加密算法之Playfair及Hill算法

1 Playfair算法

1.1 简介

Playfair密码（Playfair cipher）是一种替换密码，1854年由英国人查尔斯·惠斯通（Charles Wheatstone）发明。
它有一些不太明显的特征：密文的字母数一定是偶数；任意两个同组的字母都不会相同，如果出现这种字符必是乱码和虚码。

1.2 算法

算法依据一个5*5的字母矩阵组成的密码表。该矩阵使用一个关键词构造，方法是按从左到右、从上到下顺序填入关键词的字母（去除重复字母）后，将字母表其余字母填入。例如，当关键词取为*monarchy*时，字母矩阵为

$$\begin{bmatrix} M & O & N & A & R \\ C & H & Y & B & D \\ E & F & G & I/J & K \\ L & P & Q & S & T \\ U & V & W & X & Z \end{bmatrix} \tag{1}$$

密码表里排列有25个字母。如果一种语言字母超过25个，可以去掉使用频率最少的一个。
如，法语一般去掉w或k，德语则是把i和j合起来当成一个字母看待。英语中z使用最少，可以去掉它。

1.2.1 加密

Playfair加密方法是先将明文按两个字母一组进行分组，然后在矩阵中找对应的密文，取密文的规则如下：

1. 若明文分组出现在相同字母在一组，则在重复的明文字母中插入一个填充字母（比如 *k*)进行分割后重新

分组（如 `balloon` 被重新分组为 `ba lk lo on` ）

2. 若分组到最后一组时只有一个字母，则补充字母*k*
3. 若明文字母在矩阵中同行，则循环取其右边字母为密文（如*ar*被加密为*RM* ）
4. 若明文字母在矩阵中同列，则循环取其下边字母为密文（如*mu*被加密为*CM* ）
5. 若明文字母在矩阵中不同行不同列，则取其同行且与下一字母同列的字母为密文（如*hs*被加密为*BP*，*ea*被加密为*IM*或*JM* ）

例如，明文为 `I'm maple` ，分组成为 `Im ma pl ek` （不处理非字母字符）；用上述矩阵加密后的密文为 `EA OR QP KT` 。

1.2.2 解密

解密规则则与加密规则相反，由于密文中同一分组内没有相同字母，故仅考虑寻找明文情况，取密文规则如下：

1. 若密文字母在矩阵中同行，则循环取其左边字母为密文（如*OR*被解密为*MA* ）
2. 若密文字母在矩阵中同列，则循环取其下边字母为密文（如*KT*被加密为*EK* ）
3. 若密文字母在矩阵中不同行不同列，则取其同行且与下一字母同列的字母为密文（如*EA*被加密为*IM* ）

1.3 实践

明白了算法原理后，就可以亲手实践一个加密解密的小程序了。
在动手之前，首先是考虑如何高效的字母矩阵中找到对应的字母。
首先，在分组出现在同行或同列的情况下，会进行循环移动取操作，此时可以将第0列拷贝到第5列，第0行拷贝的第5行的方法来省略求余运算。
另外，如果单纯地使用遍历算法，那么每一次进行加密时就要对每一个分组进行时间复杂度为O(n^2)的查找操作，明显效率是很低的。为了提高查找效率，在一开始的矩阵初始化中，作者使用了**c++**的关联容器*map*进行操作，初始化代码如下：

```
PlayFair::PlayFair(string key)
{
    char letters[] = {'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'};

    keyMap.clear();//keyMap为PlayFair成员

    for(size_t i = 0; i < key.size(); ++i){//创建矩阵首字符串键对
       keyMap.insert(make_pair(key[i], keyMap.size()));//当前map大小即为字母应在的位置
    }
    for(size_t i = 0; i < 26; ++i){//创建剩余键对
        keyMap.insert(make_pair(letters[i], keyMap.size()));
    }

    /*这里采用让y和z在同一位置的方法，与传统不同*/
    if(keyMap['z'] < keyMap['y'])
        keyMap['y'] = keyMap['z']; //令z与y的位置相同
    else
        keyMap['z'] = keyMap['y']; //令z与y的位置相同

    for(size_t i = 0; i < keyMap.size(); ++i){
        char ch = 'a' + i;
        keyArray[ keyMap[ch] / 5 ][ keyMap[ch] % 5 ] = ch;
    }

    /* 拷贝第0行至第5行，第0列至第5列
     * 虽然两部分可以合并在一个循环体内，因为rows与cols相同，
     * 但为了可读性，则用这种方法，并且这种方法仅是增加了i的赋值与判断而已
     */
    for(size_t i = 0; i < rows; ++i)
        keyArray[i][cols - 1] = keyArray[i][0];
    for(size_t i = 0; i < cols; ++i)
        keyArray[rows - 1][i] = keyArray[0][i];
}
```

而在加密和解密时，算法所依赖的是字母所在矩阵中的行、列坐标值，在此可以将字母的一维值（即将字母所在矩阵当成一维数组看待时的下标值）转换为二维值（字母在矩阵中的行、列坐标值）被保存至*pair*中，从而可以高效的在矩阵中找出对应字母，这时需要另外一个存储字符和对应*pair*的map，创建代码如下：

```
//根据keyMap创建矩阵关联容器
void CreateMatrixMap(map<char, size_t> &keyMap, map<char, pair<size_t, size_t>> &matrixMap)
{
    matrixMap.clear();
    for(auto it = keyMap.begin(); it != keyMap.end(); ++it){
        matrixMap.insert(make_pair(it->first,
                                   make_pair(it->second / (PlayFair::rows - 1),
                                             it->second % (PlayFair::cols - 1)))
                         );
    }
}
```

而在进行字母的搜索时，可以直接使用*pair*对进行查找，代码如下：

```
//将ch1和ch2通过矩阵转换为其他字母
void PlayFair::FairTransfer(char &ch1, char &ch2, map<char, pair<size_t, size_t>> &matrixMap
)
{
    pair<size_t, size_t> pairCh1, pairCh2;
    pairCh1 = matrixMap[ch1];
    pairCh2 = matrixMap[ch2];

    if(pairCh1.first == pairCh2.first){//行与行相等
        ch1 = keyArray[pairCh1.first][pairCh1.second + 1];
        ch2 = keyArray[pairCh2.first][pairCh2.second + 1];
    }else if(pairCh1.second == pairCh2.second){//列与列相等
        ch1 = keyArray[pairCh1.first + 1][pairCh1.second];
        ch2 = keyArray[pairCh2.first + 1][pairCh2.second];
    }else{//不同行列
        ch1 = keyArray[pairCh1.first][pairCh2.second];
        ch2 = keyArray[pairCh2.first][pairCh1.second];
    }
}
```

