装 订 线

QRcoder实现报告

班级：计算机一班
姓名：王哲源
学号：1652228

完成日期：2017.5.20

1.题目及其描述

题目主要目的在于通过编写程序实现 QRcode 的编码过程并将编码后的 QRcode 通过 cmd 界面打印出可被二维码扫描器识别的二维码，二维码的编码主要有以下步骤

1.1 数据识别

通过识别输入数据的分析以及数据类型的识别以确定数据的类别，以便采用最优的编码方式对数据进行编码

1.2 数据编码

通过 1.1 中确定的数据类型对数据进行编码

1.3 生成纠错码

通过 1.2 中生成的编码后数据对编码生成对应的纠错码，以便识别器在扫描时利用纠错码对数据进行纠错

1.4 最终编码

通过对 1.2 中的编码数据以及 1.3 中生成的纠错码采用一定的排列数据进行重新编码，从而生成数据区的最终编码

1.5 矩阵中的模块放置

将 1.4 生成的最终编码联合二维码中一系列的功能模块按一定的标准放置于矩阵块中，完成初步的二维码图案

1.6 掩码图案

将 1.5 中生成的最初版本的二维码图案进行掩码编码，通过对图案中部分编码按照一定方式进行修改从而使得扫描器能够更好的识别出二维码图案

1.7 格式和版本信息

在 1.6 中生成的二维码中预留的空位中填入格式信息以及（有时需要）填入版本信息，前者定义了纠错等级以及掩码图案模式，后者则存在于较大版本二维码中，定义了二维码的矩阵大小。从而形成最终的二维码图案

2.整体设计思路

以下对于程序的一些设计思路进行概述

2.1 关于数据分析

首先是对数据的分析，由于要先确定二维码的 Version，而字符编码又能编码除了数字以外的编码（虽然不是最优编码），因此这里直接判断数据是否为纯数字编码即可，以此来分别确定二维码的 Version 即可。

而字符容量表可以通过打表的方式预置于头文件中，使用时直接调用与实际数据进行比对即可

2.2 关于数据编码

通过 2.1 中确定的 Version 以及数据类型对数据，分别对纯数字以及字符类型的数据进行编码即可。

2.3 关于纠错码的添加

鉴于题目要求最多对字符数为 100 的数据进行编码即可，因此此处统一采用最高纠错级别 H 进行纠错码的编写。

同 2.1，块大小信息也可以通过打表方式预置于头文件之中，使用时直接从头文件调用比对即可。

伽罗瓦算法中所需要使用到的 α 因子以及多项式除法运算中需要的对数-反对数转换都可以通过打表的方式预置于头文件，方便长除法时的直接调用

2.4 关于最终编码

调用 2.2 中的数据编码以及 2.3 中的纠错码，同时利用头文件中打表预置的块大小信息，直接对上述两码进行交叉排列，从而完成数据的最终编码

2.5 关于二维码矩阵中模块的放置

利用 2.4 中的编码，在二维矩阵中根据 0、1 的情况放置白、黑像素块

同时对于 Finder Patterns 以及 Alignment Patterns 也可以通过打表预置块像素信息，在放置时直接循环数组进行放置

2.6 关于掩码图案

掩码图案主要是对于现有像素块的坐标计算决定是否更换色块颜色，可以利用一个数组暂存该方案的掩码图案，评分后确定是否作为当前最优方案即可。

而评估方案中的第三种评价条件是基于色块固定排序的，也可以通过打表预置方便判断时直接循环调用。

2.7 关于格式和版本信息

由于格式信息要基于掩码方案进行编写，因此格式信息直接在掩码编写部分进行操作即可，版本信息则需要另行编写。两者所需的 01 串也可通过打表预置于头文件中。

特别注意，该部分所需空间需要在 2.5 中预留！

3.主要功能实现

以下对整个程序的具体实现进行较为详细的介绍

3.1 数据分析部分

首先先对输入数据进行一次遍历，若发现非数字字符立刻退出，并返回 1 表示为非纯数字字符。

接下来通过二分法来夹逼出最小所需要的 Version，以确定该二维码所需要使用的最小 Version，以便后面的操作

3.2 编码部分

首先先是通过 3.1 中确定的 Version 确定 Group 的数量，接下来通过各 Group 的预置信息进行分块编码。编码这里是直接使用了在 Github 上搜寻到的代码中 EncodeSourceData 部分，因此这里不进行描述了。

