同济大学

实验报告

装

订

线

二维码

计算机三班
1553449
王志业
2017.5.24

1. 题目及基本要求描述

1.1.基本要求

从键盘输入要生成的二维码的内容，包括文本（中英文，数字，空格，符号）和网址两种。
生成算法任选，要求以手机上任意软件扫描正确即可。
在VS2015 cmd界面下实现。

1.2.二维码简介

二维条码/二维码（2-dimensional bar code）是用某种特定的几何图形按一定规律在平面（二维方向上）分布的黑白相间的图形记录数据符号信息的；在代码编制上巧妙地利用构成计算机内部逻辑基础的“0”、“1”比特流的概念，使用若干个与二进制相对应的几何形体来表示文字数值信息，通过图象输入设备或光电扫描设备自动识读以实现信息自动处理：它具有条码技术的一些共性：每种码制有其特定的字符集；每个字符占有一定的宽度；具有一定的校验功能等。同时还具有对不同行的信息自动识别功能、及处理图形旋转变化点。

2. 整体设计思路

确定编码方式为8字节编码方式。
功能分为三大块实现：

2.1.输入内容转化为01比特流

通过位运算可以实现将输入的内容转为八位的二进制比特流，建立自定义的类型Bitstream其中data存储0，1符号，length表征0.1的长度。将输入内容转化完成后添加编码信息（信息的编码方式，信息的长度等），用规定的0，1字符串在信息比特流前添加好。在将其按八位一组转化为十进制待求纠错码用。

2.2.信息多项式，生成多项式共同在迦洛华域进行运算求纠错多项式

进入第二步，需要了解一些知识。

2.2.1 多项式长除法

1. 把被除式、除式按某个字母作降幂排列，并把所缺的项用零补齐。
2. 将分子的第一项除以分母的最高次项(即次数最高的项)，结果写在横线之上。
3. 将分母乘以刚得到结果（最终商的第一项），乘积写在分子前两项之下（同类项对齐）。
4. 从分子的相应项中减去刚得到的乘积(消去相等项，把不相等的项结合起来)，结果写在下面。然后，将分子的下一项“拿下来”。
5. 把减得的差当作新的被除式，重复前三步（直到余式为零或余式的次数低于除式的次数时为止．被除式=除式×商式+余式 ）
6. 重复第四步。直到得出结果。横线之上的多项式即为商，而剩下的数字就是余数

2.2.2迦洛华域及其算法

多项式长除法则用到了迦洛华域，其本质上是数字的有限集合，以及创建仍在该集合中的数字的一些数学运算。

二维码标准使用逐位模2算法和逐字节模100011101算法。这意味着使用迦洛华域2^8，换句话说是迦洛华域256，有时也写为GF(256)。

GF(256) 中的数字在0到255（包括）的范围内。也可以用八位字节表示的相同数字范围。

GF(256) 中的数字在0到255（包括）的范围内。GF(256)中的数学运算本质上是循环的，这意味着如果在结果大于255的数的GF(256)内执行数学运算，则需要使用模运算来获得仍然在迦洛华域中的数。

在迦洛华域中，负数具有与正数相同的值，因此-n = n。换句话说，总是使用迦洛华域算法中的数字的绝对值。

在迦洛华域内的加法和减法是相同的事情。迦洛华域中的加法和减法通过执行正常的加法或减法，然后模运算来计算。由于我们使用逐位模2算法（如二维码标准中所述），这与执行XOR运算相同。

2.2.3生成多项式和消息多项式

生成多项式$(\alpha^0x - \alpha^0) \times (\alpha^0x - \alpha^1) \times (\alpha^0x^1 + \alpha^2x^0)$。。。次数为纠错码的个数个相乘（多项式的乘法实现）

消息多项式为前面得到的输入信息的再转化为二进制的系数。需要提到的一点是，生成多项式和消息多项式的系数（coef）我们都用迦洛华域上面中α上面的指数来表示，方便我们的后续计算。

2.2.4生成纠错码

纠错码是生成多项式除消息多项式的余数的系数。对于除法我们用以下操作代替：

将生成多项式乘以消息多项式的首项然后与消息多项式进行异或运算。得到新的消息多项式。反复进行上述步骤进行消息多项式的信息数次，得到一个余式，系数即为纠错码。纠错码转为0.1比特流和消息多项式的0.1比特流合起来待用。(在此只涉及version1-5的L纠错级别，所以不涉及块的内容，只需简单的把两个比特流接起来)。

