装 订 线

实验报告

报告名称：二维码的实现

班级：计算机1班

学号：1651574

姓名：贾昊霖

完成日期：2018年5月27日

1. 题目及基本要求

1.1.题目

二维码的实现

1.2.基本要求

从键盘输入内容，生成二维码，二维码要支持多种字符的文本和网址；生成的二维码以任意扫码工具识别出为准；

所有的显示在cmd界面下实现。

2. 整体设计思路

本次作业最主要的问题在于看懂qrcode的原理，其次在于根据原理写代码。基本思路就是根据原理一步步完成，选择其中一部分进行实现。

基本的输入输出很简单，输入一段字符串，按照算出的最终结果输出二维码伪图形界面，没有多余的内容。

首先根据字符串长度选择二维码版本，为了处理简便，选择了1-5版本当中的一些。这里一般我选择了utf-8，使用L级别的纠错码。然后用自己编写的polynomial类算出纠错码。之后将编码按照一定次序放在二维数组里面算出二维码bit位所对应的坐标。然后加入掩码和固定模块信息。

我用类设计本次程序

QRcode类：

```
class QRcode
{
private:
    char origin_content[105];
    char mode_indicator[8];
    char count_indicator[15];
    char data_bit_stream[1050];
    char data_encoding[1000];
    char data_final[1000];

    char data_correct_code[2000];
    int len_origin;
    int len_data_encoding;
    int len_final;
    int version;
    int mask_version;

    int cost[8][4];

    char correct_str[20] = "111101000110011";
```

```
public:
    QRcode(char* = NULL);
    ~QRcode();

    int **pattern;
    bool **flag;

    void Solve();
    void DecideVersion();
    void CountIndicator();
    void GetDataStream();
    void ConcatenateData();
    void AddTerminator();
    void AddPadding();
    void CorrectError(Polynomial &);
    void InitialPattern();
    void AddFunctionPattern();
    void PositionDetectPattern(int, int);
    void SeperatorPattern();
    void PrintPattern();
    void DrawPattern();
    void AlignmentPattern();
    void TimingPattern();
    void FillingFormatInfo();
    void FillingData(Polynomial &);

    void MaskingPattern();

    int Str2Int(char*);
    int Str2Int_binary(char*);
    void Int2Int_binary(int, char *, int = 0);

    void print();
};
```

polynomial类：

```
class Polynomial {
public:
    int head_coe;
    PolyList next;
    Polynomial(PolyList = NULL) { next = NULL; }
    void Append(PolyNode &);
    void Append(PolyList &);
    void GetHeadCoe();
    void MulPoly(int n);
    void DivPoly(int n);
    void XorPoly(const Polynomial &);
    void ConvertAlpha2N();
    void ConvertN2Alpha();
    void PrintPoly();
};
```

以及polyNode类：

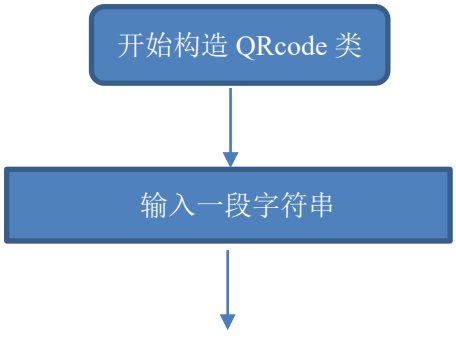
```
class PolyNode {
public:
    int coe;     //coefficient
    int exp;     //exponent
    PolyNode* next;
    PolyNode(int = 0, int = 0, PolyNode* = NULL);
};
```

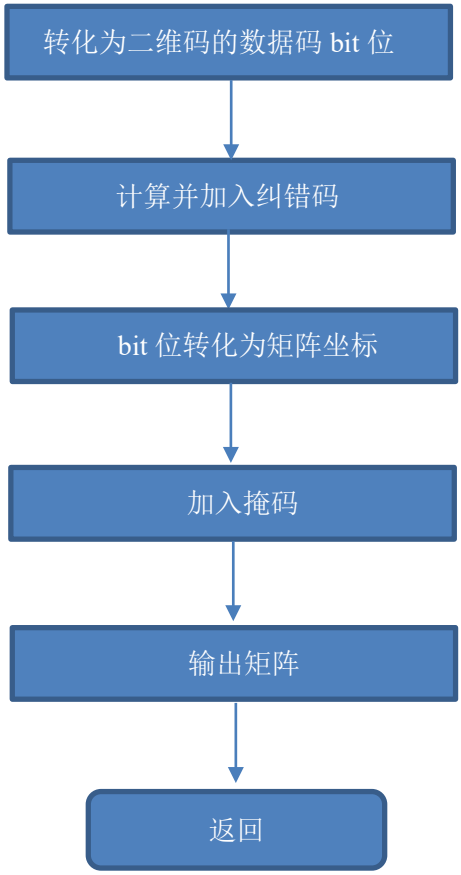
为了简化操作，一些可以打表的地方通过打表计算，比如格式信息就完全根据固定的数组决定加入的bit位值。