处理完再矩阵中查询并替换字母的问题后，就可以继续考虑下一问题——如何进行分组。
分组是很简单的处理过程，只要线性搜索明文，将非字母保留并合并两个待替换字母即可，但同时还要考虑字母数为单数的情况，方法是补字母k，代码如下：

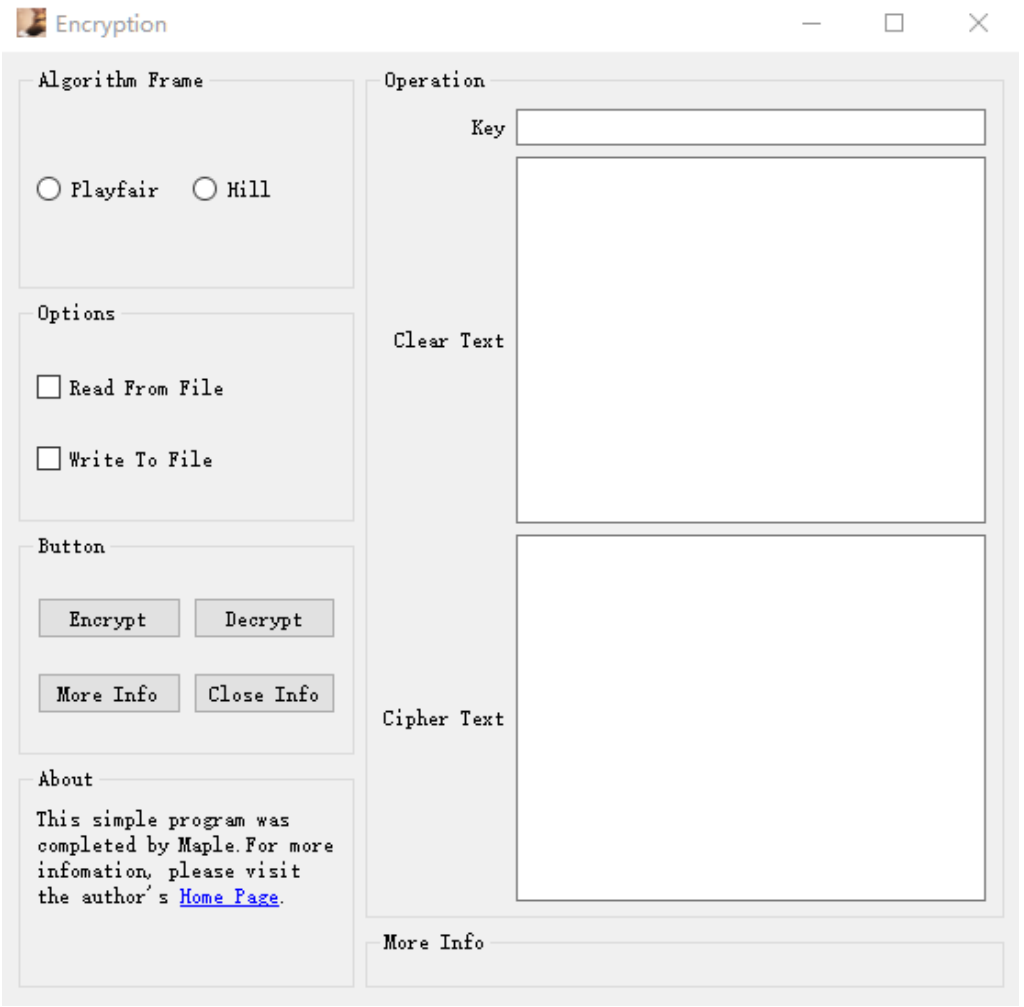
```cpp
//对明文clear进行加密
void PlayFair::encrypt(std::string &clear)
{
    string tmp;
    if(clear.empty()) return ;//无效的明文

    tmp.reserve(clear.size() * 2);//预分配空间
    str2lowstr(clear);//首先转换为小写字母形式
    map<char, pair<size_t, size_t>> matrixMap;
    CreateMatrixMap(keyMap, matrixMap);//构建矩阵关联容器

    for(size_t i = 0; i < clear.size(); ++i){
        char ch = clear.at(i);
        string norLetter = "";
        if(!isLowLetter(ch))
            tmp.push_back(ch);//不处理非小写字母
        else{
            char ch2;

            while(++i < clear.size() && !isLowLetter(ch2 = clear.at(i))){
                norLetter.push_back(ch2);//如果第一个字母为小写字母，之后的不是小写字母，则不进行处理
            }

            if(i < clear.size()){//当前还有剩余字母
                if(ch2 == ch){//若两字母相同，则赋第二个字母为k，并且使计数字母i减1
                    ch2 = 'k';
                    --i;
                }
            }else{//没有剩余字母，则补充ch2为k
                ch2 = 'k';
            }

            this->FairTransfer(ch, ch2, matrixMap);//转换字符

            /* 将两个字符和非字母字符串添加进字符串 */
            tmp.push_back(ch);
            tmp += norLetter;
            tmp.push_back(ch2);
        }
    }

    clear.swap(tmp);//交换明文为密文
}
```

解密过程与加密过程一致，只不过将33行的转换字符方法*FairTranser*改为返回字符方法*FairReturn*。*FairReturn*方法可根据解密原理参照*FairTranser*方法进行修改。详细可参考playfair.cpp.