2.3.01比特流在二维码矩阵中的排布

二维码矩阵中需要一些固定的功能模块，如下图

二维码的大小可以用公式(((V-1)*4)+21)计算，其中V是二维码版本

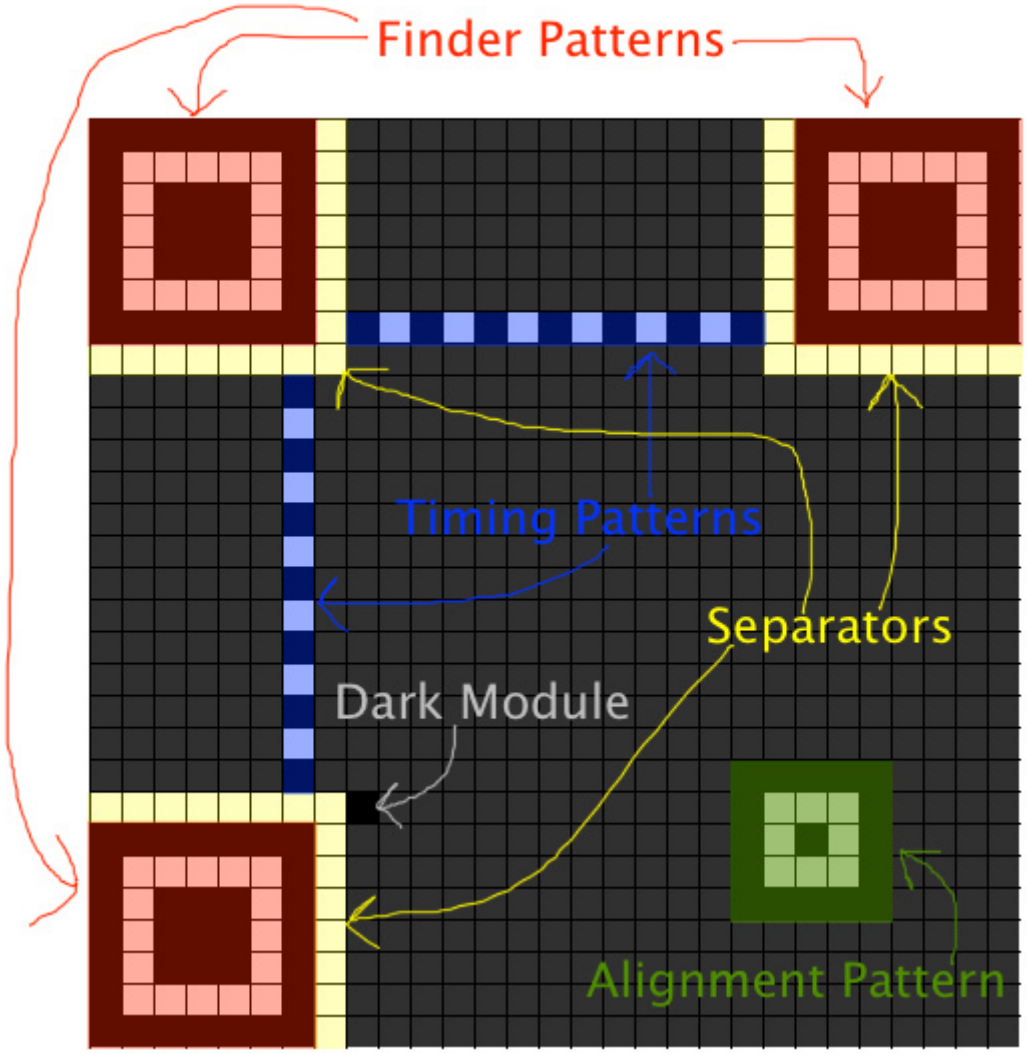
Finder patterns是二维码左上角，右上角和左下角的角落中的三个块。

Separators是finder patterns旁边的空白区域。

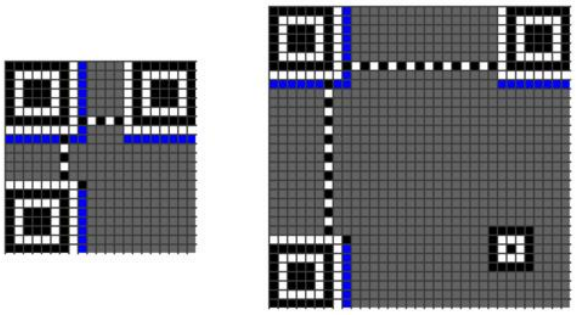
alignment patterns类似于finder patterns，但是更小，并且被放置在整个代码中。它们在版本2和更大版本中使用，它们的位置取决于二维码版本。

timing patterns是连接finder patterns的虚线。

dark module是一个单一的黑色模块，总是放置在左下角的finder patterns旁边。



保留格式信息区：蓝色区域为保留格式区，放置纠错级别，掩码模式，还有其纠错码



因版本较低，保留版本信息区不涉及。

3. 主要功能的实现

3.1比特流的转化

通过BitStream_newFromBytes函数将输入的信息流转化为八位比特流。首先放置4位模式信

息，字符模式位（0100），统计信息的字数转化位8位比特流，放在第二位。后面接信息流，在接结束码（0000），后面跟填充码。信息转化实现流程：

```
for (i = 0; (unsigned int)i < size; i++) {
    mask = 0x80;
    for (j = 0; j < 8; j++) {
        if (data[i] & mask) {
            *p = 1;
        }
        else {
            *p = 0;
        }
        p++;
        mask = mask >> 1;
    }
}
```