3. 主要功能的实现

使用QRcode类。这里就是二维码生成的大致流程。

该流程大致如下：





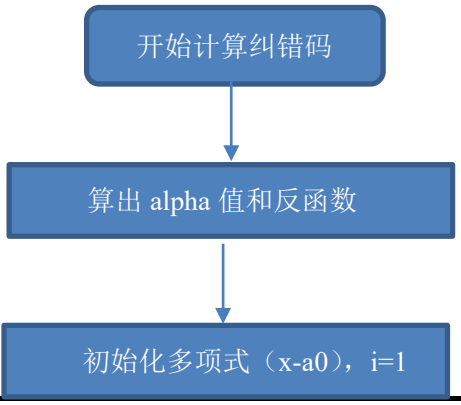
装

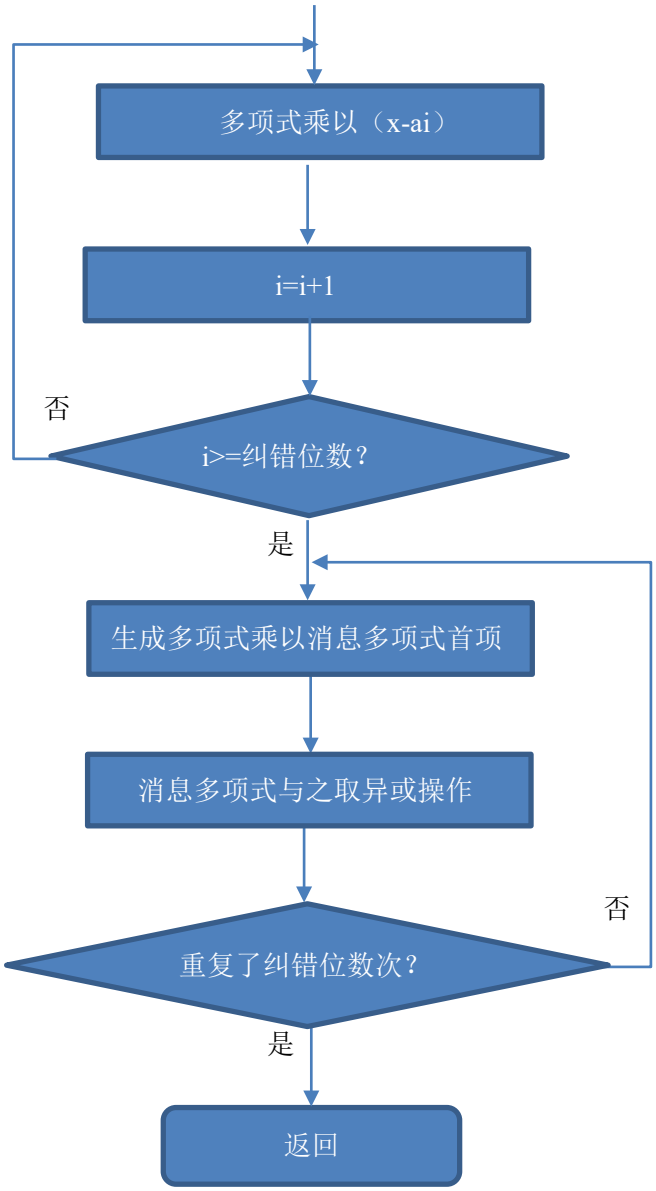
订

线

在本次调度程序当中的内容非常简单，没有任何循环部分。

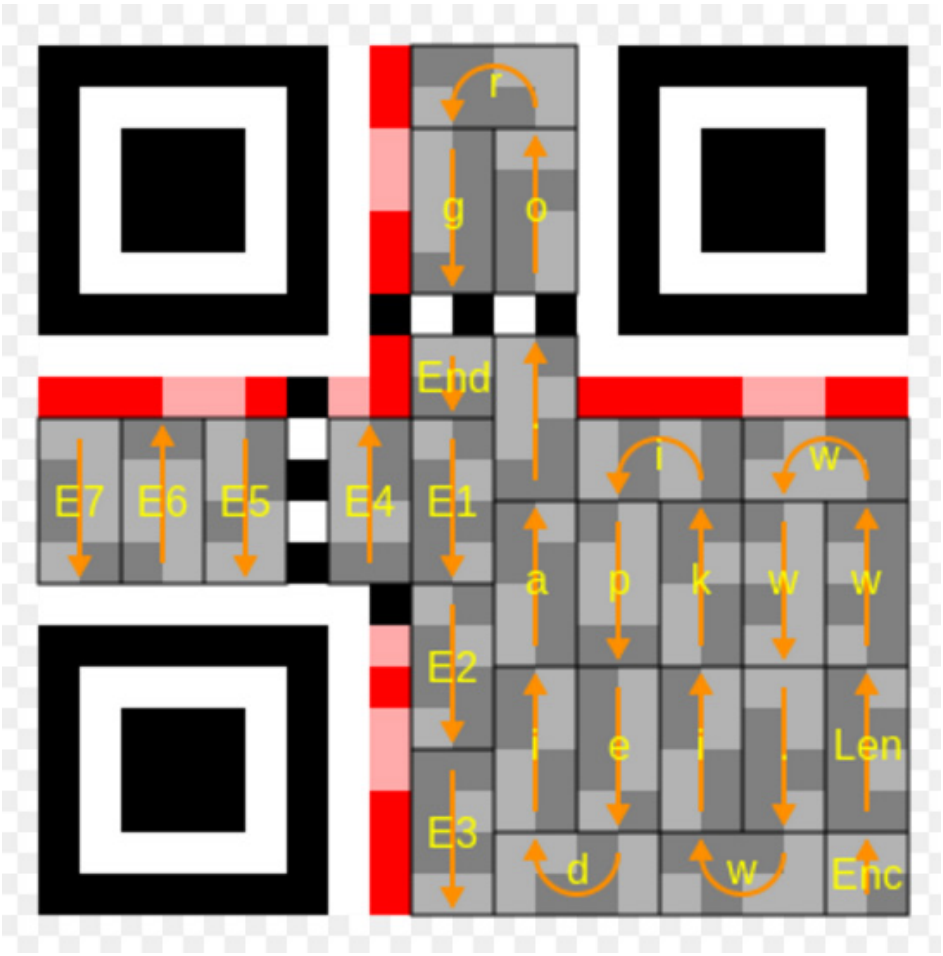
具体实现当中,自己写的polynomial类中两个工具成员函数最重要,其中一个是计算纠错码函数,基本过程如下:





