTableSize&&pGenerPolyTable[pGenPoly] != ErrCorCodeNum;
pGenPoly++)
            ;
        if (pGenPoly >= GenerTableSize)
        {
            cout << "纠错码字数有误" << endl;
            return;
        }

        const int PolyNum = MesNum + ErrCorCodeNum;
        unsigned char *MesPoly = new unsigned char[PolyNum];//开辟内存
        unsigned char *GenPoly = new unsigned char[PolyNum];
        unsigned char *tem = new unsigned char[PolyNum];

        int i, j;
        for (i = 0; i < PolyNum; i++)
        {
            MesPoly[i] = (i < MesNum ? Mes[i] : '\0');
            GenPoly[i] = (i <= ErrCorCodeNum ? GenerPolyTable[pGenPoly][ErrCorCodeNum - i] : '\0');
        }

        unsigned char CurCeof;
        unsigned char t;
        for (i = 0; i < MesNum; i++)
        {
            CurCeof = AntiLog[MesPoly[i]];
            for (j = 0; j <= ErrCorCodeNum; j++)
            {
                tem[j] = (GenPoly[j] + CurCeof) % 255;
                tem[j] = Log[tem[j]];
                t = MesPoly[i + j] ^= tem[j];
            }
        }

        for (j = 0; i < PolyNum; i++)
            result[j++] = MesPoly[i];
```

```
	delete MesPoly;//释放内存
	delete GenPoly;
	delete tem;

	return;
}

/*	**************************************
函数功能：将数据分块产生纠错码且按规则排列
Q：版本信息
data：数据码
bitString：结果比特串指针
*/
void StructMes(const QRVersion &Q, unsigned char *data, char *bitString)
{
	int BlockNo = Q.Block1No + Q.Block2No;
	int *BlockAddr = new int[BlockNo];//记得释放
	unsigned char	*ErrorCode = new unsigned char[BlockNo*Q.ECCodeWordsNo];//记得释放

	int i, j, total;
	for (i = 0, j = 0; i < Q.Block1No; i++)//块类型1
	{
		BlockAddr[i] = j;
		GenerErrorCorCode(data + j, Q.Block1DataCodeWordsNo, ErrorCode + i*Q.ECCodeWordsNo,
Q.ECCodeWordsNo);
		j += Q.Block1DataCodeWordsNo;
	}
	for (; i < BlockNo; i++)//块类型2（如果有）
	{
		BlockAddr[i] = j;
		GenerErrorCorCode(data + j, Q.Block2DataCodeWordsNo, ErrorCode + i*Q.ECCodeWordsNo,
Q.ECCodeWordsNo);
		j += Q.Block2DataCodeWordsNo;
	}

	int maxBlock = Q.Block1DataCodeWordsNo>Q.Block2DataCodeWordsNo ? Q.Block1DataCodeWordsNo :
Q.Block2DataCodeWordsNo;
	for (i = 0, total = 0; i < maxBlock; i++)
		for (j = 0; j < BlockNo; j++)
			if (j < Q.Block1No&&i < Q.Block1DataCodeWordsNo)
				ToBit(data[BlockAddr[j] + i], bitString + 8 * total++, 8);
			else if (j >= Q.Block1No)
				ToBit(data[BlockAddr[j] + i], bitString + 8 * total++, 8);

	for (i = 0; i < Q.ECCodeWordsNo; i++)
		for (j = 0; j < BlockNo; j++)
			ToBit(ErrorCode[j*Q.ECCodeWordsNo + i], bitString + 8 * total++, 8);

	bitString[total * 8] = '\0';
	delete BlockAddr;//释放空间
	delete ErrorCode;//释放空间
}

/*********************************
	布置模块部分
*********************************/

void PlaceMatrix(char(*m)[177], char *ori, const int oriCol, const int oriRow, const int left, const int top)
```

```
{/* 在二维码矩阵中指定位置填入矩阵 */
    int i, j;
    for (i = 0; i < oriRow; i++)
        for (j = 0; j < oriCol; j++)
            m[top + i][left + j] = *(ori + i*oriCol + j);
}

void SetFinder(const QRVersion &Q, char(*m)[177])
{
    /* 填充finder pattern 1代表深色，2代表浅色 */
    char finder[7][7] = {
        { 1, 1, 1, 1, 1, 1, 1 },
        { 1, 2, 2, 2, 2, 2, 1 },
        { 1, 2, 1, 1, 1, 2, 1 },
        { 1, 2, 1, 1, 1, 2, 1 },
        { 1, 2, 1, 1, 1, 2, 1 },
        { 1, 2, 2, 2, 2, 2, 1 },
        { 1, 1, 1, 1, 1, 1, 1 } };
    const int left1 = 0, top1 = 0;
    const int left2 = Q.SideSize - 7, top2 = 0;
    const int left3 = 0, top3 = Q.SideSize - 7;
    PlaceMatrix(m, *finder, 7, 7, left1, top1);
    PlaceMatrix(m, *finder, 7, 7, left2, top2);
    PlaceMatrix(m, *finder, 7, 7, left3, top3);
}