编码之后再通过预置信息对于已编码部分添加模式指示符以及（如果必要的）补齐码完成初步的数据编码。同时由于固定使用 H 级纠错，因此在完成初步编码之后，利用预置数据信息，可以预先将编码交叉排列，为纠错码留出空间，方便之后的纠错码直接填入。

3.3 纠错码部分

纠错码的生成是通过类似多项式长除法的方式生成的，由于 3.2 中已经为纠错码腾出了空间，因此这边只需要生成纠错码后直接填入即可。

这里首先是利用预置的块信息对已有的数据进行分块，利用其中数据信息生成信息多项式，接下来调用纠错码生成函数利用多项式长除法生成纠错码。

生成函数利用的是为 Reed-Solomon 纠错法，其运算方式类似多项式长除法，但是运作与伽罗瓦域之下的。因此首先利用预置的表取出 α 系数，生成生成多项式，接下来利用一个 tail 指针指向余多项式的头，不断进行伽罗瓦域下的多项式长除法，最后返回的 tail 指针所指位置之后的信息多项式系数即为纠错码

3.4 像素块的初步放置

像素块的放置可以分为以下几个步骤

首先先是 function 模块，由于 Finder Patterns、Finder Patterns、Seperator、Alignment Patterns 以及 Timing Patterns 不涉及掩码操作且均有固定位置，因此这里将这几个模块包括保留模块（用于存放版本及格式信息的部分）以及暗模块最先放置入二维码矩阵

接下来是按照固定的顺序填入数据部分，这点问题不大，但特别要注意避开已填入的

function 模块同时注意 Timing Patterns 处的放置规则。

3.5 掩码及最终二维码生成

在填入完 function 模块及 data 模块之后就要对初步生成的二维码进行掩码的添加。这里注意由于掩码只作用于 data 模块,因此为了方便区别两者,除了奇数黑偶数白的方式之外,function 模块的黑白数字均为 4 位以上二进制数，而 data 模块则为 2 位以下，同时这里特别注意保留区域应当与两者有所区别（既不受掩码修改但仍需要被添加数据)!

接下来是对于每个数据模块的掩码添加。将 3.4 执行结束后的 QR 源码拷贝至一个暂时数组中进行掩码添加，每当完成掩码添加之后在其中填入格式信息（格式信息的生成这里调用了外部已有代码，也可打表预置)，接下来进行掩码的评估。评估中共有 4 种记分情况，这里不进行赘述。得到评分后与最低分数进行比对，若得分更低则说明当前方案更优，对最优方案进行更新后继续尝试其他方案，直至全部方案尝试完毕后获得最优方案即为最终二维码。

（对于 Version 7 以上二维码还需要再填入版本信息，这里版本信息的 01 串由调用已有函数生成，当然也可以通过打表预置内置于程序直接填入）

装

4. 调试过程中遇到的问题

以下对整个程序的具体实现进行较为以下将对调试过程中遇到的几个主要问题进行概述

订

4.1 Timing Pattern 缺失

在程序写完最初测试时发现二维码无法正确生成，经过分块检测最初以为 function 模块的放置没问题，但经过输出发现编码部分并没有错误。

线

最后在将保留部分的颜色调为白色时发现 Timing Pattern 紧邻 Finder Pattern 的部分出现缺失，而这正说明是在编写保留模块时并未越过已填写入的 Timing Pattern 部分，导致保留部分直接覆盖了 Timing Pattern 部分。因此在保留模块部分中添加一句将被覆盖的 Timing Pattern 部分重新填入以修复被误改的部分

4.2 掩码出错

4.1 问题解决后虽然 function 模块问题被修正，但加入掩码之后图形还是出现问题，通过输出编码信息发现编码信息没问题，最后是通过静态查错发现，对于 QR[i][j]的色块，其对应横纵坐标应该为(j,i)，同时掩码变换中的&运算与+、*运算优先级也是一片混乱。

