****

**密码学实验**

实 验 报 告

|  |  |
| --- | --- |
| **学 院** | 网络空间安全学院 |
| **专 业** | 信息安全 |
| **班 级** | 17273612 |
| **学 号** | 17271229 |
| **学生姓名** | 杨尽能 |
| **教师姓名** | 吕秋云 |
| **完成日期** | 2019.4.30 |
| **成 绩** |  |

**实验一 Caesar密码加密算法**

* 1. **实验目的**

掌握Caesar密码加解密原理，并利用Visual C++实现。

* 1. **实验原理和内容概述**

Caesar密码的加密原理是对明文加上一个密钥得到密文，假设密钥为3，那么字母“a”对应的ASCII码23为97，加上3得100正好是字母“d”的ASCII码值。

* 1. **实验环境**

Windows10，Visual Studio2017，C++。

* 1. **实验主要步骤及实验结果记录（含截图，关键源码，文字解释等）**

1. **编程思路：**
2. 可以采用MFC编程，构建Caesar密码加解密界面，实现与用户的交互。界面中应该包含加解密选择、明文输入、密文输出、密钥设置、加解密按钮等。
3. 为了方便使用，密钥允许用户输入任意整数。然后在编写的程序中以模运算将密钥限定到0~25的数。
4. 原始Caesar密码只是针对英文字符，且不区分大小写。在这里扩展要求为可以加解密英文字符、数字、标点符号等各种字符。
5. 加密实现两种方式：一是只限定英文字母，采用公式：密文字符=‘a’或‘A’+（明文字符-‘a’或‘A’+password%26+26）%26，解密反之；二是对任意字符，直接利用公式：密文字符=明文字符+password，解密反之。
6. **关键代码及注释：**

加解密按钮事件：

void CCaesar1Dlg::OnBnClickedButton1()

{

// TODO: 在此添加控件通知处理程序代码

int check\_1 = ((CButton\*)GetDlgItem(IDC\_RADIO1))->GetCheck();//获取加密选项是否被选中

int check\_2 = ((CButton\*)GetDlgItem(IDC\_RADIO2))->GetCheck();//获取解密选项是否被选中

CString message;//明文

CString cipher;//密文

CString key; //密钥

GetDlgItemText(IDC\_EDIT1, message); //把明文中的字符读入到message中

GetDlgItemText(IDC\_EDIT3, key); //把密钥内容读到key中

char \*message1 = message.GetBuffer(0); //把message转换为字符类型

char \*cipher1;

int key1 = 0;//默认密钥=0

cipher1 = message1;

int size = message.GetLength();

for (int i = 0; i < size; i++) {

if(message1[i]>='A'&&message1[i]<='Z'){

if (check\_1 == 1 && check\_2 == 0) { //如果选中加密

key1 = \_ttoi(key);

}

else if (check\_1 == 0 && check\_2 == 1) {//如果选中解密

key1 = 26 - \_ttoi(key);

}

cipher1[i] = 'A' + (message1[i] - 'A' + key1 % 26 + 26) % 26;

}

if (message1[i] >= 'a'&&message1[i] <= 'z') {

if (check\_1 == 1 && check\_2 == 0) { //如果选中加密

key1 = \_ttoi(key);

}

else if (check\_1 == 0 && check\_2 == 1) {//如果选中解密

key1 = 26 - \_ttoi(key);

}

cipher1[i] = 'a' + (message1[i] - 'a' + key1 % 26 + 26) % 26;

}

if (message1[i] >= '0'&&message1[i] <= '9') {

if (check\_1 == 1 && check\_2 == 0) { //如果选中加密

key1 = \_ttoi(key);

}

else if (check\_1 == 0 && check\_2 == 1) {//如果选中解密

key1 = 10 - \_ttoi(key);

}

cipher1[i] = '0' + (message1[i] - '0' + key1 % 10 + 10) % 10;

}

if (message1[i] >= '!'&&message1[i] <= '/') {

if (check\_1 == 1 && check\_2 == 0) { //如果选中加密

key1 = \_ttoi(key);

}

else if (check\_1 == 0 && check\_2 == 1) {//如果选中解密

key1 = 15 - \_ttoi(key);