装 订 线

另一个部分为字符数组当中的bit位转化为矩阵坐标表示,大致方法如下:



(截取Wikipedia进行说明)

从右下角开始,以两列为一组，先检查右侧的bit位是否可以存储数据，如果可以，就放置一个bit；然后检查左侧的bit位，如果可以就放置一个bit。

然后，向上或者向下移动一行，继续做以上的判定。这里移动的方向由所在列决定。如果到了顶部或者底部，就向左移动两列继续。如果遇到Timing Patterns所在列，就再移动一列。

所要存储的数据内容后全部置0，使得最后补充几位0的情况不用特殊讨论。

以上为一部分重要过程。

本次大作业代码全部自己完成，没有利用任何他人代码，全部用类封装

4. 调试过程碰到的问题

在调试过程中，由于二维码不容易发现错误，一般仅仅根据扫出来的结果判定是否有错误，碰到的问题一般很难找到出在哪里。

其中自己共编写三个类

装 订 线

```
class PolyNode { ... };
class Polynomial { ... };
class QRcode { ... };
```

分别作用为：生成多项式链表类、多项式操作类、以及二维码类

其中自己写polynomial类时 算法的检查很不容易，并且学长的ppt有误!!

Ⅱ.生成纠错码

⑤ ：对应项系数做异或

Step 3. 多项式除法求余

$$M'(x) = 0x^{25} + 176x^{24} + 144x^{23} + 197x^{22} + 126x^{21} + 161x^{20} + 216x^{19} + 18x^{18} + 236x^{17} + 17x^{16} + 236x^{15} + 17x^{14} + 236x^{13} + 17x^{12} + 236x^{11} + 17x^{10} + 236x^{9} + 17x^{8} + 236x^{7}$$

$$\alpha^4 x^{18}\, G_7(x) = 16x^{25} + 163x^{24} + 243x^{23} + 85x^{22} + 146x^{21} + 176x^{20} + 52x^{19} + 3x^{18} + 0x^{17} + 0x^{16} + \cdots$$

这个系数给错了...

导致浪费了大量的时间....

其中编写了这几个函数用来检验中间过程的正误，十分方便：

```
void Polynomial::PrintPoly()
{
    PolyList p = this->next;
    while (p) {
        cout << "coeffcient:" << p->coe << endl;
        cout << "exponent:" << p->exp << endl;
        p = p->next;
    }
}
```

装 订 线

```
void QRcode::PrintPattern()
{
    for (int i = 0; i < pattern_size[version]; i++) {
        for (int j = 0; j < pattern_size[version]; j++) {
            if (pattern[j][i] == BLACK)
                cout << BLACK;
            else if (pattern[j][i] == WHITE)
                cout << WHITE;
            else
                cout << '.';
            }
        putchar('\n');
    }
}
```

```
void QRcode::print()
{
    setcolor(COLOR_BLACK);
    cout << "version:" << this->version << endl;
    cout << "origin_content:" << this->origin_content << endl;
    cout << "count_indicator:" << this->count_indicator << endl;
    cout << "data_bit_stream:" << this->data_bit_stream << endl;
    cout << "data_encoding:" << this->data_encoding << endl;
    cout << "len_data_encoding:" << this->len_data_encoding << endl;
    cout << "data_final:" << this->data_final << endl;
    cout << "len_final:" << this->len_final << endl;
}
```

给我很大的便利

5. 心得体会

5.1. 经验教训

本次作业和以前的作业相比差别非常大，主要问题在于看懂二维码原理，完成一种方式。调试的时候遇到的问题很难找到，这里使用之前的调试界面也不一定能直观地发现问题所在。所以还是回到了以前的中途暂停处理的办法。

在理解了二维码原理之后，就直接按照流程来完成。这里我先完成了一个小部分，比如完成版本1的L纠错模式的掩码0生成一小段，通过这个测试判断纠错码是否生成正确。然后，再逐渐加入其他分支，这样补全所有要求需要完成的部分，可以比较容易发现难以查找的错误。即使是这样，也还是在找bug的时候很困难。

通过这次作业，发现写函数的顺序对于查bug的难度有影响。先写到最基础的可测试部分，测试完成后继续写下面的代码，会减少很多麻烦。

从最近的作业看来，我们的目标是理解一个功能具体是如何实现的，而不是使用已有函数实

现几乎所有功能。通过深入了解，可以学到很多。

5.2. 作业总结

由于完全是按照二维码生成的步骤分配函数的，在代码前后的关系上可以做到有联系。在每次函数调用的时候，如果遇到需要再分函数，就继续分下去。按照步骤执行，大的逻辑一般都会很清晰。

在作业当中，最主要的重用代码方式是事先写好工具函数。在本次作业当中，最重要一个是纠错码生成函数，其次是字符数组的bit位排列到矩阵当中。这两个函数都适用于所有本次作业需要的大小。