在解决掩码部分后图形问题仍然没解决，输出中间过程发现每步 memcpy 给暂存 QRcode 数组的数据只有前 4 字节，才明白自己对指针变量又进行了 sizeof 操作，复制部分仅复制了前 4 字节部分。

除了前两个部分出错，经过逐条还发现了自己的另一个思维漏洞。原来的思路是对每种掩码方案执行一次时与最低得分进行比对，若低于最低得分则将当前方案作为最优方案赋给结果数组，但实际上每次执行都要对原已经放置一部分色块的数组进行相同的操作，因而不能一找到当前最优方案就改变原数组，而应该是暂存于另一个数组中当执行完毕之后再将该数组赋给结果数组，

否则会导致其他方案出错。

5.总结部分

5.1 教训

这次的二维码编写暴露出了自己一些基础知识的不牢固，比如在位运算与数值运算间的优先级混乱，所幸 VS2015 的编译器会对运算优先级可能出错的地方进行 warning 才避免了更大工作量的后期查错；还有一个就是老师强调无数次的不要对数组指针进行 sizeof 操作的问题还是犯了，而且严重影响了程序的运行也对查错造成了不小的影响（因为这一个问题我在上面浪费了至少半小时的时间）

同时，思路的过于简单化（如 4.2 中提及的第三个问题）也对程序的查错造成了不小的麻烦，这点值得反省

5.2 收获

由于二维码中涉及大量的数学运算以及数学转化思想，这次的程序在编写前及编写时查阅了大量的文字资料以及查看了大量的他人代码，了解了诸多关于数据编码方面以及二维码生成的知识，对于自己的知识面拓展有十足的帮助。

同时通过阅读 github 上的参考代码（同时也将其中的代码作为了编码部分的引用代码），其中通过打表的方式预置大量固定的信息的方式也受到了很多启发（仅头文件有 700 行，其中预置数据部分占了 600 行）