void SetSeperator(const QRVersion &Q, char(*m)[177])
{
    /* 填充Seperator 位置 2代表浅色 */
    int i;
    for (i = 0; i < 8; i++)
    {
        m[7][i] = m[i][7] = 2;
        m[7][Q.SideSize - 8 + i] = m[i][Q.SideSize - 8] = 2;
        m[Q.SideSize - 8][i] = m[Q.SideSize - 8 + i][7] = 2;
    }
}

void SetAlignMent(const QRVersion &Q, char(*m)[177])
{
    /* 填充AlignMent 1代表深色，2代表浅色 */
    char align[5][5] = {
        { 1, 1, 1, 1, 1 },
        { 1, 2, 2, 2, 1 },
        { 1, 2, 1, 2, 1 },
        { 1, 2, 2, 2, 1 },
        { 1, 1, 1, 1, 1 } };

    const int row1 = 7, row2 = Q.SideSize - 8;
    const int col1 = 7, col2 = Q.SideSize - 8;
    const int ncoord = AL[Q.Version - 1].ncoord;
    int i, j;
    int left, top;
    for (i = 0; i < ncoord; i++)
        for (j = 0; j < ncoord; j++)
        {
            left = AL[Q.Version - 1].coord[i] - 2;
            top = AL[Q.Version - 1].coord[j] - 2;
            if (i == 0 && j == 0 || i == 0 && j == ncoord - 1 || i == ncoord - 1 && j == 0)
```

装 订 线

```
                continue;
            PlaceMatrix(m, *align, 5, 5, left, top);
        }
}

void SetTiming(const QRVersion &Q, char(*m)[177])
{
    /* 填充Timing pattern 1代表深色，2代表浅色 */
    const int pos1 = 6, pos2 = Q.SideSize - 8;
    int i;
    for (i = pos1; i < pos2; i++)
        if (i % 2 == 0)
            m[i][pos1] = m[pos1][i] = 1;
        else
            m[i][pos1] = m[pos1][i] = 2;
}

void SetDark(const QRVersion &Q, char(*m)[177])
{
    /* 填充Dark pattern 1代表深色 */
    m[Q.Version * 4 + 9][8] = 1;
}

void FormatReserved(const QRVersion&Q, char(*m)[177])
{
    /* 保留Format区域，用-1填充区分 */
    const int pos1 = 8, pos2 = Q.SideSize - 8;
    int   j;
    for (j = 0; j <= pos1; j++)
        if (j != 6)
            m[j][pos1] = m[pos1][j] = -1;
    m[pos1][pos2] = -1;
    for (j = pos2 + 1; j < Q.SideSize; j++)
        m[pos1][j] = m[j][pos1] = -1;
}

void VersionReserved(const QRVersion&Q, char(*m)[177])
{
    /* 保留Version区域，用-1填充区分 */
    if (Q.Version < 7)
        return;
    const int pos = Q.SideSize - 11;
    int i, j;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 6; j++)
            m[pos + i][j] = m[j][pos + i] = -1;
}

void ConvertBitInfo(char &ori, char &des)
{/* 将bit字符串转化，4代表浅色，3代表深色 */
    if (ori == '0')
        des = 4;
    else if (ori == '1')
        des = 3;
}

/*
函数功能：布置模块
Q：版本信息
```

```
m：二维码矩阵首行指针
bitString：填充的数据bit字符串
*/
void ModulePlacement(const QRVersion&Q, char(*m)[177], char *bitString)
{
	SetFinder(Q, m);
	SetSeperator(Q, m);
	SetAlignMent(Q, m);
	SetTiming(Q, m);
	SetDark(Q, m);
	FormatReserved(Q, m);
	VersionReserved(Q, m);

	int maxside = (Q.Version - 1) * 4 + 21;
	int ColBlock = (Q.Version - 1) * 2 + 10;
	int RightPos;
	char *pbitString = bitString;
	int i, j;
	for (i = ColBlock - 1; i >= 0; i--)//填充数据区域
	{
		RightPos = i * 2 + 1;
		if (i > 2)
			RightPos++;
		if (i % 2)
			for (j = maxside - 1; j >= 0; j--)
			{
				if (m[j][RightPos] == 0)
					ConvertBitInfo(*pbitString++, m[j][RightPos]);
				if (m[j][RightPos - 1] == 0)
					ConvertBitInfo(*pbitString++, m[j][RightPos - 1]);
			}
		else
			for (j = 0; j < maxside; j++)
			{
				if (m[j][RightPos] == 0)
					ConvertBitInfo(*pbitString++, m[j][RightPos]);
				if (m[j][RightPos - 1] == 0)
					ConvertBitInfo(*pbitString++, m[j][RightPos - 1]);
			}
	}
}