6. 附件：源程序

90-b4-main.cpp:

```
/* 1651574 贾昊霖 1班*/
#include "90-b4.h"

int main()
{
    char str[1000];
    cout << "有些中文会变成日文..\n(由于掩模方式没有优化，所以扫描时间可能略长，请耐心等待扫描结果):\n";
    cout << "可能过长的字符串会跳出异常..本程序全部由是自己写的，包括纠错码的生成\n程序仍待完善，但时间实在有限，忘老师多多理解...\n";
    cout << "请输入串：\n";
    cin >> str;
    QRcode qr = str;
    qr.Solve();

    //test();
    return 0;
}
```

90-b4-exist.cpp:

//无任何借用代码

90-b4-myself.cpp:

```
/* 1651574 贾昊霖 1班*/
#include "90-b4.h"

PolyNode::PolyNode(int coe, int exp, PolyNode* next)
{
    this->coe = coe;
    this->exp = exp;
    this->next = next;
}
```

```
void Polynomial::GetHeadCoe()
{
    head_coe = this->next->coe;
}

void Polynomial::Append(PolyList &node)
{
    PolyList p = this->next, q = NULL;
    while (p && p->exp > node->exp) {
        q = p;
        p = p->next;
    }
    if (p == this->next) {
        this->next = node;
        node->next = p;
    }
    else {
        q->next = node;
        node->next = p;
    }
}

void Polynomial::Append(PolyNode &node)
{
    PolyList p = this->next;
    while (p && p->exp > node.exp) {
        p = p->next;
    }
    node.next = p->next;
    p->next = &node;
}

void Polynomial::MulPoly(int n)
{
    PolyList p = this->next;
    while (p) {
        p->coe += n;
        p->coe %= 255;
        p = p->next;
    }
}

void Polynomial::DivPoly(int n)
{
    PolyList p = this->next;
    while (p) {
        p->coe += 255;
        p->coe -= n;
        p->coe %= 255;
        p = p->next;
    }
}

void Polynomial::XorPoly(const Polynomial &L)
{
    PolyList p = L.next, q = this->next;
    if (!p || !q)
        return;
    this->next = q->next;
```

装 订 线

```
    delete q;

    q = this->next;
    p = p->next;

    while (p && q) {
        q->coe ^= p->coe;
        q = q->next;
        p = p->next;
    }
}

void Polynomial::ConvertAlpha2N()
{
    PolyList p = this->next;
    while (p) {
        p->coe = alpha2n[p->coe];
        p = p->next;
    }
}

void Polynomial::ConvertN2Alpha()
{
    PolyList p = this->next;
    while (p) {
        p->coe = n2alpha[p->coe];
        p = p->next;
    }
}

void Polynomial::PrintPoly()
{
    PolyList p = this->next;
    while (p) {
        cout << "coeffcient:" << p->coe << endl;
        cout << "exponent:" << p->exp << endl;
        p = p->next;
    }
}



QRcode::QRcode(char* str)
{
    if (!str) {
        cout << "input:\n";
        cin >> origin_content;
    }
    else
        strcpy(origin_content, str);
    strcpy(mode_indicator, "0100");
}

QRcode::~QRcode()
{
    cout << "the end of the program";
}

void QRcode::DecideVersion()
```