资料参考来源：http://tiierr.xyz/2017/02/24/二维码-生成原理/

代码参考来源：https://github.com/myang-git/QR-Code-Encoder-for-Objective-C

6.附件：程序（仅为 myself 部分）

```
    int encode(const char *const s, const int
len, bool(*QRcode)[MAX_MODULESIZE])
    {
        bool is_digit = check_str(s, len);
        int  ver  =  get_encode_version(len,
is_digit);
        if (ver == -1)
            return -1;

        unsigned                          char
data[MAX_DATACODEWORD];
        int databit;
        encode_data(ver,   s,   len,   data,
databit);
        extend_data(ver, data, databit);

        unsigned char code[MAX_ALLCODEWORD];
        recode(ver, data, code);
        ECcode_append(ver, data, code);

        unsigned                          char
Orgin_QR[MAX_MODULESIZE][MAX_MODULESIZE];
        memset(Orgin_QR,                  0,
sizeof(Orgin_QR));
        Function_Part(ver,       Orgin_QR);
Version_Info(ver, Orgin_QR);
        Data_Part(ver, code, Orgin_QR);
        Msk_Part(ver, Orgin_QR);
        Version_Info(ver, Orgin_QR);

        const int margin = ver * 4 + 17;
        sFor(i, 0, margin) sFor(j, 0, margin)
            QRcode[i][j] = Orgin_QR[i][j] &
1;

        return ver;
    }

    bool check_str(const char *const s, const
int len)
    {
        For(i, 1, len)
            if (!(s[i] >='0' && s[i] <='9'))
                return 0;
```

```
        return 1;
    }

    int     get_encode_version(const     int
len,const bool is_digit)
    {
        int Nbit = is_digit ? calc_digit(len) :
calc_ascii(len);
        int l = 1, r = 40;
        while (l != r)
        {
            int mid = (l + r) >> 1;
            int Maxbit = calc_bit(mid);
            Maxbit < Nbit ? l = mid + 1 : r
= mid;
        }
        return l > 40 ? -1 : l;
    }
    int calc_digit(const int len)
    {
        int Div = len / 3, Mod = len % 3;
        int res = Div * 10 + 3 * Mod + (Mod !=
0);
        return res + (len <= 1425 ? (len <= 235 ?
14 : 16) : 18);
    }
    int calc_ascii(const int len)
    {
        return 8 * len + (len <= 98 ? 12 : 50);
    }
    int calc_bit(const int ver)
    {
        int formbit = ver <= 6 ? 31 : 67;
        int funbit = ver == 1 ? 202 : (int)(180
+ 25 * pos_bit(ver) + 2 * (4 * ver + 17) - 10
* sqrt(3 + pos_bit(ver)));
        int databit = (4 * ver + 17)*(4 * ver
+ 17) - formbit - funbit;
        return          (databit          -
(QR_VersonInfo[ver].ncAllCodeWord          -
QR_VersonInfo[ver].ncDataCodeWord[3]) * 8) / 8
* 8;
    }
    int pos_bit(const int ver)
    {
        if (ver == 1)
            return 0;
        if (ver>1 && ver <= 6)
            return 1;
        if (ver>6 && ver <= 13)
            return 6;
        if (ver>13 && ver <= 20)
            return 13;
        if (ver>20 && ver <= 27)
            return 22;
        if (ver>27 && ver <= 34)
            return 33;
        // ver>34 && ver <= 40
        return 46;
    }

    void encode_data(const int ver, const char
*const s, const int len, unsigned char *data,
int &databit)
    {
        int Group = ver >= 27 ? 2 : (ver >= 10 ?
1 : 0);
        For(i, Group, 2)
        {
            if (!EncodeSourceData(s + 1, len,
i, data, databit)) /////s+1 !!! std 下标从 0
开始
                continue;
            if (!i)
            {
                For(j, 1, 9)
                    if ((databit + 7) / 8 <=
QR_VersonInfo[j].ncDataCodeWord[QR_LEVEL_H])
                        return;
            }
            else if (i == 1)
            {
                For(j, 10, 26)
                    if ((databit + 7) / 8 <=
QR_VersonInfo[j].ncDataCodeWord[QR_LEVEL_H])
                        return;
            }
            else //i == 2
            {
                For(j, 27, 40)
                    if ((databit + 7) / 8 <=
QR_VersonInfo[j].ncDataCodeWord[QR_LEVEL_H])
                        return;
            }
            /////////else if 位置注意!!! 大
括号不可省!!! r
        }
    }
    void extend_data(const int ver, unsigned
char *data, int &databit)
    {
        int            infobit            =
QR_VersonInfo[ver].ncDataCodeWord[QR_LEVEL_H
];
        int