0x80,位运算取第一位data[i]&mask,取到1位1，取到0为0，记录在BitStream的data里面。

3.2.比特流转化为信息

是上述过程的逆过程

3.3多项式乘法

MultiplyPoly(polynomial *La, polynomial *Lb)La,Lb分别代表两个多项式。

实现过程为将两个多项式的每个项都互相乘在一起，然后将该新生节点插在pnew里，事先ListInsert编写时实现的功能是将新节点按照顺序插入，指数相同的合在一起。所以完成后pnew指向的是已经排好序的多项式。最后删除La,Lb。防止内存泄露。

```
/*多项式的乘法*/
Status MultiplyPoly(polynomial *La, polynomial *Lb)
{
    polynomial data;
    polynomial pnew = NULL;
    polynomial pa = *La, pb = *Lb;
    InitList(&pnew);
    if (Node_count(*La) == 0 || Node_count(*Lb) == 0)
        return ERROR;
    while (pa->next)
    {
        while (pb->next)
        {
            data = (polynomial)malloc(sizeof(polynode));
            data->coef = GFmulti(pa->next->coef, pb->next->coef);
            data->expn = pa->next->expn + pb->next->expn;
            ListInsert(&pnew, data);
            pb = pb->next;
        }
        pb = *Lb;
        pa = pa->next;
    }
    DestroyList(La);
    DestroyList(Lb);
    *La = pnew;
    return OK;
}
```

4.调试过程碰到的问题

4.1.粗心大意导致的错误

在动态内存申请的时候用了malloc(n),在这里应该区分以下new 和malloc的使用区别，

New后面跟着的是[申请类型的个数]，malloc后面跟着的是（申请的内存字节数），所以后面还需乘以sizeof(类型),因为这样的一个小错误调试了一下午，浪费时间精力，引以为戒。

装 订 线

5. 心得体会

这次的大作业难度较大，难点在于需要学习大量的新知识，而作为一个程序工程师，这个职业的特征正是如此，需要不断的学习新知识，补充能量。这样的大作业可以让我们适应未来的工作，培养快速学习的能力，养成终身学习的习惯，会在未来受益匪浅。

这次大作业期间还有个小插曲，我丢失了我的电脑（哭），写到一半的大作业也随之而去。影响了我大作业的进度，所以这次提交的这个大作业是这四天集中写成的一个简化版本，只按照要求（100个字符以内）支持version1-5这5个版本，未避免块的分类等，全部采用了纠错级别为L的模式，后面的掩码也是采用了一种，没有挑选得分最低的过程，但是优点在于，所有的核心过程都是自己实现的，没有采用网上现成的函数库（要不然就不会这么low了。。。），后来想找个掩码的现成程序，发现接口已经对不上了。只好手算一个。