装 订 线

```
{
    int i;
    for (i = 0; i < sizeof(data_capacity) / sizeof(int); i++) {
        if (len_origin <= data_capacity[i])
            break;
    }
    version = i;
}

void QRcode::AddTerminator()
{
    if (len_origin < data_capacity[version]) {
        strcat(data_encoding, "0000");
        len_data_encoding += 4;
    }
}

void QRcode::AddPadding()
{
    const char str[][9] = { "11101100" ,"00010001" };
    data_final[0] = '\0';
    strcpy(data_final, data_encoding);
    len_final = len_data_encoding;
    //the process of adding '0' is unneccessary  due to the solution of 8-bits
    for (int i = 0; len_final < data_total[version] * 8; i ^= 1) {
        len_final += 8;
        strcat(data_final, str[i]);
    }
}

void QRcode::CorrectError(Polynomial &message_ply)
{
    Polynomial generate_ply;

    char tmp[10];
    int ctr = len_final / 8 - 1 + correction_number[version];       //important!

    for (char *p = data_final; p < data_final + len_final; p += 8) {
        strncpy(tmp, p, 8);
        tmp[8] = '\0';
        PolyList pl = new PolyNode(Str2Int(tmp), ctr--);
        message_ply.Append(pl);
    }
    for (int i = ctr; i >= 0; i--) {
        PolyList pl = new PolyNode(0, i);
        message_ply.Append(pl);
    }
    for (int i = 0; i <= correction_number[version]; i++) {         // attetion: <=
        PolyList pl = new PolyNode(generator_polynomial[version][i][0],
generator_polynomial[version][i][1]);
        generate_ply.Append(pl);
    }
    for (int i = 0; i < data_total[version]; i++) {
        message_ply.GetHeadCoe();
        int headcoe = message_ply.head_coe;
        int headcoe_alpha = n2alpha[headcoe];
        generate_ply.MulPoly(headcoe_alpha);
        generate_ply.ConvertAlpha2N();
        message_ply.XorPoly(generate_ply);
```

```
        generate_ply.ConvertN2Alpha();
        generate_ply.DivPoly(headcoe_alpha);
    }
    //setcolor(COLOR_BLACK);
    //message_ply.PrintPoly();
    //cout << "========================\n";
    //generate_ply.PrintPoly();
}

void QRcode::PositionDetectPattern(int x, int y)
{
    for (int i = 0; i < 7; i++)
        for (int j = 0; j < 7; j++)
            pattern[x + i][y + j] = BLACK;
    for (int i = 1; i < 6; i++) {
        pattern[x + i][y + 1] = pattern[x + i][y + 5] = WHITE;
        pattern[x + 1][y + i] = pattern[x + 5][y + i] = WHITE;
    }
}

void QRcode::SeperatorPattern()
{
    for (int i = 0; i <= 7; i++) {
        pattern[7][i] = pattern[pattern_size[version] - 8][i] = WHITE;
        pattern[i][7] = pattern[i][pattern_size[version] - 8] = WHITE;
    }
    for (int i = pattern_size[version] - 8; i <= pattern_size[version] - 1; i++) {
        pattern[i][7] = WHITE;
        pattern[7][i] = WHITE;
    }
}

void QRcode::TimingPattern()
{
    for (int i = 8; i <= pattern_size[version] - 9; i++)
        pattern[6][i] = pattern[i][6] = (i % 2) ? WHITE : BLACK;
}
void QRcode::AlignmentPattern()
{
    for (int i = 0; i <= 4; i++) {
        pattern[pattern_size[version] - 9 + i][pattern_size[version] - 9] =
            pattern[pattern_size[version] - 9][pattern_size[version] - 9 + i] = BLACK;
        pattern[pattern_size[version] - 9 + i][pattern_size[version] - 5] =
            pattern[pattern_size[version] - 5][pattern_size[version] - 9 + i] = BLACK;
    }
    setcolor(COLOR_BLACK);
    for (int i = 0; i <= 2; i++) {
        pattern[pattern_size[version] - 8 + i][pattern_size[version] - 8] =
            pattern[pattern_size[version] - 8][pattern_size[version] - 8 + i] = WHITE;
        pattern[pattern_size[version] - 8 + i][pattern_size[version] - 6] =
            pattern[pattern_size[version] - 6][pattern_size[version] - 8 + i] = WHITE;
    }
    setcolor(COLOR_BLACK);
    pattern[pattern_size[version] - 7][pattern_size[version] - 7] = BLACK;

}

void QRcode::AddFunctionPattern()
{
```

```
    PositionDetectPattern(0, 0);
    PositionDetectPattern(pattern_size[version] - 7, 0);
    PositionDetectPattern(0, pattern_size[version] - 7);
    SeperatorPattern();
    TimingPattern();
    if (version > 1)
        AlignmentPattern();
    // Dark Dot
    pattern[8][pattern_size[version] - 8] = BLACK;
}

void QRcode::FillingFormatInfo()
{
    //strcpy(correct_version, "01");
    //strcpy(mask_version, "100");
    mask_version = 7;
    strcpy(correct_str, mask_module[mask_version]);
    //strrev(correct_str);

    int p = strlen(correct_str) - 1;
    for (int i = 1; i <= 8; i++, p--)
        pattern[pattern_size[version] - i][8] = (correct_str[p] - '0') ? BLACK : WHITE;
    for (int i = 1; i <= 7; i++, p--)
        pattern[8][pattern_size[version] - 8 + i] = (correct_str[p] - '0') ? BLACK :
WHITE;
    p = strlen(correct_str) - 1;
    for (int i = 0; i <= 5; i++, p--)
        pattern[8][i] = (correct_str[p] - '0') ? BLACK : WHITE;
    pattern[8][7] = (correct_str[p--] - '0') ? BLACK : WHITE;
    pattern[8][8] = (correct_str[p--] - '0') ? BLACK : WHITE;
    pattern[7][8] = (correct_str[p--] - '0') ? BLACK : WHITE;
    for (int i = 5; i >= 0; i--, p--)
        pattern[i][8] = (correct_str[p] - '0') ? BLACK : WHITE;
}

void QRcode::InitialPattern()
{
    pattern = new int*[pattern_size[version] + 1];
    for (int i = 0; i <= pattern_size[version]; i++) {
        pattern[i] = new int[pattern_size[version] + 1];
        memset(pattern[i], EMPTY, sizeof(int) * pattern_size[version]);
    }
}

void QRcode::DrawPattern()
{
    for (int i = 0; i < pattern_size[version]; i++) {
        for (int j = 0; j < pattern_size[version]; j++) {
            if (pattern[j][i] == BLACK)
                setcolor(COLOR_BLACK);
            else if (pattern[j][i] == WHITE)
                setcolor(COLOR_HWHITE);
            else
                setcolor(COLOR_WHITE);
            cout << "  ";
        }
        putchar('\n');
    }
}
```