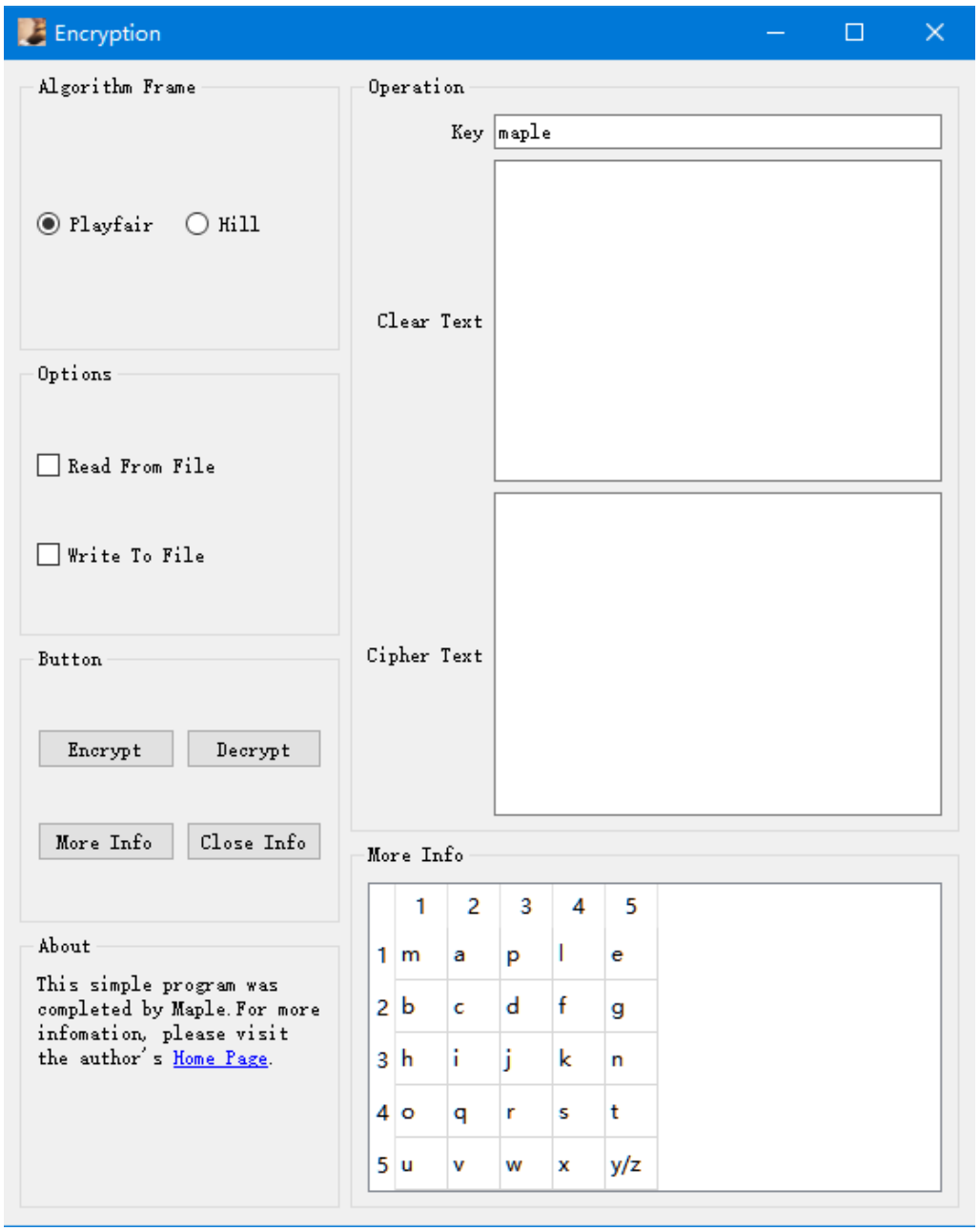
1.4 演示

此部分用于显示我的小程序成果，简单介绍功能部分，如无需进一步了解该算法的朋友可以绕过此节。
该程序基于 `QT5开发框架` ，运行环境为 `windows` 。

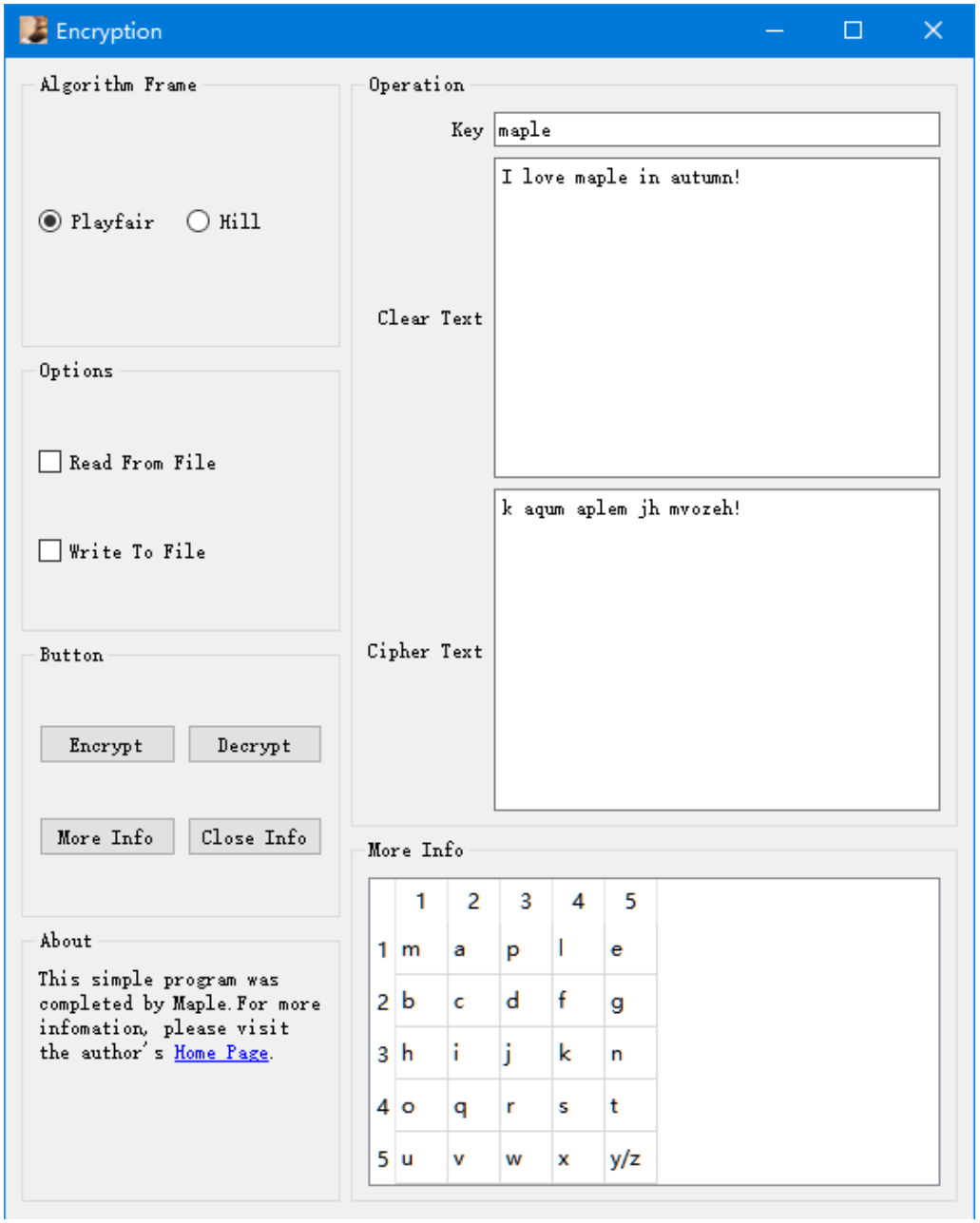
- 初始界面
  程序的初始界面如下图所示，左上角的 `Algorithm Frame` 为算法选取框，在这里，可以选取的算法有 `Playfair` 以及即将要介绍的 `Hill` 算法。算法选取框下面是 `Options` 框，用于选择是否从文件中读取及是否向文件中写入结果。而在下方的 `Button` 框，则用于执行**加密**、**解密**、**查看更多信息**及**关闭更多信息**功能。最下方的则是作者的一些介绍。
  右侧的 `Key` 则是用于填入生成算法矩阵的密钥，如若不填充，则默认以字母序填充矩阵。