6. 附件：源程序

```
#include "90-b4.h"
#include "cmd_console_tools.h"
//****************************************
*****************

/*以下为迦洛华域运算部分函数*/

//****************************************
*****************
unsigned int GFadd(unsigned int a, unsigned int
b)
{
    unsigned int c = alphato(a)^ alphato(b);
    return powerof(c);
}
unsigned int GFmulti(unsigned int a, unsigned
int b)
{
    unsigned int c = (a+b)%255;
    return c;
}
unsigned int alphato(unsigned int n)
{
    int N = n % 255;
    return AlphaTo[N];
}
unsigned int powerof(unsigned int n)
{
    return PowerOf[n - 1];
}
//****************************************
*****************
```

```
/*以下为多项式操作部分函数*/

//****************************************
*****************


/* 初始化线性表 */
Status InitList(polynomial *L)
{
    /* 申请头结点空间，赋值给头指针 */
    *L =
(polynomial)malloc(sizeof(polynode));
    if (*L == NULL)
        exit(LOVERFLOW);
    (*L)->next = NULL;
    return OK;
}
/* 删除线性表 */
Status DestroyList(polynomial *L)
{
    polynomial q, p = *L; //指向首元

                                /* 整个链表(含
头结点)依次释放 */
    while (p) {    //若链表为空，则循环不执行
        q = p->next; //抓住链表的下一个结点
        free(p);
        p = q;
    }
    *L = NULL;    //头指针置NULL
    return OK;
}
/*删除指定的节点*/
```

```
Status delete_node(polynomial L, polynomial node)
{
    polynomial temp = NULL;
    if (L == NULL)
        return ERROR;
    while (L->next)
    {
        if ((node->coef == L->coef) && (node->expn == L->expn))
        {
            temp = L->next;
            L->next = L->next->next;
            free(temp);
            return OK;
        }
        L = L->next;
    }
    return ERROR;
}
/*求链表元素个数*/
int Node_count(polynomial L)
{
    int i = 0;
    if (L->next == NULL)
        return 0;
    while (L->next)
    {
        i++;
        L = L->next;
    }
    return i;
}
/* 按指数从小到大排序在指定的位置插入一个新元素 */
Status ListInsert(polynomial *L, polynomial e)
{
    polynomial  p = *L;     //p指向头结点

                            /*找到适合放e->expn的位置*/
    while (p->next) {
        if (p->next->expn < e->expn)
        {
            e->next = p->next;
            p->next = e;
            break;
        }
        if (p->next->expn == e->expn)
        {

    p->next->coef=GFadd(p->next->coef, e->coef);
            break;
        }
        p = p->next;
    }
    if (p->next == NULL)
    {
        e->next = p->next;
        p->next = e;
    }

    return OK;
}
/*多项式的乘法*/
Status MultiplyPoly(polynomial *La, polynomial *Lb)
{
    polynomial data;
    polynomial pnew = NULL;
    polynomial pa = *La, pb = *Lb;
    InitList(&pnew);
   if (Node_count(*La) == 0 || Node_count(*Lb) == 0)
        return ERROR;
    while (pa->next)
    {
        while (pb->next)
        {
            data = (polynomial)malloc(sizeof(polynode));
            data->coef = GFmulti(pa->next->coef, pb->next->coef);
            data->expn = pa->next->expn + pb->next->expn;
            ListInsert(&pnew, data);
            pb = pb->next;
        }
        pb = *Lb;
        pa = pa->next;
    }
    DestroyList(La);
    DestroyList(Lb);
    *La = pnew;
    return OK;
}
/*多项式归并操作，La,Lb归并入La，并入操作为^*/
Status ListUnion(polynomial *La, polynomial Lb)
{

    polynomial p = (*La)->next, q = Lb->next, pre = *La, temp;
    int flag = 0;
    while (p&&q)
    {



        if (p->expn == q->expn)
        {
            if ((alphato(p->coef) ^
```

```
alphato(q->coef)) != 0||flag==1)
                {
                    p->coef =
powerof(alphato(p->coef) ^ alphato(q->coef));
                    pre->next = p;
                    pre = p;
                    temp = q;
                    q = q->next;
                    p = p->next;

                }
                else
                {
                    temp = q;
                    q = q->next;
                    p = p->next;
                    free(temp);
                    flag = 1;
                }
            }

    }
    pre->next = p ? p : q;
    return OK;
}
//遍历线性表
Status ListTraverse(polynomial L)
{

    polynomial p = L->next;


    while (p)
    {

    printf("%u*x^(%u)+",alphato( p->coef),
p->expn);
        p = p->next;
    }
    printf("\n");
    if (p)
        return ERROR;

    printf("\n");//最后打印一个换行,只是为了
好看，与算法无关
    return OK;
}
//生成多项式。。。
polynomial genaretor(int version)
{
    polynomial La, Lb, e;
    InitList(&La);
    switch (version)
    {
        case 17:
        {
            InitList(&La);
            for (int i = 0; i < 2; i++)
            {
                e =
(polynomial)malloc(sizeof(polynode));
                e->coef = 0;
                e->expn = i;
                ListInsert(&La, e);
            }

            for (int i = 0; i < 6; i++)
            {
                InitList(&Lb);
                    e =
(polynomial)malloc(sizeof(polynode));
                    e->coef = 0;
                    e->expn = 1;
                    ListInsert(&Lb, e);
                    e =
(polynomial)malloc(sizeof(polynode));
                    e->coef = i+1;
                    e->expn = 0;
                    ListInsert(&Lb, e);
                    MultiplyPoly(&La,
&Lb);
            }
            return La;
        }
        case 32:
        {
            InitList(&La);
            for (int i = 0; i < 2; i++)
            {
                e =
(polynomial)malloc(sizeof(polynode));
                e->coef = 0;
                e->expn = i;
                ListInsert(&La, e);
            }

            for (int i = 0; i < 9; i++)
            {
                InitList(&Lb);
                e =
(polynomial)malloc(sizeof(polynode));
                e->coef = 0;
                e->expn = 1;
                ListInsert(&Lb, e);
                e =
(polynomial)malloc(sizeof(polynode));
                e->coef = i + 1;
                e->expn = 0;
                ListInsert(&Lb, e);
                MultiplyPoly(&La, &Lb);
            }
            return La;
        }
        case 53:
```