装 订 线

```
void QRcode::PrintPattern()
{
    for (int i = 0; i < pattern_size[version]; i++) {
        for (int j = 0; j < pattern_size[version]; j++) {
            if (pattern[j][i] == BLACK)
                cout << BLACK;
            else if (pattern[j][i] == WHITE)
                cout << WHITE;
            else
                cout << '.';
        }
        putchar('\n');
    }
}

void QRcode::FillingData(Polynomial &message_ply)
{
    PolyList p = message_ply.next;
    //cout << "====================\n";
    //cout << data_final << endl;
    //cout << strlen(data_final) / 8 << endl;
    //cout << "====================\n";

    flag = new bool*[pattern_size[version] + 1];
    for (int i = 0; i <= pattern_size[version]; i++) {
        flag[i] = new bool[pattern_size[version] + 1];
        memset(flag[i], 0, sizeof(bool) * pattern_size[version]);
    }


    strcpy(data_correct_code, data_final);

    while (p) {
        int tmp_num = p->coe;
        char tmp_str[10];
        Int2Int_binary(tmp_num, tmp_str);
        strcat(data_correct_code, tmp_str);
        //cout << tmp_num << " " << tmp_str << endl;
        p = p->next;
    }

    int ctr = 0;
    for (int i = pattern_size[version] - 1, t = 1; i >= 0; i -= 2, t ^= 1) {
        if (i == 6)
            i--;

        if ((t && i > 6) || (!t && i < 6)) {
            for (int j = pattern_size[version] - 1; j >= 0; j--) {
                for (int k = 0; k >= -1; k--) {
                    if (pattern[i + k][j] == EMPTY) {
                        pattern[i + k][j] = (data_correct_code[ctr++] - '0') ? BLACK :
WHITE;
                        flag[i + k][j] = true;
                    }
                    //DrawPattern();
                    //Sleep(70);
                }
            }
```

```
        }
        else {
            for (int j = 0; j <= pattern_size[version] - 1; j++) {
                for (int k = 0; k >= -1; k--) {
                    if (pattern[i + k][j] == EMPTY) {
                        pattern[i + k][j] = (data_correct_code[ctr++] - '0') ? BLACK :
WHITE;
                        flag[i + k][j] = true;
                    }
                    //DrawPattern();
                    //Sleep(70);

                }
            }
        }
    }
}

void QRcode::MaskingPattern()
{
    cout << endl << endl << endl;
    for (int i = 0; i < pattern_size[version]; i++) {
        for (int j = 0; j < pattern_size[version]; j++) {
            if (flag[j][i])
                if ((((i + j) % 2) + ((i * j) % 3)) % 2 == 0)
                    pattern[j][i] = !pattern[j][i];
        }
    }
}

void QRcode::Solve()
{
    DecideVersion();

    //Data analyse
    CountIndicator();
    GetDataStream();
    ConcatenateData();
    //if (!strcmp(data_encoding,
"01000000110101001000011001010110110001101100011011110010110000100000011101110110111
10111001001101100011001000010000 1"))
    //  cout << "yes\n";

    AddTerminator();

    //Padding add
    AddPadding();

    //Correct error
    Polynomial message_ply;
    CorrectError(message_ply);

    InitialPattern();
    AddFunctionPattern();
    FillingFormatInfo();


    FillingData(message_ply);
```

```
    MaskingPattern();

    //Draw pattern

    DrawPattern();

    //PrintPattern();
    //DrawPattern();
}

int QRcode::Str2Int(char *str)
{
    int len = strlen(str);
    int num = 0;
    for (int i = 0; i < len; i++) {
        num *= 2;
        num += str[i] - '0';
    }
    return num;
}

int QRcode::Str2Int_binary(char *str)
{
    int len = strlen(str), num = 0;
    char *p = str;
    for (; p < str + len; p++)
        if (*p != '0')
            break;
    for (; p < str + len; p++) {
        num *= 10;
        num += *p - '0';
    }
    return num;
}

void QRcode::Int2Int_binary(int n, char *str, int length)
{
    int num = 0;
    length = CountNumLength - 1;
    int tmp_2pow = int(pow(2, length));
    char *p = str;

    while (tmp_2pow) {
        *p++ = n / tmp_2pow + '0';
        n %= tmp_2pow;
        tmp_2pow /= 2;
    }
    *p = '\0';
}

void QRcode::CountIndicator()
{
    len_origin = strlen(origin_content);
    Int2Int_binary(len_origin, count_indicator);
}

void QRcode::GetDataStream()
{
    char tmp_str[10];
```

```
    strcpy(data_bit_stream, "");
    for (int i = 0; i < len_origin; i++) {
        char tmp_ch = origin_content[i];
        char tmp_n = char(1);
        for (int j = 0; j < 8; j++)
            tmp_str[j] = ((tmp_ch >> (7 - j)) & tmp_n) + '0';
        tmp_str[8] = '\0';
        strcat(data_bit_stream, tmp_str);
    }
}

void QRcode::ConcatenateData()
{
    strcpy(data_encoding, mode_indicator);
    strcat(data_encoding, count_indicator);
    strcat(data_encoding, data_bit_stream);
    len_data_encoding = strlen(data_encoding);
}

void QRcode::print()
{
    setcolor(COLOR_BLACK);
    cout << "version:" << this->version << endl;
    cout << "origin_content:" << this->origin_content << endl;
    cout << "count_indicator:" << this->count_indicator << endl;
    cout << "data_bit_stream:" << this->data_bit_stream << endl;
    cout << "data_encoding:" << this->data_encoding << endl;
    cout << "len_data_encoding:" << this->len_data_encoding << endl;
    cout << "data_final:" << this->data_final << endl;
    cout << "len_final:" << this->len_final << endl;
}

void test()
{
    Polynomial pa, pb;
    /*int test[][2] =
{ 16,25,19,24,99,23,144,22,236,21,17,20,236,19,17,18,236,17,17,16,236,15,17,14,236,1
3,17,12,236,11,1,10,236,9,17,8,236,7,0,6,0,5,0,4,0,3,0,2,0,1,0,0 };
    int test2[][2] = { 0,7,87,6,229,5,146,4,149,3,238,2,102,1,21,0 };*/

    int test[][2] =
{ 32,25,91,24,11,23,120,22,209,21,114,20,220,19,77,18,67,17,64,16,236,15,17,14,236,1
3,17,12,236,11,17,10,0,9,0,8,0,7,0,6,0,5,0,4,0,3,0,2,0,1,0,0 };
    int test2[][2] = { 0,10,251,9,67,8,46,7,61,6,118,5,70,4,64,3,94,2,32,1,45,0 };

    //for (int i = 0; i < 19; i++) {
    //  PolyList pl = new PolyNode(16,25,NULL);
    //  //pa.Append(&PolyNode(16, 25, NULL));
    //  pa.Append(pl);
    //}

    for (int i = 0; i < sizeof(test) / sizeof(*test); i++) {
        PolyList pl = new PolyNode(test[i][0], test[i][1]);
        pa.Append(pl);
    }
    for (int i = 0; i < sizeof(test2) / sizeof(*test2); i++) {
        PolyList pl = new PolyNode(test2[i][0], test2[i][1]);
        //PolyList pl = new PolyNode(alpha2n[test2[i][0]], test[i][1]);
        pb.Append(pl);
```

```
    }
    cout << "==================\n";
    cout << "pa:\n";
    pa.PrintPoly();
    cout << "==================\n";
    cout << "pb:\n";
    pb.PrintPoly();

    for (int i = 0; i < 16; i++) {
        cout << "==================\n";
        pa.GetHeadCoe();
        int headcoe = pa.head_coe;
        int headcoe_alpha = n2alpha[headcoe];
        cout << "headcoe:" << headcoe << endl;
        cout << "headcoe_alpha:" << headcoe_alpha << endl;

        cout << "==================\n";
        cout << "after multiply, pb:\n";
        pb.MulPoly(headcoe_alpha);
        pb.PrintPoly();

        cout << "==================\n";
        cout << "after convert alpha to n, pb:\n";
        pb.ConvertAlpha2N();
        pb.PrintPoly();


        cout << "==================\n";
        cout << "after xor, pa:\n";
        pa.XorPoly(pb);
        pa.PrintPoly();

        cout << "==================\n";
        cout << "recover pb:\n";
        pb.ConvertN2Alpha();
        pb.DivPoly(headcoe_alpha);
        pb.PrintPoly();
    }

}
```