/*********************************
	掩码及评估部分
*********************************/

void E1calcu(char &ch, int &penalty, int clear = 0)
{/*  评价方式1的计算函数*/
	static int white, black;
	if (clear)
		white = black = 0;
	else if (ch)
	{
		white = 0;
		black++;
		if (black == 5)
			penalty += 3;
		else if (black > 5)
			penalty++;
```

装 订 线

```
        }
        else
        {
                white++;
                black = 0;
                if (white == 5)
                        penalty += 3;
                else if (white > 5)
                        penalty++;
        }
}
void OppositeBit(char &ch)
{/* 3代表深色  4代表浅色  */
        if (ch == 3)
                ch = 0;
        else if (ch == 4)
                ch = 1;
}

int Evaluate1(const QRVersion&Q, char(*m)[177])
{
        int penalty = 0;
        int i, j;
        for (i = 0; i < Q.SideSize; i++)
                for (j = 0; j < Q.SideSize; j++)
                        E1calcu(m[i][j], penalty);

        E1calcu(m[0][0], penalty, 1);//清零数据

        for (i = 0; i < Q.SideSize; i++)
                for (j = 0; j < Q.SideSize; j++)
                        E1calcu(m[j][i], penalty);
        return penalty;
}
int Evaluate2(const QRVersion&Q, char(*m)[177])
{
        int penalty = 0;
        int i, j;
        for (i = 0; i < Q.SideSize - 1; i++)
                for (j = 0; j < Q.SideSize - 1; j++)
                        if (m[i][j] == m[i][j + 1] && m[i][j] == m[i + 1][j] && m[i][j] == m[i + 1][j + 1])
                                penalty += 3;
        return penalty;
}
int Evaluate3(const QRVersion&Q, char(*m)[177])
{
        char find[] = { 1,0,1,1,1,0,1 };
        const int nfind = sizeof(find) / sizeof(char);
        int penalty = 0;
        int i, j, k;
        for (i = 0; i < Q.SideSize; i++)
                for (j = 0; j < Q.SideSize - nfind; j++)
                {
                        for (k = 0; k < nfind; k++)
                                if (find[k] != m[i][j + k])
                                        break;
                        if (k == nfind)
                                penalty += 40;

                        for (k = 0; k < nfind; k++)
```

装 订 线

```
                    if (find[k] != m[j + k][i])
                        break;
                if (k == nfind)
                    penalty += 40;
            }
    return penalty;
}
int Evaluate4(const QRVersion&Q, char(*m)[177])
{
    int penalty = 0;
    int black = 0;
    int i, j;
    for (i = 0; i < Q.SideSize; i++)
        for (j = 0; j < Q.SideSize; j++)
            black += m[i][j];
    black = int(((double)black / (Q.SideSize*Q.SideSize)) * 100);
    int low = black / 5 * 5;
    int up = (black + 4) % 5;
    low = abs(low - 50) / 5;
    up = abs(up - 50) / 5;
    penalty = 10 * (low > up ? up : low);
    return penalty;
}

void MaskingMode(const QRVersion &Q, const int Mode, char(*m)[177], char(*mask)[177])
{/* 进行八种掩模 */
    int row, column;
    for (row = 0; row < Q.SideSize; row++)
        for (column = 0; column < Q.SideSize; column++)
        {
            if (m[row][column] == -1 || m[row][column] == 2)
                mask[row][column] = 0;
            else
                mask[row][column] = m[row][column];
            if (mask[row][column] == 3 || mask[row][column] == 4)
            {
                if (Mode == 0 && (row + column) % 2 == 0)
                    OppositeBit(mask[row][column]);
                else if (Mode == 1 && row % 2 == 0)
                    OppositeBit(mask[row][column]);
                else if (Mode == 2 && column % 3 == 0)
                    OppositeBit(mask[row][column]);
                else if (Mode == 3 && (row + column) % 3 == 0)
                    OppositeBit(mask[row][column]);
                else if (Mode == 4 && (row / 2 + column / 3) % 2 == 0)
                    OppositeBit(mask[row][column]);
                else if (Mode == 5 && (row * column) % 2 + (row * column) % 3 == 0)
                    OppositeBit(mask[row][column]);
                else if (Mode == 6 && ((row * column) % 2 + (row * column) % 3) % 2 == 0)
                    OppositeBit(mask[row][column]);
                else if (Mode == 7 && ((row + column) % 2 + (row * column) % 3) % 2 == 0)
                    OppositeBit(mask[row][column]);
                mask[row][column] %= 2;
            }
        }
}
int EvaluateMask(const QRVersion&Q, char(*mask)[177])
{/* 评价掩模方式 */
    int penalty = 0;
    int(*const p[])(const QRVersion&, char(*)[177]) = { Evaluate1 ,Evaluate2, Evaluate3, Evaluate4 };
```