装 订 线

```
        {
            InitList(&La);
            for (int i = 0; i < 2; i++)
            {
                e =
(polynomial)malloc(sizeof(polynode));
                e->coef = 0;
                e->expn = i;
                ListInsert(&La, e);
            }

            for (int i = 0; i < 14; i++)
            {
                InitList(&Lb);
                e =
(polynomial)malloc(sizeof(polynode));
                e->coef = 0;
                e->expn = 1;
                ListInsert(&Lb, e);
                e =
(polynomial)malloc(sizeof(polynode));
                e->coef = i + 1;
                e->expn = 0;
                ListInsert(&Lb, e);
                MultiplyPoly(&La, &Lb);
            }
            return La;
        }
        case 78:
        {
            InitList(&La);
            for (int i = 0; i < 2; i++)
            {
                e =
(polynomial)malloc(sizeof(polynode));
                e->coef = 0;
                e->expn = i;
                ListInsert(&La, e);
            }

            for (int i = 0; i < 19; i++)
            {
                InitList(&Lb);
                e =
(polynomial)malloc(sizeof(polynode));
                e->coef = 0;
                e->expn = 1;
                ListInsert(&Lb, e);
                e =
(polynomial)malloc(sizeof(polynode));
                e->coef = i + 1;
                e->expn = 0;
                ListInsert(&Lb, e);
                MultiplyPoly(&La, &Lb);
            }
            return La;
        }
        case 106:
        {
            InitList(&La);
            for (int i = 0; i < 2; i++)
            {
                e =
(polynomial)malloc(sizeof(polynode));
                e->coef = 0;
                e->expn = i;
                ListInsert(&La, e);
            }

            for (int i = 0; i < 25; i++)
            {
                InitList(&Lb);
                e =
(polynomial)malloc(sizeof(polynode));
                e->coef = 0;
                e->expn = 1;
                ListInsert(&Lb, e);
                e =
(polynomial)malloc(sizeof(polynode));
                e->coef = i + 1;
                e->expn = 0;
                ListInsert(&Lb, e);
                MultiplyPoly(&La, &Lb);
            }
            return La;
        }

    }
    return La;
}
//消息多项式。。。size 是多项式的系数的个数
polynomial messege(int version, unsigned int
data[],int size)
{
    polynomial L,e;
    InitList(&L);
    switch (version)
    {
        case 17:
            for (int i = 0; i < size; i++)
            {

                e =
(polynomial)malloc(sizeof(polynode));
                e->coef = powerof(data[i]);
                e->expn = size-i-1+7;
                ListInsert(&L, e);
            }
            break;
        case 32:
            for (int i = 0; i < size; i++)
            {

                e =
```

```
(polynomial)malloc(sizeof(polynode));
                    e->coef = powerof(data[i]);
                    e->expn = size - i - 1 + 10;
                    ListInsert(&L, e);
               }
               break;
          case 53:
               for (int i = 0; i < size; i++)
               {

                    e =
(polynomial)malloc(sizeof(polynode));
                    e->coef = powerof(data[i]);
                    e->expn = size - i - 1 + 15;
                    ListInsert(&L, e);
               }
               break;
          case 78:
               for (int i = 0; i < size; i++)
               {

                    e =
(polynomial)malloc(sizeof(polynode));
                    e->coef = powerof(data[i]);
                    e->expn = size - i - 1 + 20;
                    ListInsert(&L, e);
               }
               break;
          case 106:
               for (int i = 0; i < size; i++)
               {

                    e =
(polynomial)malloc(sizeof(polynode));
                    e->coef = powerof(data[i]);
                    e->expn = size - i - 1 + 26;
                    ListInsert(&L, e);
               }
               break;
          case 14:
               for (int i = 0; i < size; i++)
               {

                    e =
(polynomial)malloc(sizeof(polynode));
                    e->coef = powerof(data[i]);
                    e->expn = size - i - 1 + 10;
                    ListInsert(&L, e);
               }
               break;
     }
     return L;
}
//求纠错码,size是信息多项式的个数
polynomial correct(polynomial Lme,int size,int
version)
{
     unsigned int r;
     for (int i = 0; i < size; i++)
     {
          polynomial pm = Lme->next,Lge=
genaretor(version),pg = Lge->next;
          r = pm->expn - pg->expn;
          while (pg)
          {
               pg->expn += r;
               pg->coef = GFmulti(pg->coef,
pm->coef);
               pg = pg->next;
          }

          ListUnion(&Lme, Lge);

     }

     polynomial pm = Lme->next;
     return Lme;
}
//将纠错多项式转化为纠错数组
unsigned int *trans(polynomial L)
{
     unsigned int *data;
     polynomial p = L->next;
     data=(unsigned int *)
malloc(Node_count(L)*sizeof(unsigned int));
     for (int i = 0; i < Node_count(L); i++)
     {
          data[i] = alphato(p->coef);
          p = p->next;
     }
     return data;
}

//****************************************
****************

/*以下为信息与二进制流的转化部分函数*/

//****************************************
****************
//释放创建的信息流
void BitStream_free(BitStream *bstream)
{
     if (bstream != NULL) {
          free(bstream->data);
          free(bstream);
     }
}
//创建新的新的比特流
BitStream *BitStream_new(void)
{
     BitStream *bstream;

     bstream = (BitStream
```

```
*)malloc(sizeof(BitStream));
    if (bstream == NULL) return NULL;

    bstream->length = 0;
    bstream->data = NULL;

    return bstream;
}

//为申请的比特流创建空间
int BitStream_allocate(BitStream *bstream, int
version)
{
    unsigned char *cdata;

    if (bstream == NULL) {
        return -1;
    }

    cdata = (unsigned char
*)malloc(int((version + 2) * 8));
    if (cdata == NULL) {
        return -1;
    }

    if (bstream->data) {
        free(bstream->data);
    }
    bstream->length = (version + 2) * 8;
    bstream->data = cdata;

    return 0;
}
//将输入信息转化为比特流,包括选择的模式，信息
长度，结束符
BitStream *BitStream_newFromBytes(int version,
unsigned int size, unsigned char *data)
{
    unsigned char mask;
    int i, j;
    unsigned char *p, mode[4] = { 0,1,0,0 },
app1[8] = { 1,1,1,0,1,1,0,0 }, app2[8] =
{ 0,0,0,1,0,0,0,1 };
    BitStream *bstream;

    bstream = BitStream_new();
    if (bstream == NULL) return NULL;

    if (BitStream_allocate(bstream, version))
{
        BitStream_free(bstream);
        return NULL;
    }
    p = bstream->data;
    for (i = 0; i < 4; i++)
    {
        *p = mode[i];
        p++;
    }
    mask = 0x80;
    for (i = 0; i < 8; i++)
    {
        if (char(size) & mask) {
            *p = 1;
        }
        else {
            *p = 0;
        }
        p++;
        mask = mask >> 1;
    }
    for (i = 0; (unsigned int)i < size; i++) {
        mask = 0x80;
        for (j = 0; j < 8; j++) {
            if (data[i] & mask) {
                *p = 1;
            }
            else {
                *p = 0;
            }
            p++;
            mask = mask >> 1;
        }
    }
    if (8 * (version + 2) - 8 * size + 12  >0)
    {
        int stop = (8 * (version + 2) - 8 * size
+ 12)<4 ? (8 * (version + 2) - 8 * size + 12) :
4;
        for (i = 0; i < stop; i++)
        {
            *p = 0;
            p++;
        }
    }
    if (8 * (version + 2) - 8 * size + 12 > 4)
    {
        int stop = version - size;
        while (stop > 0)
        {
            for (i = 0; i < 8; i++)
            {
                *p = app1[i];
                p++;
            }
            stop--;
            if (stop > 0)
            {
                for (i = 0; i < 8; i++)
                {
                    *p = app2[i];
                    p++;
                }
                stop--;
```

```
                }
            }
        }
        return bstream;
}
//将比特流转化成消息多项式系数
unsigned int *BitStream_toByte(BitStream *
&bstream)
{
        int i, j, size, bytes;
        unsigned char *p, v;
        unsigned int *data;

        size = bstream->length;
        if (size == 0) {
                return NULL;
        }
        data = (unsigned int *)malloc(int((size +
7) / 8 * sizeof(unsigned int)));
        if (data == NULL) {
                return NULL;
        }
        bytes = int((size + 7) / 8);

        p = bstream->data;
        for (i = 0; i < bytes; i++) {
                v = 0;
                for (j = 0; j < 8; j++) {
                        v = v << 1;
                        v |= *p;
                        p++;
                }
                data[i] = unsigned int(v);
        }
        return data;
}
//将纠错多项式系数转化为比特流
BitStream *BitStream_tobit(polynomial L)
{
        unsigned char mask, *p;
        int i, j;
        BitStream *bstream;
        unsigned int *cdata;
        cdata = trans(L);
        bstream = BitStream_new();
        if (bstream == NULL) return NULL;

        if (BitStream_allocate(bstream,
Node_count(L) - 2)) {
                BitStream_free(bstream);
                return NULL;
        }

        p = bstream->data;
        for (i = 0; i < Node_count(L); i++) {
                mask = 0x80;
                for (j = 0; j < 8; j++) {
                        if (cdata[i] & mask) {
                                *p = 1;
                        }
                        else {
                                *p = 0;
                        }
                        p++;
                        mask = mask >> 1;
                }
        }
        return bstream;
}
//获取输入的信息的长度
unsigned int strlength(unsigned char *mess)
{
        unsigned int len = 0;
        while (*mess != '\0')
        {
                len++;
                mess++;
        }
        return len;
}
//根据输入的信息选择合适的version
int choose_version(unsigned int size)
{
        if (size > 0 && size <= 17)
                return L_1;
        else if (size > 17 && size <= 32)
                return L_2;
        else if (size > 32 && size <= 53)
                return L_3;
        else if (size > 53 && size <= 78)
                return L_4;
        else if (size > 78 && size <= 106)
                return L_5;
        else
                return -1;
}
//将信息多项式和纠错多项式的比特流合在一起
unsigned char *BitStream_Union(BitStream
*bstream, BitStream *obstream, int version)
{
        unsigned char *data, *p;
        if (version == 17)
        {
                data = (unsigned
char*)malloc(bstream->length +
obstream->length);
                p = data;
                memcpy(p, bstream->data,
bstream->length);
                p = p + bstream->length;
                memcpy(p, obstream->data,
obstream->length);

        }
```