}

cipher1[i] = '!' + (message1[i] - '!' + key1 % 15 + 15) % 15;

}

if (message1[i] >= ':'&&message1[i] <= '@') {

if (check\_1 == 1 && check\_2 == 0) { //如果选中加密

key1 = \_ttoi(key);

}

else if (check\_1 == 0 && check\_2 == 1) {//如果选中解密

key1 = 7 - \_ttoi(key);

}

cipher1[i] = ':' + (message1[i] - ':' + key1 % 7 + 7) % 7;

}

if (message1[i] >= '{'&&message1[i] <= '}') {

if (check\_1 == 1 && check\_2 == 0) { //如果选中加密

key1 = \_ttoi(key);

}

else if (check\_1 == 0 && check\_2 == 1) {//如果选中解密

key1 = 3 - \_ttoi(key);

}

cipher1[i] = '{' + (message1[i] - '{' + key1 % 3 + 3) % 3;

}

}

cipher = cipher1;

SetDlgItemText(IDC\_EDIT2, cipher);

}

清空操作：

void CCaesar1Dlg::OnBnClickedButton2()

{

// TODO: 在此添加控件通知处理程序代码

CString empty("");

SetDlgItemText(IDC\_EDIT1, empty);

SetDlgItemText(IDC\_EDIT2, empty);

SetDlgItemText(IDC\_EDIT3, empty);

}

1. **扩展要求实现：**

扩展要求对数字和标点符号也进行加解密。这个实现起来其实和英文字母是差不多的。对英文字母来说，范围是从‘A’到‘Z’或‘a’到‘z’，中间都是有26个字符，依据公式可以实现；而对于数字来说，范围从‘0’到‘9’，其中有10个数，只需把公式中的26改为10即可实现对数字的加解密；标点符号也是一样的道理。

其中扩展部分的代码为：

if (message1[i] >= '0'&&message1[i] <= '9') {

if (check\_1 == 1 && check\_2 == 0) { //如果选中加密

key1 = \_ttoi(key);

}

else if (check\_1 == 0 && check\_2 == 1) {//如果选中解密

key1 = 10 - \_ttoi(key);

}

cipher1[i] = '0' + (message1[i] - '0' + key1 % 10 + 10) % 10;

}

if (message1[i] >= '!'&&message1[i] <= '/') {

if (check\_1 == 1 && check\_2 == 0) { //如果选中加密

key1 = \_ttoi(key);

}

else if (check\_1 == 0 && check\_2 == 1) {//如果选中解密

key1 = 15 - \_ttoi(key);

}

cipher1[i] = '!' + (message1[i] - '!' + key1 % 15 + 15) % 15;

}

if (message1[i] >= ':'&&message1[i] <= '@') {

if (check\_1 == 1 && check\_2 == 0) { //如果选中加密

key1 = \_ttoi(key);

}

else if (check\_1 == 0 && check\_2 == 1) {//如果选中解密

key1 = 7 - \_ttoi(key);

}

cipher1[i] = ':' + (message1[i] - ':' + key1 % 7 + 7) % 7;

}

if (message1[i] >= '{'&&message1[i] <= '}') {

if (check\_1 == 1 && check\_2 == 0) { //如果选中加密

key1 = \_ttoi(key);