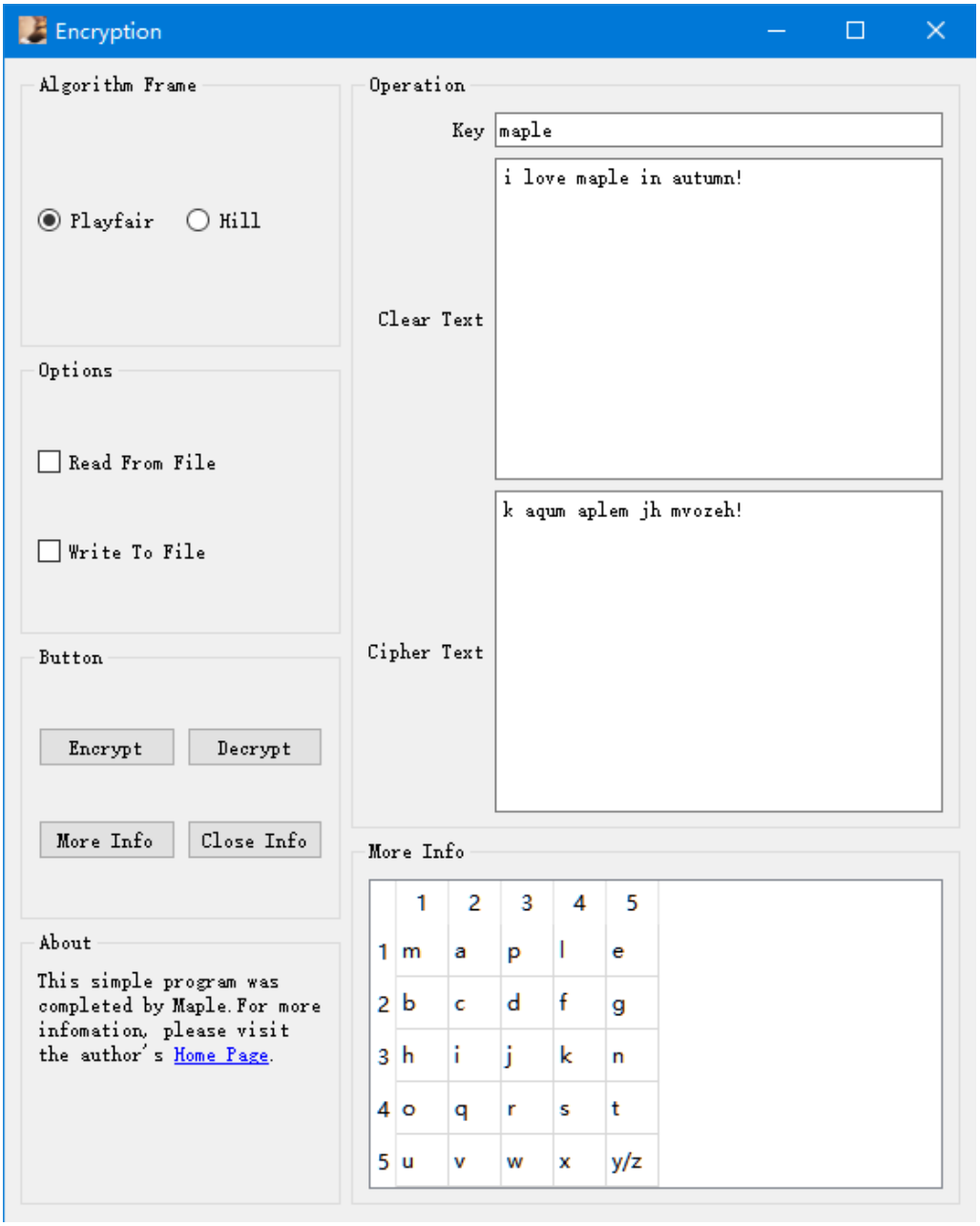
- 输入密钥并生成矩阵
  在选择了 `Playfair` 算法后，在 `Key` 处输入密钥*maple*，并点击 `More Info` 按钮后，即可显示算法矩阵，如下图所示。（当然，如果你没有选择任何一个算法就想显示更多信息的话，是会弹出一个提醒框，表示你没有选择任何算法的，在此不做篇幅介绍）