装 订 线

```
    else
    {
        data = (unsigned
char*)malloc(bstream->length +
obstream->length + 7);
        p = data;
        memcpy(p, bstream->data,
bstream->length);
        p = p + bstream->length;
        memcpy(p, obstream->data,
obstream->length);
        char *zero;
        zero = (char *)malloc(7);

        for (int i = 0; i < 7; i++)
        {
            zero[i] = 0;
        }
        p = p + obstream->length;
        memcpy(p, zero, 7);

    }
    return data;
}


//**************************************
*****************

/*以下为0-1矩阵操作部分函数*/

//**************************************
*****************

//0->white 1->black
int ver_to_size(int version)
{
    int v;
    switch (version)
    {
        case 17:
            v = 1;
            break;
        case 32:
            v = 2;
            break;
        case 53:
            v = 3;
            break;
        case 78:
            v = 4;
            break;
        case 106:
            v = 5;
            break;
    }
    return v;
}
QRarray **newarray(int v)
{
    QRarray **array;
    array = new QRarray*[(v - 1) * 4 + 21];
    for (int i = 0; i < (v - 1) * 4 + 21; i++)
    {
        array[i] = new QRarray[(v - 1) * 4 +
21];
    }
    return array;
}
/*在x,y位置处添加一个 Finder Pattern*/
void Finder_pattern(int x, int y, QRarray
**array, int v)
{
    array[x][y].col = 1;
    array[x][y].ocu = 1;
    for (int i = 1; i <= 4; i++)
    {
        for (int j = -i; j <= i; j++)
        {
            if (x + i < (v - 1) * 4 + 21 &&
y + j >= 0 && y + j<(v - 1) * 4 + 21)
            {
                array[x + i][y + j].col = i %
2;
                array[x + i][y + j].ocu = 1;
            }
            if (x - i >= 0 && y + j >= 0 &&
y + j<(v - 1) * 4 + 21)
            {
                array[x - i][y + j].col = i %
2;
                array[x - i][y + j].ocu = 1;
            }
            if (y + i < (v - 1) * 4 + 21 &&
x + j >= 0 && x + j<(v - 1) * 4 + 21)
            {
                array[x + j][y + i].col = i %
2;
                array[x + j][y + i].ocu = 1;
            }
            if (y - i >= 0 && x + j >= 0 &&
x + j<(v - 1) * 4 + 21)
            {
                array[x + j][y - i].col = i %
2;
                array[x + j][y - i].ocu = 1;
            }
        }
    }
}
/*在x,y位置处添加一个Alignment Patterns*/
void Alignment_Patterns(int x, int y, QRarray
**array)
{
```