}

else if (check\_1 == 0 && check\_2 == 1) {//如果选中解密

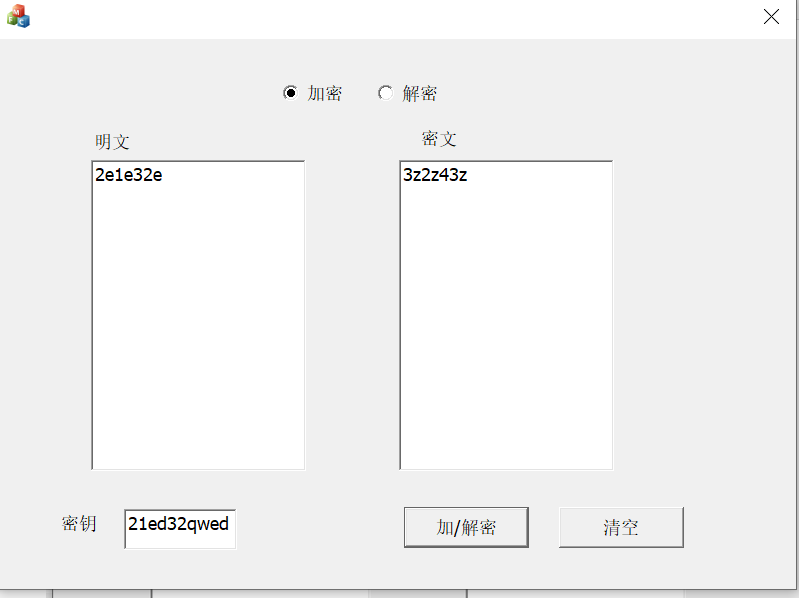
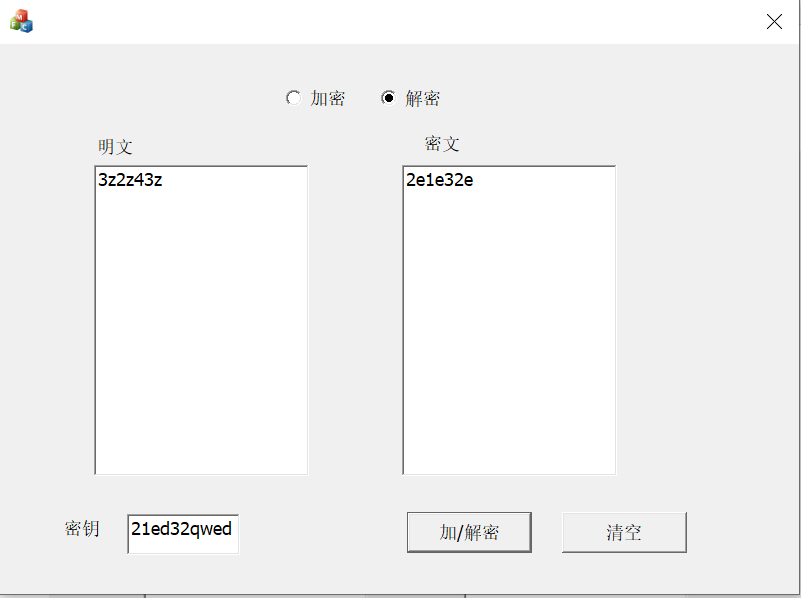
key1 = 3 - \_ttoi(key);

}

cipher1[i] = '{' + (message1[i] - '{' + key1 % 3 + 3) % 3;