90-b4.h:

```
/* 1651574 贾昊霖 1班*/
#pragma once

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
#include <cmath>
#include "cmd_console_tools.h"

using namespace std;
typedef int Status;

//#define ERROR -1
#define OK 1
#define CountNumLength 8
#define BLACK 1
#define WHITE 0
```

```
#define EMPTY 0x3F3F3F3F

#define CalcSize(V) (((V-1)*4)+21)

#define DarkModule() ([（4 * V）+ 9],8)

const int n2alpha[] =
{ INT_MAX ,0 ,1 ,25 ,2 ,50 ,26 ,198 ,3 ,223 ,51 ,238 ,27 ,104 ,199 ,75 ,4 ,100 ,224
,14 ,52 ,141 ,239 ,129 ,28 ,193 ,105 ,248 ,200 ,8 ,76 ,113 ,5 ,138 ,101 ,47 ,225 ,36
 ,15 ,33 ,53 ,147 ,142 ,218 ,240 ,18 ,130 ,69 ,29 ,181 ,194 ,125 ,106 ,39 ,249 ,185
,201 ,154 ,9 ,120 ,77 ,228 ,114 ,166 ,6 ,191 ,139 ,98 ,102 ,221 ,48 ,253 ,226 ,152 ,
37 ,179 ,16 ,145 ,34 ,136 ,54 ,208 ,148 ,206 ,143 ,150 ,219 ,189 ,241 ,210 ,19 ,92 ,
131 ,56 ,70 ,64 ,30 ,66 ,182 ,163 ,195 ,72 ,126 ,110 ,107 ,58 ,40 ,84 ,250 ,133 ,186
 ,61 ,202 ,94 ,155 ,159 ,10 ,21 ,121 ,43 ,78 ,212 ,229 ,172 ,115 ,243 ,167 ,87 ,7 ,1
12 ,192 ,247 ,140 ,128 ,99 ,13 ,103 ,74 ,222 ,237 ,49 ,197 ,254 ,24 ,227 ,165 ,153 ,
119 ,38 ,184 ,180 ,124 ,17 ,68 ,146 ,217 ,35 ,32 ,137 ,46 ,55 ,63 ,209 ,91 ,149 ,188
 ,207 ,205 ,144 ,135 ,151 ,178 ,220 ,252 ,190 ,97 ,242 ,86 ,211 ,171 ,20 ,42 ,93 ,15
8 ,132 ,60 ,57 ,83 ,71 ,109 ,65 ,162 ,31 ,45 ,67 ,216 ,183 ,123 ,164 ,118 ,196 ,23 ,
73 ,236 ,127 ,12 ,111 ,246 ,108 ,161 ,59 ,82 ,41 ,157 ,85 ,170 ,251 ,96 ,134 ,177 ,1
87 ,204 ,62 ,90 ,203 ,89 ,95 ,176 ,156 ,169 ,160 ,81 ,11 ,245 ,22 ,235 ,122 ,117 ,44
 ,215 ,79 ,174 ,213 ,233 ,230 ,231 ,173 ,232 ,116 ,214 ,244 ,234 ,168 ,80 ,88 ,175 };
const int alpha2n[] =
{ 1 ,2 ,4 ,8 ,16 ,32 ,64 ,128 ,29 ,58 ,116 ,232 ,205 ,135 ,19 ,38 ,76 ,152 ,45 ,90 ,
180 ,117 ,234 ,201 ,143 ,3 ,6 ,12 ,24 ,48 ,96 ,192 ,157 ,39 ,78 ,156 ,37 ,74 ,148 ,5
3 ,106 ,212 ,181 ,119 ,238 ,193 ,159 ,35 ,70 ,140 ,5 ,10 ,20 ,40 ,80 ,160 ,93 ,186 ,
105 ,210 ,185 ,111 ,222 ,161 ,95 ,190 ,97 ,194 ,153 ,47 ,94 ,188 ,101 ,202 ,137 ,15
,30 ,60 ,120 ,240 ,253 ,231 ,211 ,187 ,107 ,214 ,177 ,127 ,254 ,225 ,223 ,163 ,91 ,1
82 ,113 ,226 ,217 ,175 ,67 ,134 ,17 ,34 ,68 ,136 ,13 ,26 ,52 ,104 ,208 ,189 ,103 ,20
6 ,129 ,31 ,62 ,124 ,248 ,237 ,199 ,147 ,59 ,118 ,236 ,197 ,151 ,51 ,102 ,204 ,133 ,
23 ,46 ,92 ,184 ,109 ,218 ,169 ,79 ,158 ,33 ,66 ,132 ,21 ,42 ,84 ,168 ,77 ,154 ,41 ,
82 ,164 ,85 ,170 ,73 ,146 ,57 ,114 ,228 ,213 ,183 ,115 ,230 ,209 ,191 ,99 ,198 ,145
,63 ,126 ,252 ,229 ,215 ,179 ,123 ,246 ,241 ,255 ,227 ,219 ,171 ,75 ,150 ,49 ,98 ,19
6 ,149 ,55 ,110 ,220 ,165 ,87 ,174 ,65 ,130 ,25 ,50 ,100 ,200 ,141 ,7 ,14 ,28 ,56 ,1
12 ,224 ,221 ,167 ,83 ,166 ,81 ,162 ,89 ,178 ,121 ,242 ,249 ,239 ,195 ,155 ,43 ,86 ,
172 ,69 ,138 ,9 ,18 ,36 ,72 ,144 ,61 ,122 ,244 ,245 ,247 ,243 ,251 ,235 ,203 ,139 ,1
1 ,22 ,44 ,88 ,176 ,125 ,250 ,233 ,207 ,131 ,27 ,54 ,108 ,216 ,173 ,71 ,142 ,INT_MAX };