装 订 线

```
    array[x][y].col = 1;
    array[x][y].ocu = 1;
    for (int i = 1; i <= 2; i++)
    {
        for (int j = -i; j <= i; j++)
        {
            array[x + i][y + j].col = (i + 1) %
2;
            array[x + i][y + j].ocu = 1;
            array[x - i][y + j].col = (i + 1) %
2;
            array[x - i][y + j].ocu = 1;
            array[x + j][y + i].col = (i + 1) %
2;
            array[x + j][y + i].ocu = 1;
            array[x + j][y - i].col = (i + 1) %
2;
            array[x + j][y - i].ocu = 1;
        }
    }
}
/*在x,y位置添加一个Timing Pattern 和dark
module*/
void Timing_Patterns(QRarray **array, int v)
{
    array[(v - 1) * 4 + 21 - 8][8].col = 1;
    array[(v - 1) * 4 + 21 - 8][8].ocu = 1;
    for (int i = 0; i < (v - 1) * 4 + 21; i++)
    {
        if (array[6][i].ocu == 0)
        {
            if (i % 2 == 0)
                array[6][i].col = 1;
            else
                array[6][i].col = 0;
            array[6][i].ocu = 1;
        }
        if (array[i][6].ocu == 0)
        {
            if (i % 2 == 0)
                array[i][6].col = 1;
            else
                array[i][6].col = 0;
            array[i][6].ocu = 1;
        }
    }
}
/*版本信息放置占位*/
void version_patterns(QRarray **array, int v)
{
    for (int i = 0; i < 9; i++)
        array[i][8].ocu = 1;
    for (int i = 0; i < 9; i++)
        array[8][i].ocu = 1;
    for (int i = (v - 1) * 4 + 21 - 1; i >= (v
- 1) * 4 + 21 - 8; i--)
        array[8][i].ocu = 1;
    for (int i = (v - 1) * 4 + 21 - 1; i >= (v
- 1) * 4 + 21 - 7; i--)
        array[i][8].ocu = 1;
}
/*数据码放置*/
void data_patterns(QRarray **array, int v,
unsigned char data[])
{

    int len = (v - 1) * 4 + 21 - 1, count = 0;
    for (int j = len; j >= 0; j -= 2)
    {
        if (j == 6)
        {
            j++;
            continue;
        }
        if (count % 2 == 0)
        {
            for (int i = len; i >= 0; i--)
            {
                for (int k = j; k >= j - 1;
k--)
                {
                    if (array[i][k].ocu ==
0)
                    {
                        array[i][k].col =
int(*data);
                        data++;
                    }
                    else
                        continue;
                }
            }
        }
        else
        {
            for (int i = 0; i <= len; i++)
            {
                for (int k = j; k >= j - 1;
k--)
                {
                    if (array[i][k].ocu ==
0)
                    {
                        array[i][k].col =
int(*data);
                        data++;
                    }
                    else
                        continue;
                }
            }
        }
        count++;
    }
```

```
}
/*输出二维码图形*/
void output(HANDLE hout, QRarray **array, int
v)
{
    system("cls");
    setconsoleborder(hout, 100, 50);
    showstr(hout, 0, 0, "  ", COLOR_HWHITE,
0);
    showstr(hout, 2 * ((v - 1) * 4 + 20) + 4,
0, "  ", COLOR_HWHITE, 0);
    showstr(hout, 0, 1, "  ", COLOR_HWHITE,
0);
    showstr(hout, 2 * ((v - 1) * 4 + 20) + 4,
1, "  ", COLOR_HWHITE, 0);
    for (int i = 0; i < (v - 1) * 4 + 21; i++)
    {
        for (int j = 0; j < (v - 1) * 4 + 21;
j++)
        {
            gotoxy(hout, j, i);
            if (array[i][j].col == 0)
                showstr(hout, 2 * j + 2, i +
1, "  ", COLOR_HWHITE, 0);
            else
                showstr(hout, 2 * j + 2, 1 +
i, "  ", COLOR_BLACK, 0);
            showstr(hout, 2 * j + 2, 0, "  ",
COLOR_HWHITE, 0);
            showstr(hout, 2 * j + 2, (v - 1)
* 4 + 21 + 1, "  ", COLOR_HWHITE, 0);
        }
        showstr(hout, 0, i + 2, "  ",
COLOR_HWHITE, 0);
        showstr(hout, 2 * ((v - 1) * 4 + 20)
+ 4, i + 2, "  ", COLOR_HWHITE, 0);

    }
}
//创建基本二维码矩阵
QRarray **QRcode(int version)
{
    QRarray **array;
    int v;
    v = ver_to_size(version);
    array = newarray(v);
    Finder_pattern(3, 3, array, v);
   Finder_pattern(3, (v - 1) * 4 + 21 - 4, array,
v);
   Finder_pattern((v - 1) * 4 + 21 - 4, 3, array,
v);
    switch (v)
    {
        case 1:
            break;
        case 2:
            Alignment_Patterns(18, 18,
array);
            break;
        case 3:
            Alignment_Patterns(22, 22,
array);
            break;
        case 4:
            Alignment_Patterns(26, 26,
array);
            break;
        case 5:
            Alignment_Patterns(30, 30,
array);
            break;
    }
    Timing_Patterns(array, v);
    version_patterns(array, v);



    return array;
}
//掩模
/*int mark_mask(QRarray **array, int mode, int
v)
{
int mark = 0;
switch (mode)
{
case 0:
for (int i = 0; i < (v - 1) * 4 + 21; i++)
for (int j = 0; j < (v - 1) * 4 + 21; j++)
{
if ((i + j) % 2 == 0 && array[i][j].ocu == 0)
{
if (array[i][j].col == 0)
array[i][j].col = 1;
else
array[i][j].col = 0;
}
}

break;
case 1:
for (int i = 0; i < (v - 1) * 4 + 21; i++)
for (int j = 0; j < (v - 1) * 4 + 21; j++)
{
if (i % 2 == 0 && array[i][j].ocu == 0)
{
if (array[i][j].col == 0)
array[i][j].col = 1;
else
array[i][j].col = 0;
}
}

break;
```