}

1. **运行截图**：

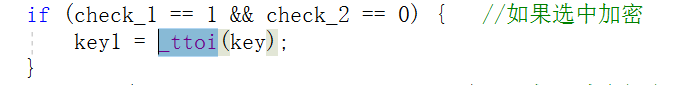
  

可以看到明文加密后的密文解密出来和明文是一样的。

* 1. **实验分析总结及心得**

1. **遇到的问题和解决方法：**
2. **问题：**界面里输入的文本是CString类型的，而CString类型不能直接进行运算。

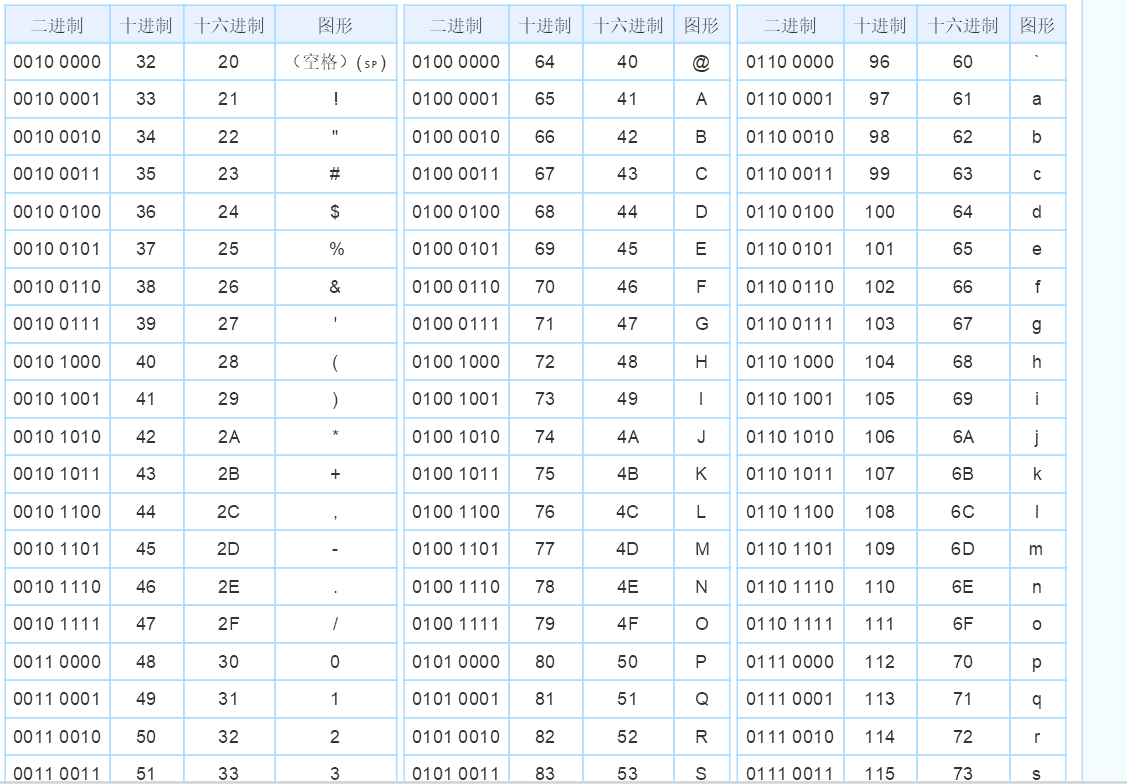
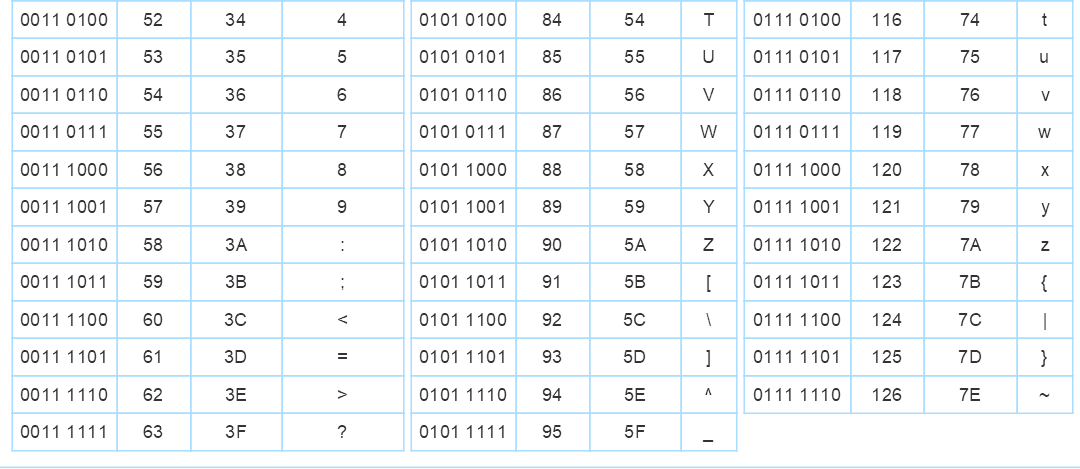
**解决方法：**在C++中可以直接使用\_ttoi(CString)来把CString类型的变量直接转化为int类型，然后就可以进行运算。