const int data_capacity[] = { 17,32,53,78,106 };
const int data_total[] = { 19,34,55,80,108 };
const int correction_number[] = { 7,10,15,20,26 };
const int generator_polynomial[][28][2] = {
    { 0,7,87,6,229,5,146,4,149,3,238,2,102,1,21,0 },
{ 0,10,251,9,67,8,46,7,61,6,118,5,70,4,64,3,94,2,32,1,45,0 },
{ 0,15,8,14,183,13,61,12,91,11,202,10,37,9,51,8,58,7,58,6,237,5,140,4,124,3,5,2,99,1
,105,0 },
{ 0,20,17,19,60,18,79,17,50,16,61,15,163,14,26,13,187,12,202,11,180,10,221,9,225,8,8
3,7,239,6,156,5,164,4,212,3,212,2,188,1,190,0 },
{ 0,26,173,25,125,24,158,23,2,22,103,21,182,20,118,19,17,18,145,17,201,16,111,15,28,
14,165,13,53,12,161,11,21,10,245,9,142,8,13,7,102,6,48,5,227,4,153,3,145,2,218,1,70,
0 }
};
const char correct_level[][3] = { "01","00","11","10" };
const char mask_module[][16] = { "111011111000100", "111001011110011",
"111110110101010", "111100010011101", "110011000101111",
"110001100011000","110110001000001","110100101110110" };
const int remain_bit[] = { 0,7,7,7,7 };
const int alignmentPattern_loc[][2] = { { 0,0 },{ 18,18 },{ 22,22 },{ 26,26 },{ 30,30 } };
const int pattern_size[] = { 21,25,29,33,37 };
```

```
class PolyNode {
public:
    int coe;//coefficient
    int exp;//exponent
    PolyNode* next;
    PolyNode(int = 0, int = 0, PolyNode* = NULL);
};

typedef PolyNode* PolyList;

class Polynomial {
public:
    int head_coe;
    PolyList next;
    Polynomial(PolyList = NULL) { next = NULL; }
    void Append(PolyNode &);
    void Append(PolyList &);
    void GetHeadCoe();
    void MulPoly(int n);
    void DivPoly(int n);
    void XorPoly(const Polynomial &);
    void ConvertAlpha2N();
    void ConvertN2Alpha();
    void PrintPoly();
};

class QRcode
{
private:
    char origin_content[105];
    char mode_indicator[8];
    char count_indicator[15];
    char data_bit_stream[1050];
    char data_encoding[1000];
    char data_final[1000];

    char data_correct_code[2000];
    int len_origin;
    int len_data_encoding;
    int len_final;
    int version;
    int mask_version;

    int cost[8][4];

    char correct_str[20] = "111101000110011";

public:
    QRcode(char* = NULL);
    ~QRcode();

    int **pattern;
    bool **flag;

    void Solve();
    void DecideVersion();
    void CountIndicator();
```

装 订 线

```
    void GetDataStream();
    void ConcatenateData();
    void AddTerminator();
    void AddPadding();
    void CorrectError(Polynomial &);
    void InitialPattern();
    void AddFunctionPattern();
    void PositionDetectPattern(int, int);
    void SeperatorPattern();
    void PrintPattern();
    void DrawPattern();
    void AlignmentPattern();
    void TimingPattern();
    void FillingFormatInfo();
    void FillingData(Polynomial &);

    void MaskingPattern();

    int Str2Int(char*);
    int Str2Int_binary(char*);
    void Int2Int_binary(int, char *, int = 0);

    void print();
};
```

装 订 线