zerobit=8*infobit-databit>4?4:infobit-databi
t;
        if (zerobit > 0)
            databit += zerobit;

        const unsigned char extendbit[2] =
{ 0xec,0x11 };
```

```
        int pre = 0;
        sFor(i, (databit + 7) / 8, infobit)
            data[i] = extendbit[pre],pre ^= 1;
    }

    void recode(const int ver, const unsigned char *const data, unsigned char *code)
    {
        int                         codebit= QR_VersonInfo[ver].ncAllCodeWord;
        memset(code, 0, codebit);

        int          Block1bit          = QR_VersonInfo[ver].RS_BlockInfo1[QR_LEVEL_H].ncRSBlock;
        int          Block2bit          = QR_VersonInfo[ver].RS_BlockInfo2[QR_LEVEL_H].ncRSBlock;
        int Blockbit = Block1bit + Block2bit;

        int           data1bit          = QR_VersonInfo[ver].RS_BlockInfo1[QR_LEVEL_H].ncDataCodeWord;
        int           data2bit          = QR_VersonInfo[ver].RS_BlockInfo2[QR_LEVEL_H].ncDataCodeWord;

        int pdata = 0, pcode = 0;
        sFor(i, 0, Block1bit)
        {
            sFor(j, 0, data1bit)
                code[j*Blockbit + pcode] = data[pdata++];
            ++pcode;
        }
        sFor(i, 0, Block2bit)
        {
            sFor(j, 0, data2bit)
            {
                unsigned char &tcode = code[j < data1bit ? j*Blockbit + pcode : data1bit*Blockbit + i];
                tcode = data[pdata++];
            }
            ++pcode;
        }
    }

    void ECcode_append(const int ver, const unsigned char *const data, unsigned char *code)
    {
        int          Block1bit          = QR_VersonInfo[ver].RS_BlockInfo1[QR_LEVEL_H].ncRSBlock;
        int          Block2bit          = QR_VersonInfo[ver].RS_BlockInfo2[QR_LEVEL_H].ncRSBlock;
        int Blockbit = Block1bit + Block2bit;

        int           databit           = QR_VersonInfo[ver].ncDataCodeWord[QR_LEVEL_H];
        int           data1bit          = QR_VersonInfo[ver].RS_BlockInfo1[QR_LEVEL_H].ncDataCodeWord;
        int           poly1bit          = QR_VersonInfo[ver].RS_BlockInfo1[QR_LEVEL_H].ncAllCodeWord - data1bit;
        int           data2bit          = QR_VersonInfo[ver].RS_BlockInfo2[QR_LEVEL_H].ncDataCodeWord;
        int           poly2bit          = QR_VersonInfo[ver].RS_BlockInfo2[QR_LEVEL_H].ncAllCodeWord - data2bit;

        int pdata = 0, pcode = 0;
        sFor(i, 0, Block1bit)
        {
            int tail = 0;
            unsigned char MsgPoly[2 * MAX_CODEBLOCK];
            memset(MsgPoly,              0, sizeof(MsgPoly));
            memcpy(MsgPoly, data + pdata, data1bit);
            encode_ECcode(MsgPoly,   tail, data1bit, poly1bit);

            sFor(j, 0, poly1bit)
                code[j*Blockbit + databit + pcode] = MsgPoly[tail + j];
            pdata += data1bit;
            ++pcode;
        }
        sFor(i, 0, Block2bit)
        {
            int tail = 0;
            unsigned char MsgPoly[2 * MAX_CODEBLOCK];
            memset(MsgPoly,              0, sizeof(MsgPoly));
            memcpy(MsgPoly, data + pdata, data2bit);
            encode_ECcode(MsgPoly,   tail, data2bit, poly2bit);

            sFor(j, 0, poly2bit)
                code[j*Blockbit + databit + pcode] = MsgPoly[tail + j];
            pdata += data2bit;
            ++pcode;
```

```
        }
    }
    void encode_ECcode(unsigned char*Msg, int &tail, const int Msgbit, const int polybit)
    {
        int *poly = new(nothrow) int[polybit];
        if (poly == NULL)
        {
            printf("No Free Memory\n");
            exit(-1);
        }
        sFor(i, 0, polybit)
            poly[i] = byRSExp[polybit][i];
        sFor(i, 0, Msgbit)
        {
            int cur = tail++;
            if (!Msg[cur])
                continue;
            unsigned char cur_exp = byIntToExp[Msg[cur]];
            For(j, cur + 1, polybit + cur)
            {
                unsigned char texp = (unsigned char)((poly[j - cur - 1] + cur_exp)%255);
                Msg[j] ^= byExpToInt[texp];
            }
        }
        delete[]poly;
    }

    void Function_Part(const int ver, unsigned char(*QR)[MAX_MODULESIZE])