```
    int i;
    for (i = 0; i < 4; i++)
        penalty += p[i](Q, mask);
    return penalty;
}

/* **********************************
函数功能：进行掩模
Q：版本信息  m：原矩阵  mask：掩模后的矩阵  pmode：以引用返回掩模方式
*/
void Masking(const QRVersion &Q, char(*m)[177], char(*mask)[177], int &pmode)
{
    int penalty = 0;
    pmode = 0;
    int i, tem;
    for (i = 0; i < 8; i++)
    {
        MaskingMode(Q, i, m, mask);
        tem = EvaluateMask(Q, mask);
        if (i == 0)
            penalty = tem;
        if (penalty >= tem)
            pmode = i;
    }
    MaskingMode(Q, pmode, m, mask);
}

/**********************************
    填充格式版本信息部分
**********************************/

/*
    函数功能：填充格式版本信息
    Q：版本信息  m：二维码矩阵  pmode：掩模方式
*/
void SetFormatVersion(const QRVersion &Q, char(*m)[177], int pmode)
{
    char Format[16] = { 0 };
//  GenerFormatString(Q, Format, pmode);//采用计算方式
    strcpy(Format, F[VersionLevel * 8 + pmode].fillString);

    int i, j, k;
    for (i = 8, j = 0, k = 0; j < Q.SideSize; j++, k++)
    {
        if (j == 6)
            j++;
        else if (j == 8)
            j = Q.SideSize - 7;
        m[i][j] = Format[k] - '0';
    }
    for (i = Q.SideSize - 1, j = 8, k = 0; i >= 0; i--, k++)
    {
        if (i == Q.SideSize - 8)
            i = 8;
        else if (i == 6)
            i--;
        m[i][j] = Format[k] - '0';
    }
```

```
    if (Q.Version >= 7)//采用查表方式
        for (i = 0, k = 17; i < 6; i++)
            for (j = Q.SideSize - 11; j < Q.SideSize - 8; j++)
                m[i][j] = m[j][i] = V[Q.Version - 7].fillString[k--] - '0';
}

/*********************************
    控制台打印显示部分
*********************************/

typedef BOOL(WINAPI *PROCSETCONSOLEFONT)(HANDLE, DWORD);

void setcolor(HANDLE hout, const int bg_color, const int fg_color)
{
    SetConsoleTextAttribute(hout, bg_color * 16 + fg_color);
}
void setconsolefont(const HANDLE hout, const int font_no)
{
    HMODULE hKernel32 = GetModuleHandleA("kernel32");
    PROCSETCONSOLEFONT setConsoleFont = (PROCSETCONSOLEFONT)GetProcAddress(hKernel32,
"SetConsoleFont");
    setConsoleFont(hout, font_no);
    return;
}
void setconsoleborder(const HANDLE hout, const int cols, const int lines)
{
    char cmd[80];

    system("cls");
    sprintf(cmd, "mode con cols=%d lines=%d", cols, lines);
    system(cmd);

    return;
}

void ScreenOutput(const QRVersion &Q, char(*m)[177])
{
    const HANDLE hout = GetStdHandle(STD_OUTPUT_HANDLE);
    const int size = Q.SideSize;
    if (Q.Version>6)
        setconsolefont(hout, 0);
    else
        setconsolefont(hout, 1);

    setconsoleborder(hout, 2 * (size + 4), size + 4);

    int i, j;
    for (i = -1; i <= size; i++)
    {
        for (j = -1; j <= size; j++)
        {
            if (i == -1 || i == size || j == -1 || j == size || m[i][j] == 0)
                setcolor(hout, NormalWhite, NormalWhite);
            else if (m[i][j] == 1)
                setcolor(hout, NormalBlank, NormalBlank);
            cout << "  ";
        }
        cout << endl;
    }
}
```