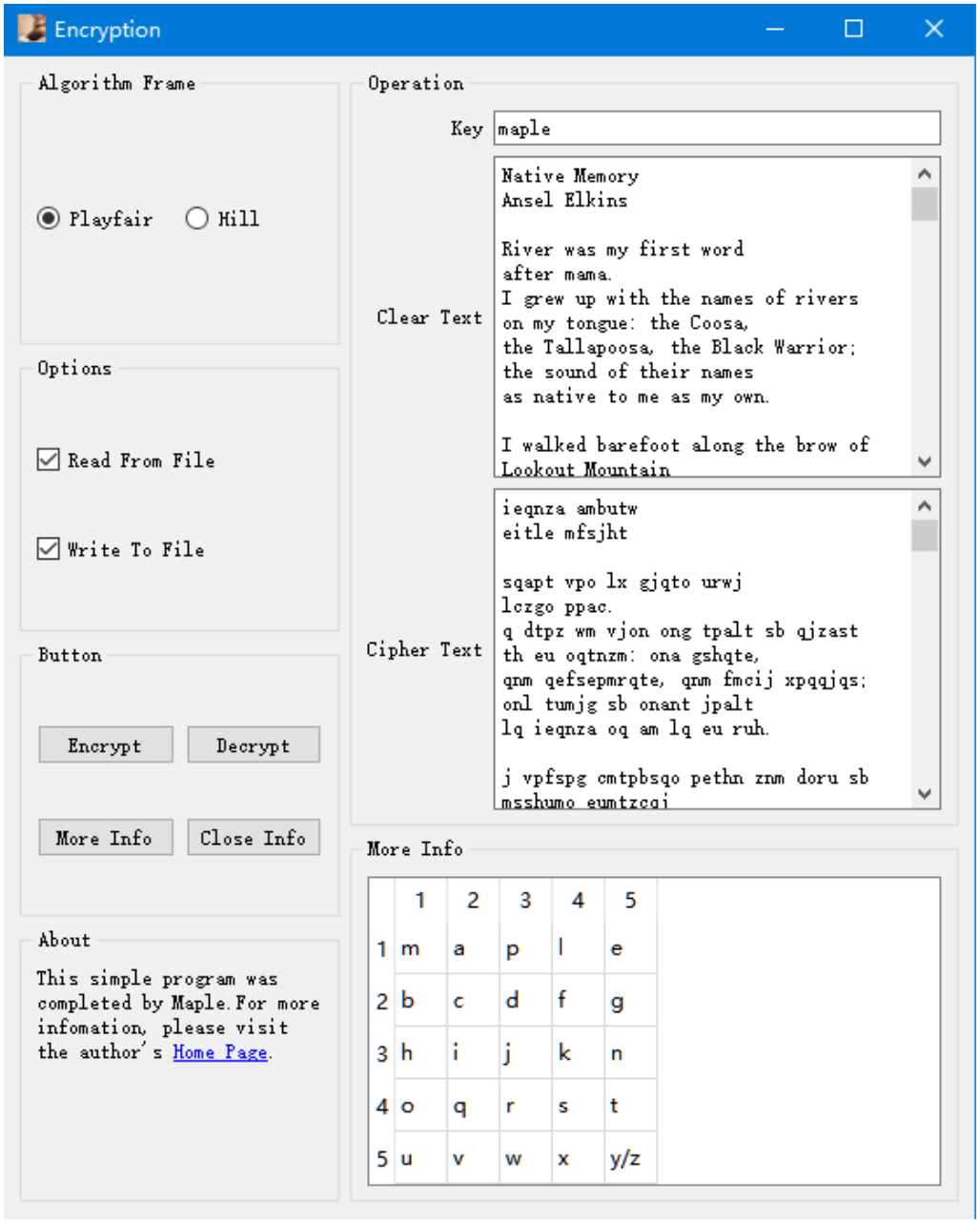
- 加密
  在 `Operation` 框中，有表示明文的 `Clear Text` 和密文的 `Cipher Text` 文本输入框，当向进行加密时，可以在明文框输入明文，在点击加密按钮 `Encrypt` 后，加密后的明文将会出现在密文框，如下图所示，明文I love maple in autumn!被加密为密文k aqum aplem jh mvozeh!



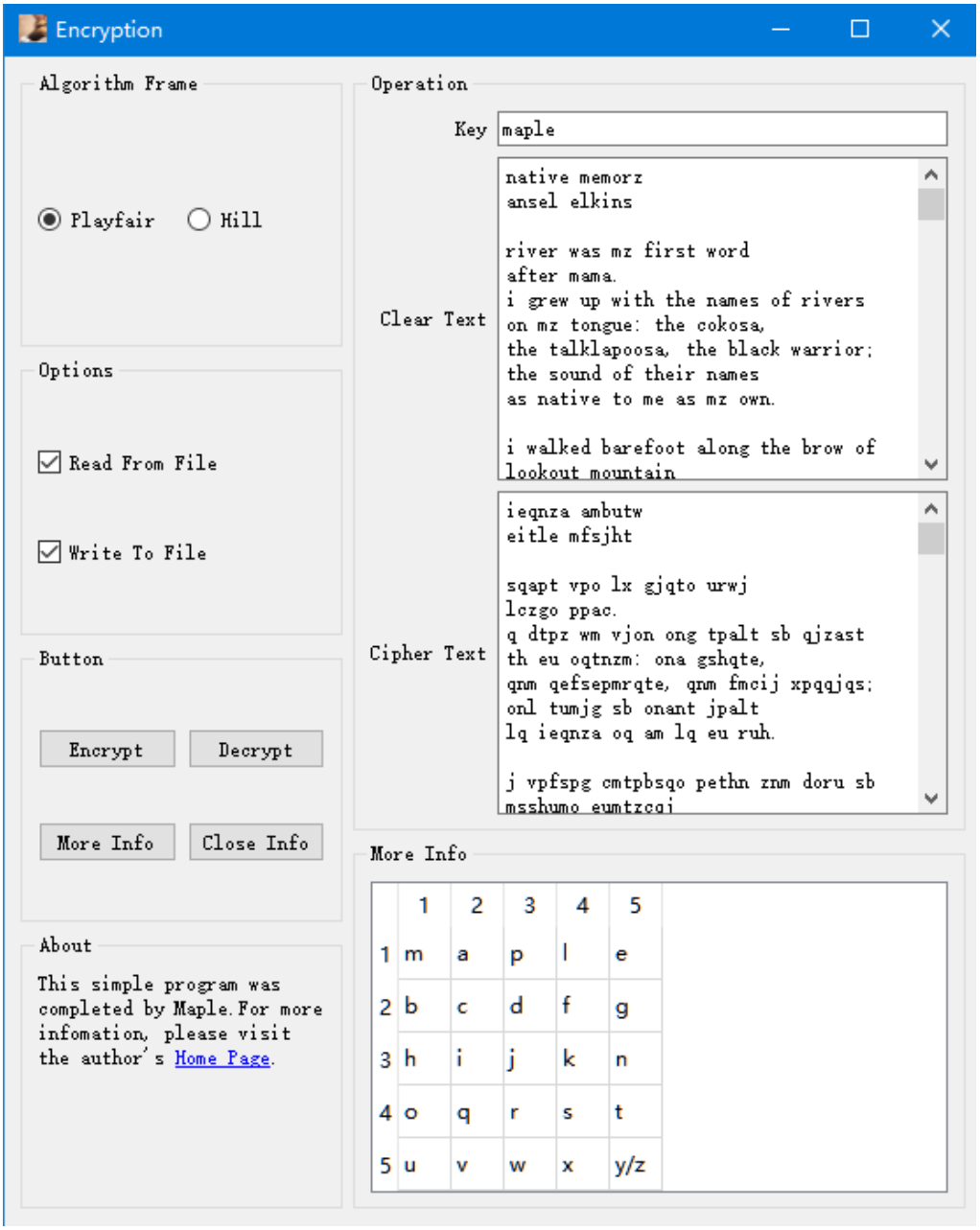
- 解密
  如果看到上述那句话，一般人当然会意思不明所以，而这就需要进行解密了，直接点击解密按钮 `Decrypt` ，则可以将密文解密并放置于明文框中。如下图所示，密文被解密为i love maple in autumn!。可以看到大写*I*被改为了小写*i*，这是因为算法矩阵是填充了小写字母的缘故，所有字母都会被相应的小写字母所替代。



- 从文件中读取并写入文件
  当勾选了复选框 `Read From File` 及 `Write To File` 后（也可以只选其中一个执行部分功能）就可以从文件中读取明文进行加密并将密文存储于文件中。点击加密按钮后便会出现文件选择框，在此，读取一份名为 `poem.txt` 的文件，里面是一首诗，接下来查看加密情况和加密文件，如下图所示，加密文件被读取到明文框，密文被显示到密文框。