    {
        Finder_Patterns(ver, QR);
        Seperator(ver, QR);
        Alignment_Patterns(ver, QR);
        Timing_Patterns(ver, QR);
        Dark_Module_And_Retain(ver, QR);
    }
    void Finder_Patterns(const int ver, unsigned char(*QR)[MAX_MODULESIZE])
    {
        int margin = ver * 4 + 17;
        const bool base_pattern[Finder_MARGIN][Finder_MARGIN] =
        {
            { 1,1,1,1,1,1,1 },
            { 1,0,0,0,0,0,1 },
            { 1,0,1,1,1,0,1 },
            { 1,0,1,1,1,0,1 },
            { 1,0,1,1,1,0,1 },
            { 1,0,0,0,0,0,1 },
            { 1,1,1,1,1,1,1 }
        };

        sFor(i, 0, Finder_MARGIN) sFor(j, 0, Finder_MARGIN)
            QR[i][j] = QR[i][margin - 7 + j] = QR[margin - 7 + i][j] = base_pattern[i][j] ? fun_Black : fun_White;
    }
    void Seperator(const int ver, unsigned char(*QR)[MAX_MODULESIZE])
    {
        int margin = ver * 4 + 17;
        For(j, 0, 7)
            QR[7][j] = QR[7][margin - 8 + j] = QR[margin - 8][j] = fun_White;
        For(i, 0, 7)
            QR[i][7] = QR[i][margin - 8] = QR[margin - 8 + i][7] = fun_White;
        QR[7][7] = QR[margin - 7][7] = QR[7][margin - 7] = fun_White;
    }
    void Alignment_Patterns(const int ver, unsigned char(*QR)[MAX_MODULESIZE])
    {
        int cnt = QR_VersonInfo[ver].ncAlignPoint;
        sFor(i, 0, cnt)
        {
            sFor(j, 0, cnt)
                add_AlPattern(QR, QR_VersonInfo[ver].nAlignPoint[i], QR_VersonInfo[ver].nAlignPoint[j]);
            add_AlPattern(QR, QR_VersonInfo[ver].nAlignPoint[i], Alignment_Common);
            add_AlPattern(QR, Alignment_Common, QR_VersonInfo[ver].nAlignPoint[i]);
        }
    }
    void add_AlPattern(unsigned char(*QR)[MAX_MODULESIZE], const int x, const int y)
    {
        const bool base_pattern[Alignment_MARGIN][Alignment_MARGIN] =
        {
            { 1,1,1,1,1 },
            { 1,0,0,0,1 },
            { 1,0,1,0,1 },
            { 1,0,0,0,1 },
            { 1,1,1,1,1 }
        };
        if (QR[x][y])
            return;
```

```
        For(i, -2, 2) For(j, -2, 2)
            QR[x + i][y + j] = base_pattern[i
+ 2][j + 2] ? fun_Black : fun_White;
    }
    void  Timing_Patterns(const  int  ver,
unsigned char(*QR)[MAX_MODULESIZE])
    {
        int margin = ver * 4 + 17;
        For(k, 8, margin - 9)
            QR[k][6] = QR[6][k] = k & 1 ?
fun_White : fun_Black;
    }
    void Dark_Module_And_Retain(const int ver,
unsigned char(*QR)[MAX_MODULESIZE])
    {
        int margin = ver * 4 + 17;
        QR[margin - 8][8] = fun_Black;
        QR[8][8] = Retain;
        For(j, 0, 7)
            QR[8][j] = QR[8][margin - 8 + j]
= Retain;
        For(i, 0, 7)
            QR[i][8] = QR[margin - 8 + i][8]
= Retain;
        QR[6][8] = QR[8][6] = fun_Black;
//Timing Pattern
        if (ver <= 6)
            return;
        For(i, margin - 11, margin - 9) For(j,
0, 5)
            QR[i][j] = Retain;
        For(i, 0, 5) For(j, margin - 11,
margin - 9)
            QR[i][j] = Retain;
    }

    void  Data_Part(const  int  ver,  const
unsigned  char  *const  code,  unsigned
char(*QR)[MAX_MODULESIZE])
    {
        int margin = ver * 4 + 17;
        int x = margin, y = margin - 1;
        int deltax = 1, deltay = 1;
        int            totbit            =
QR_VersonInfo[ver].ncAllCodeWord;
        sFor(i, 0, totbit) opFor(j, 7, 0)
        {
            do
            {
                x += deltax;
                deltax = -deltax;
                if (deltax < 0)
                {
                    y += deltay;
                    if (y < 0 || y ==
margin)
                    {
                        y = y < 0 ? 0 :
margin - 1;
                        deltay = -deltay;
                        x -= 2;

                        if  (x  ==  6)
//*****Timing Pattern
                            --x;
                    }
                }
            } while (QR[x][y]);

            //printf("%d %d %d\n", x, y,
code[i] & (1 << j));
            QR[x][y] = (code[i] & (1 << j)) ?
data_Black : data_White;
        }