1. **问题：**在对标点符号进行加密时，出现有一些标点符号不能被加密，原样输出的情况。



**解决方法：**这个问题困扰了我好久，最后发现错误原因很白痴。。。。就是没有把所有标点符号的ASCII码都包含进去！！！解决方法很简单，对照ASCII码表，找到所有标点符号的范围，即如下图所示：

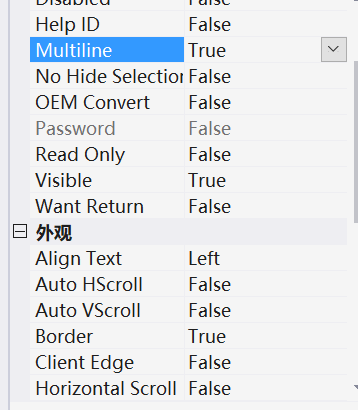
可以看到‘！’~‘/’、‘：’~‘@’、‘[’~‘`’、‘{’~‘~’是标点符号的四个范围，把这些全部作为筛选条件，即可对所有标点符号进行加密。



1. **问题：**界面中明文和密文编辑框不能自动换行，输入的字符会一直向右延伸，不方便用户查看。



**解决方法：**这个问题经过上网查询后得到了解决：在编辑框的属性界面，将Multiline属性设置为True，将AutoHScroll属性设置为False，这样可以实现每一行填满后自动换行。



更改结果：



1. **总结经验：**
2. 这是我第一次用MFC实现简单界面编程，感觉很新奇。没有想象中的难，但初次使用时在熟悉界面控件的时候也碰到了不少问题，还好最后都解决了。MFC界面编程时对每一个控件都可以单独编程，比如加解密两个按钮就要分别写出两个触发事件，这和以往我们在编写程序时只需要用一个主函数来进行测试有很大的不同。但是这种方式确实能够更好地进行人机交互。
3. MFC的编辑框中用的是CString变量，不能直接用来做运算等操作，因此需要用\_ttoi等函数将其转化为可计算的int等类型。
4. 加密算法本质上就是输入明文和密钥，然后输出加密后的密文。而对称加密算法还要求能够用同一个密钥对密文解密，然后输出加密前的明文。搞清楚了基本的输入输出，就只需要关注算法本身就行了。其他的对称加密算法也是这样，只不过算法不一样而已。
5. 一个好的加密算法不仅要求完成目标，还要考虑算法的效率问题，程序运行的速度要尽可能地快，否则也很难说是一个好的算法。因此这就要求我们要尽可能地优化算法，使其运行速度加快。

**实验二 DES密码加密算法**

1. **实验目的**

掌握DES密码加解密原理，并利用Visual C++实现。

1. **实验原理和内容概述**

利用DES加密算法，实现“Hi,this is DES!”字串的加密，并同时解密。

1. **实验环境**

Windows10，Visual Studio2017，C++。

1. **实验主要步骤及实验结果记录（含截图，关键源码，文字解释等）**
2. **编程思路：**
   * 1. 采用MFC编程，构建DES密码加解密界面，实现与用户的交互。界面中应该包含明文输入、密文输出、解密文输出、密钥设置、加解密按钮等。
     2. 为了增加程序的正确性、可读性和可维护性，实现模块化编程，将算法实现的功能写成子函数后供上一层函数调用，避免直接在一个函数里完成所有代码的编写。
3. **关键代码及注释：**
4. **DES加/解密**

static void DES\_Yangjinneng(char Out[8], char In[8], bool flag) { //flag表示加密或解密

static bool M[64];

static bool Temp[32];

static bool\* Li = &M[0];

static bool\* Ri = &M[32];//64 bits明文 经过IP置换后，分成左右两份

ByteToBit(M, In, 64); //8个字符转换成64比特

if (flag) { //加密

Transform(M, M, IP\_Table, 64);//M进行IP置换

for (int i = 0; i < 16; i++) {

memcpy(Temp, Ri, 32); //暂存右半边

F\_func(Ri, SubKey[i]); //上一轮的右半边和第二轮子密钥轮变换

Xor(Ri, Li, 32); //异或

memcpy(Li, Temp, 32); //下一轮左半边等于上一轮右半边

}

memcpy(Temp, Li, 32);

memcpy(Li, Ri, 32);

memcpy(Ri, Temp, 32); //保证加密和解密的一致性，交换Ri和Li

Transform(M, M, IPInv\_Table, 64);

}

else {

Transform(M, M, IP\_Table, 64);

for (int i = 15; i >= 0; i--) { //子密钥使用顺序和加密相反

memcpy(Temp, Ri, 32);