而在点击解密后，也可以将加密文件进行解密并保存在明文文件中，明文同样会被显示到明文框，只不过所有字母都被改为小写，并可能出现在出现同组字母为相同时添加额外字母或最后一个分组只有一个字母的情况，此时都会多补充一个字母 `k` 。如下图所示。



2 Hill算法

`Hill` 密码也是一种多字母替代密码，他由数学家 `Lester Hill` 于1929年研制成功。
密码基于矩阵的线性变换，最大的好处就是隐藏了字符的频率信息，使得通过传统的字符频率来破译密文的方法失败。
该密码算法取*m*个连续的明文字母，并用*m*个密文字母代替，这种代替由*m*个线性方程决定。若 `m = 3` ，则该系统可以描述为

$$C_1 = (K_{11}P_1 + K_{12}P_2 + K_{13}P_3)mod26$$
$$C_2 = (K_{21}P_1 + K_{32}P_2 + K_{33}P_3)mod26$$
$$C_3 = (K_{31}P_1 + K_{32}P_2 + K_{33}P_3)mod26$$

或用矩阵表示

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} \quad (2)$$

- *C*和*P*是长度为3的列向量，分别表示明文和密文

- *K*是3 x 3矩阵，表示加密密钥
- 加密操作要执行模26运算

2.1 加密实例

例如，加密密钥为

$$K = \begin{bmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{bmatrix} \tag{3}$$

欲对明文 `pay more money` 进行加密，则先将明文按m(此例中*m* = 3)个字母为一组进行分组(不足3个字母时补字母，如x)，然后对每组字母求密文。该明文的前3个字母表示为

$$pay = [15 \quad 0 \quad 24]^T \tag{4}$$

计算密文的过程如下：

$$K[15 \quad 0 \quad 24]^T = [375 \quad 819 \quad 486]^T mod 26 = [11 \quad 13 \quad 18]^T = LNS \tag{5}$$

依次类推，可得密文为 `LNS HDL EWM TRW` 。
解密时使用逆矩阵，

$$K^{-1} = \begin{bmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{bmatrix} \tag{6}$$

对密文

$$LNS = [11 \quad 13 \quad 18]^T \tag{7}$$

做解密运算如下

$$K^{-1}[11 \quad 13 \quad 18]^T mod 26 = [431 \quad 494 \quad 570]^T mod 26 = [15 \quad 0 \quad 24]^T = pay \tag{8}$$

从上述算法过程可以看出， `Hill` 密码能够完全隐藏单字母的频率，而在*m*的取值越大时，其隐藏的字母频率的个数也越大，如在*m* = 3时，能够隐藏两个字母的频率信息。

2.1 密钥求法

用作加密的矩阵必须在自然数集上是可逆的，否则就不可能成功译码。并且只有在矩阵的行列式值和26互质时，该矩阵才可用。

求出逆矩阵的方法，可以通过使用伴随矩阵计算行列式值或用增广矩阵进行初等变换得到，这些都是一些基本的线性代数知识，读者可以自行查阅。

这里给出一个求逆矩阵的方法，代码参考自某C++大神，但是求出的逆矩阵的元素值为小数，需要将其转换为整数并乘上相应的倍数使得逆矩阵的行列式值与26互质时才能作为解密矩阵。在代码中的注释提示了相应的修改。

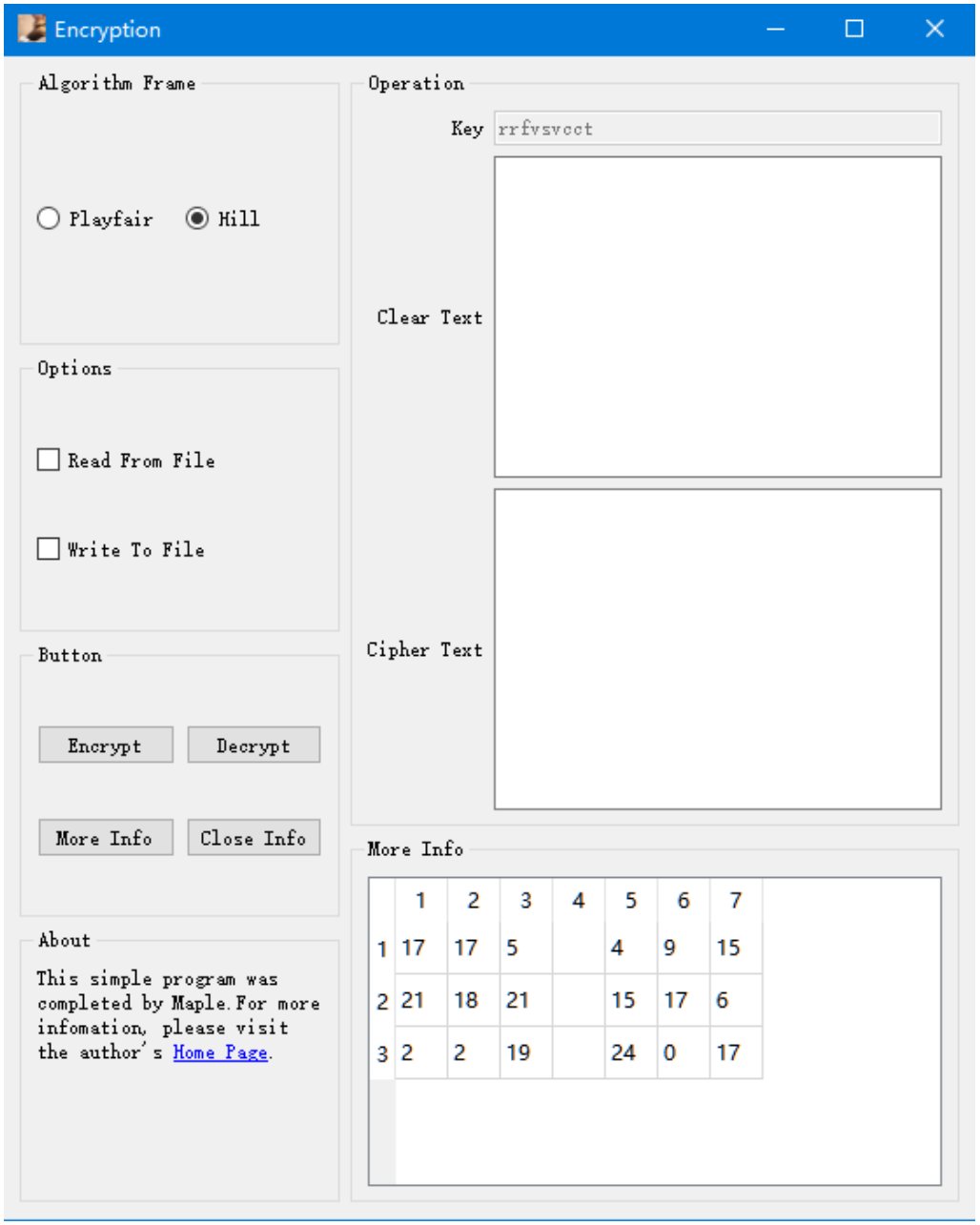
```cpp
#include <cmath>
#include <utility>

template<typename T>
struct matrix_size;

template<typename T, size_t N>
struct matrix_size<T[N][N]>{
    typedef T type;
    static size_t const size = N;
};

template<typename Array>
bool CreateInvMatrix(Array const &in_, Array &out){
    typedef typename matrix_size<Array>::type type;
    size_t const n = matrix_size<Array>::size;
    Array in;
    /* 复制输入，并初始化out */
    for (size_t i = 0; i != n; ++i) {
        for (size_t j = 0; j != n; ++j) {
            in[i][j] = in_[i][j];
            out[i][j] = 0;
        }
        out[i][i] = 1;// 将out赋值为单位阵
    }

    type tmp = 1;
    for (size_t c = 0; c != n; ++c) {
        /* 选取列主元，避免出现in[c][c]为0或过小导致商溢出的情形 */
        size_t pivot = 0;
        type maximum = 0;
        for (size_t r = c; r != n; ++r) {
            type const tmp = std::abs(in[r][c]);
            if (tmp > maximum) {
                maximum = tmp;
                pivot = r;
            }
        }
        if (maximum == 0) return false;// 不可逆

        /* 交换c, pivot两行 */
        for (size_t i = c; i != n; ++i) std::swap(in[c][i], in[pivot][i]);
        for (size_t i = 0; i != n; ++i) std::swap(out[c][i], out[pivot][i]);

        /* 正规化 */
        for (size_t i = c + 1; i != n; ++i) {
            in[c][i] /= in[c][c];
        }
        for (size_t i = 0; i != n; ++i) {
            out[c][i] /= in[c][c];
        }

        tmp *= in[c][c];//记录除数以恢复

        // 行变换消元
        in[c][c] = 0;
        for (size_t r = 0; r != n; ++r) {
            for (size_t i = 0; i != n; ++i) {
                in[r][i] += in[c][i] * -in[r][c];
                out[r][i] += out[c][i] * -in[r][c];
            }
        }
```

```
    }

    tmp *= 9;
    //在上述的密钥中，为了将行列式值修改为26互质，
    //故需乘上倍数因子9，具体数值可在恢复为整数后，
    //再通过求行列式值与26的最大公约数为1的方法，
    //不断改变倍数因子以获得

    /* 将小数恢复为整数 */
    for (size_t i = 0; i != n; ++i)
        for (size_t j = 0; j != n; ++j)
            out[i][j] *= tmp;

    return true;
}
```