装 订 线

```
case 3:
for (int i = 0; i < (v - 1) * 4 + 21; i++)
for (int j = 0; j < (v - 1) * 4 + 21; j++)
{
if (j % 3 == 0 && array[i][j].ocu == 0)
{
if (array[i][j].col == 0)
array[i][j].col = 1;
else
array[i][j].col = 0;
}
}

break;

}

}
unsigned char* mask(QRarray **array,int v)
{
for (int i = 0; i < (v - 1) * 4 + 21; i++)
for (int j = 0; j < (v - 1) * 4 + 21; j++)
{
if ((i + j) % 2 == 0)
{
if (array[i][j].col == 0)
array[i][j].col = 1;
else
array[i][j].col = 0;
}
}
}*/
//掩模纠错码
//填充纠错级别，掩模信息码
QRarray **cor_mask_bit(unsigned char *data,
QRarray **array, int v)
{
    for (int i = 0; i < 6; i++)
        array[i][8].col = (unsigned
int)(data[i]);
    for (int i = 0; i < 8; i++)
        array[8][(v - 1) * 4 + 21 - 1 - i].col
= unsigned int(data[i]);
    array[7][8].col = (unsigned int)data[6];
    array[8][8].col = (unsigned int)data[7];
    array[8][7].col = (unsigned int)data[8];
    for (int i = 0; i < 6; i++)
        array[8][i].col = (unsigned
int)data[14 - i];
    for (int i = 0; i < 7; i++)
        array[(v - 1) * 4 + 21 - 1 - i][8].col
= (unsigned int)data[14 - i];
    return array;
}
#include "90-b4.h"
#include "cmd_console_tools.h"
int main()
{

    const HANDLE hout =
GetStdHandle(STD_OUTPUT_HANDLE);
    QRarray **array;

    array = QRcode(17);
    /*输入信息，构建信息比特流*/
    unsigned char *bit_mess;
    BitStream *bstream,*obstream;
    cout << "输入要转换为二维码的信息（100
字符以内，纠错级别L)：";
    unsigned char mess[100],*p=mess;
    cin >> mess;
    int version;
    version =
choose_version(strlength(mess));
    int v = ver_to_size( version);
   bstream=BitStream_newFromBytes(version,
strlength(mess),mess);//转化为01比特流
    unsigned int *mess_data;
    mess_data = BitStream_toByte(bstream);
    polynomial La;
    InitList(&La);
    La =
messege(version,mess_data,version+2);
    ListTraverse(La);
    La = correct(La, version+2, version);
    ListTraverse(La);
    obstream = BitStream_tobit(La);
    bit_mess = BitStream_Union(bstream,
obstream, version);
    array = QRcode(version);
    data_patterns(array,
ver_to_size(version),bit_mess);
    unsigned char mask_data[] =
{ 1,1,1,1,0,1,0,0,0,1,1,0,0,1,1 };
    for (int i = 0; i < (v - 1) * 4 + 21; i++)
        for (int j = 0; j < (v - 1) * 4 + 21;
j++)
        {
            if ((i/2 + j/3) % 2 == 0 &&
array[i][j].ocu == 0)
            {
                if (array[i][j].col == 0)
                    array[i][j].col = 1;
                else
                    array[i][j].col = 0;
            }
        }
    cor_mask_bit(mask_data, array, v);
    output(hout, array,
ver_to_size(version));
```

```
    return 0;
}
```

装

订

线