    }

    void Msk_Part(const int ver, unsigned
char(*QR)[MAX_MODULESIZE])
    {
        unsigned                        char
mskcode[MAX_MODULESIZE][MAX_MODULESIZE];
        unsigned                        char
best[MAX_MODULESIZE][MAX_MODULESIZE];
        int lowest_score = 19980413;
        int tmp = 0;

        For(i, 0, 7)
        {
            memcpy(mskcode,             QR,
sizeof(mskcode)); //QR是指针啊!!!!!!!!

            add_msk(ver, i, mskcode);
            Format_Info(ver, i, mskcode);

            int cur_score = Evaluate_Msk(ver,
mskcode);
            if (cur_score >= lowest_score)
                continue;
            lowest_score = cur_score;
            tmp = i;
            memcpy(best,            mskcode,
sizeof(mskcode));//QR不能直接修改 原版还要用
        }
        memcpy(QR, best, sizeof(best));
    }
    void add_msk(const int ver, const int
msk_id, unsigned char(*QR)[MAX_MODULESIZE])
    {
        int margin = ver * 4 + 17;
        sFor(i, 0, margin) sFor(j, 0, margin)
        {
            if (!(QR[i][j] >> 3))
```

```
                continue;
            if (trans_msk(msk_id, j, i))
                continue;
            QR[i][j] ^= 1;
        }
    }
    int trans_msk(const int msk_id, const int x, const int y)
    {
        switch (msk_id)
        {
        case 0:
            return (x + y) & 1;
        case 1:
            return x & 1;
        case 2:
            return y % 3;
        case 3:
            return (x + y) % 3;
        case 4:
            return (x / 2 + y / 3) & 1;
        case 5:
            return (x*y & 1) + (x*y % 3);
        case 6:
            return ((x*y & 1) + (x*y % 3)) & 1; //先计算运算再位运算!!
        default: //case 7:
            return ((x + y & 1) + (x*y % 3)) & 1;
        }
    }
    int Evaluate_Msk(const int ver, const unsigned char (*const QR)[MAX_MODULESIZE])
    {
        int margin = ver * 4 + 17;
        int score = 0;
        //Condition 1
        sFor(i, 0, margin) for (int j = 0; j < margin - 4;)
        {
            int same = 1,k;
            for (k = j + 1; k < margin; ++k)
                if ((QR[i][j] & 1) == (QR[i][k] & 1))
                    ++same;
                else
                    break;
            if (same >= 5)
                score += same - 2;
            j = k;
        }
        sFor(j, 0, margin) for (int i = 0; i < margin - 4;)
        {
            int cnt = 1, k;
            for (k = i + 1; k < margin; ++k)
                if ((QR[i][j] & 1) == (QR[k][j] & 1))
                    ++cnt;
                else
                    break;
            if (cnt >= 5)
                score += cnt - 2;
            i = k;
        }
        //Condition 2
        sFor(i, 0, margin - 1) sFor(j, 0, margin - 1)
            if (((QR[i][j] & 1) == (QR[i][j + 1] & 1))
                && ((QR[i][j] & 1) == (QR[i + 1][j] & 1))
                && ((QR[i][j] & 1) == (QR[i + 1][j + 1] & 1)))
                score += 3;
        //Condition 3
        const int cond[2][11] = { {1,0,1,1,1,0,1,0,0,0,0},{0,0,0,0,1,0,1,1,1,0,1} };
        sFor(i, 0, margin) sFor(j, 0, margin - 10)
        {
            bool succ = 1;
            For(op, 0, 1) For(k, 0, 10)
                if ((QR[i][j + k] & 1) != cond[op][k])
                {
                    succ = 0;
                    break;
                }
            score += 40 * succ;
        }
        sFor(i, 0, margin - 10) sFor(j, 0, margin)
        {
            bool succ = 1;
            For(op, 0, 1) For(k, 0, 10)
                if ((QR[i + k][j] & 1) != cond[op][k])
                {
                    succ = 0;
                    break;
                }
            score += 40 * succ;
        }
        //Condition 4
        int cnt = 0;
        sFor(i, 0, margin) sFor(j, 0, margin)
            cnt += QR[i][j] & 1;
        int per = int(100.0*cnt / (margin*margin));
        int A = (per / 5) * 5;
```

```
        int B = (per / 5 + 1) * 5;
        int dA = abs(A - 50) / 5;
        int dB = abs(B - 50) / 5;
        score += Min(dA, dB) * 10;

        return score;
    }
```

装

订

线