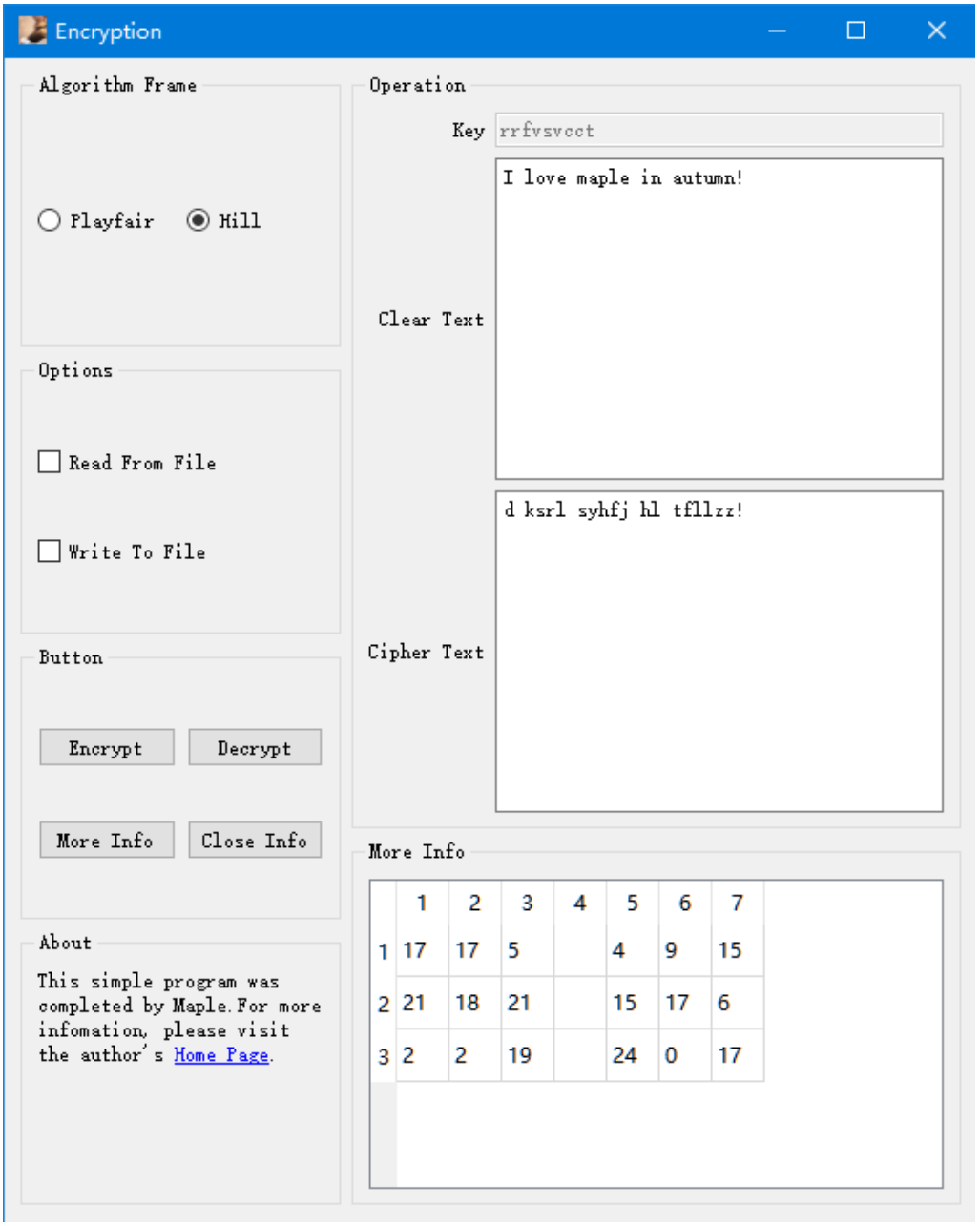
2.2 演示

此小节同样是为了显示我的小程序成果，简单介绍加密解密功能，如无需进一步了解该算法的朋友可以绕过此节。该程序与上一程序共用一个 `UI` ，在同一界面里提供加密解密功能。

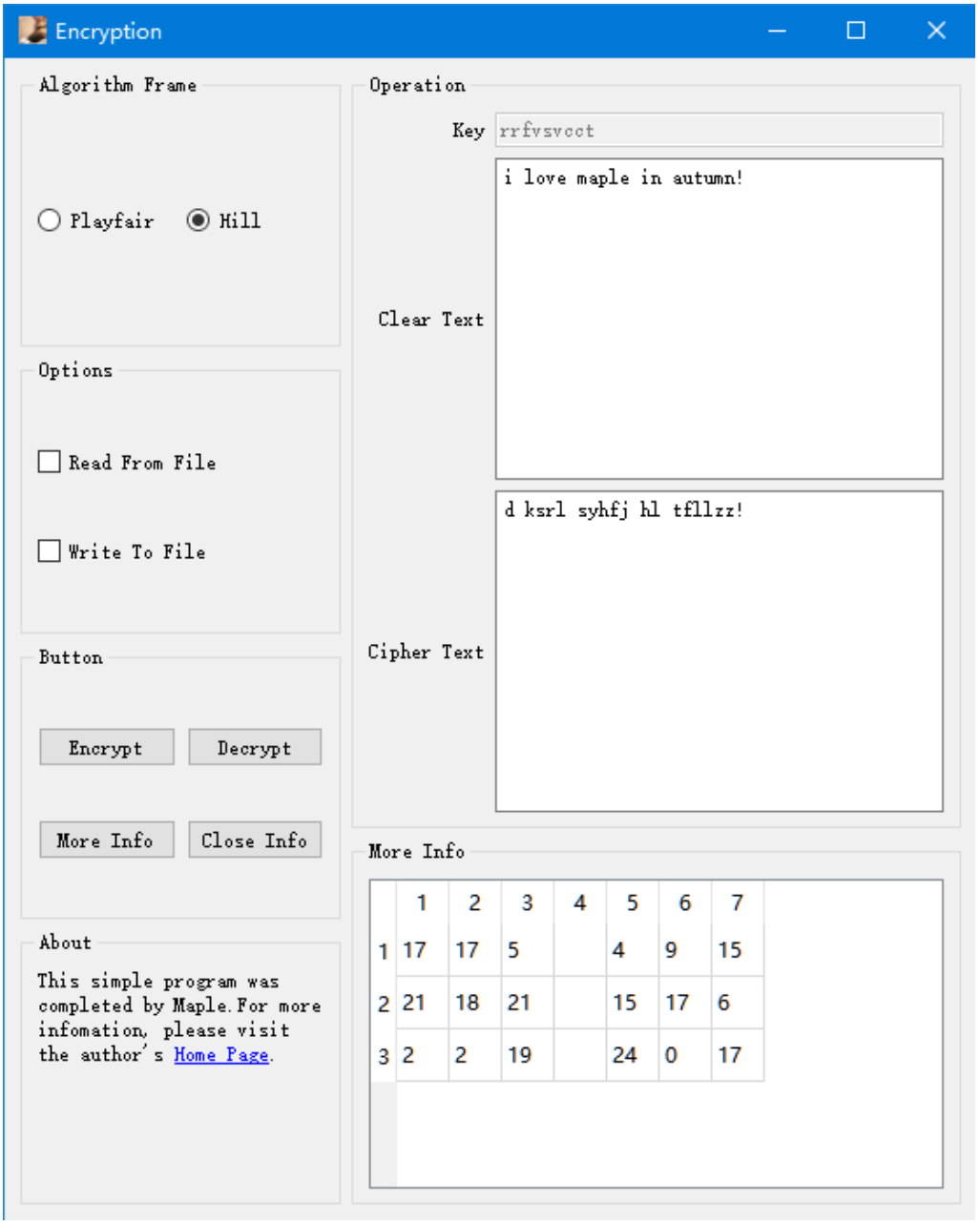
- 密钥及生成矩阵
  在此，将密钥给写死在程序中了，但理论上是可以求出逆矩阵的，只不过需要废点脑力和时间去求出倍数因子。但可以先不考虑这些，明白原理也可。
  在 `More Info` 框中，显示了两个矩阵，左边是加密矩阵，右边则是解密矩阵。



- 加密
  在明文框中输入明文I love maple in autumn!，点击加密按钮后则可以看到密文框出现了密文d ksrl syhfj hl tfllzz!，如下图所示。



- 解密
  在上一步骤之后，直接点击解密按钮，即可进行解密。如下图所示，密文被解密为i love maple in autumn!。可以看到大写I被改为了小写i，这是因为算法矩阵是填充了小写字母的缘故，所有字母都会被相应的小写字母所替代。



至于加密文件和解密文件的功能就不在这里进行演示了，其效果和 `Playfair` 雷同。如想进一步了解程序代码，可访问程序仓库

3 总结

在编写这个程序的时候，一开始，把更多的时间花在了如何构建好数据结构上，因为本质上，这两个算法的思想都还比较简单，不需要耗费大量的精力就能够理解并实现出来，但在考虑如何优化的时候，就需要先动一下脑筋。如在编写 `Playfair` 时，为提高查找效率，使用了关联容器 `map` 和 `pair` 。而在处理复选框时，使用了掩码知识来减少变量空间(如下所示)。

```
/* 使用异或运算来表示设置读写文件标志，
 * 其中,读为二进制1，不读为0，写为1，不写为0
 * 读标志在第0位，写标志在第1位
 */
#define READ_MASK  0x01
#define WRITE_MASK 0x02

//使用异或运算翻转标志位
void MainWindow::checkReadFile(){ rwFileFlag ^= READ_MASK; }
void MainWindow::checkWriteFile(){ rwFileFlag ^= WRITE_MASK; }

bool isReadFromFile(void) { return (rwFileFlag & READ_MASK) == READ_MASK; }//判断是否从文件中读取
bool isWriteToFile(void) { return (rwFileFlag & WRITE_MASK) == WRITE_MASK; }//判断是否写入文件
```

所以，得出一个结论，要完成一件任务其实是可以很轻松地完成，但如果想要在更高的层面上进取，则要多花费时间来思考和设计。相信一如既往地坚持高水平开发，会让自己在编程之路走得更远，对事物的看法也能上升更高境界！

希望如此！