F\_func(Ri, SubKey[i]);

Xor(Ri, Li, 32);

memcpy(Li, Temp, 32);

}

memcpy(Temp, Li, 32);

memcpy(Li, Ri, 32);

memcpy(Ri, Temp, 32); //保证加密和解密的一致性，交换Ri和Li

Transform(M, M, IPInv\_Table, 64);

}

BitToByte(Out, M, 64);

}

1. **设置子密钥:**

**输入为8个字节，输出为16轮的48位子密钥存放在SubKey[16][48]。**

static void SetSubKey(char Key[8]) {

static bool K[64];

static bool \* KL = &K[0];

static bool \* KR = &K[28];//将64位密钥串去掉8位奇偶校验位后，分成两份

ByteToBit(K, Key, 64);

Transform(K, K, PC1\_Table, 56);//PC\_1置换

for (int i = 0; i < 16; i++) {

RotateL(KL, 28, LS\_Table[i]);//循环左移

RotateL(KR, 28, LS\_Table[i]);

Transform(SubKey[i], K, PC2\_Table, 48);//PC\_2置换

}

}

1. **F函数：**

static void F\_func(bool In[32], bool Ki[48]) {

static bool MR[48];

Transform(MR, In, E\_Table, 48); //E变换

Xor(MR, Ki, 48);//异或

S\_func(In, MR);//S盒代替

Transform(In, In, P\_Table, 32);//P盒置换

}

1. **S盒**

static void S\_func(bool Out[32], bool In[48]) { //将8组，每组6 bits的串，转化为8组，每组4 bits

for (char i = 0, j, k; i < 8; i++) { //分成8组，每组6比特

j = (In[0 + 6 \* i] << 1) + In[5 + 6 \* i];//第一个比特和第六个比特

k = (In[1 + 6 \* i] << 3) + (In[2 + 6 \* i] << 2) + (In[3 + 6 \* i] << 1) + In[4 + 6 \* i];//中间4个比特

ByteToBit(Out + 4 \* i, &S\_Box[i][j][k], 4);//取出第j行第k列元素

}

}

1. **矩阵变化**

static void Transform(bool \*Out, bool \*In, const char \*Table, int len) {

static char Temp[256];

for (int i = 0; i < len; i++)

Temp[i] = In[Table[i] - 1];//取Table[i]个元素

memcpy(Out, Temp, len);

}

1. **字节转换为位**

static void ByteToBit(bool \*Out, const char \*In, int bits) {

for (int i = 0; i < bits; ++i)

Out[i] = (In[i >> 3] >> (i & 7)) & 1;//确定每一位来自哪个字符；确定是哪一位，移动至末尾；与1相与，取出结果

}

1. **加密按钮事件**

void CDESYangjinnengDlg::OnBnClickedButton1()

{

// TODO: 在此添加控件通知处理程序代码

//读入明文并记录长度

CString message;

GetDlgItemText(IDC\_EDIT1, message);

int size = message.GetLength();

block = (size - 1) / 8 + 1;//经填充后的分组长度

//读入密钥并设置子密钥

CString k;

GetDlgItemText(IDC\_EDIT2, k);

char\* key = k.GetBuffer(0);//转换成字符数组类型

if (strlen(key) < 8) {

AfxMessageBox(\_T("密钥长度必须8位！"));

}

else {

SetSubKey(key);

//分组加密

char m\_block[8] = { 0 };//明文分组64bit

char c\_block[8] = { 0 };//密文分组

for (int i = 0; i < block - 1; i++) {

for (int j = 0; j < 8; j++)

m\_block[j] = message.GetAt(8 \* i + j);//把明文对应位置字符放到明文分组中

DES\_Yangjinneng(c\_block, m\_block, 1);

for (int j = 0; j < 8; j++)

c[8 \* i + j] = c\_block[j]; //把密文分组放入全局变量密文

}

//最后一组进行填充和加密

for (int j = 0; j < 8; j++)

m\_block[j] = 0;//补0

for (int j = 0; j < (size - 1) % 8 + 1; j++)

m\_block[j] = message.GetAt(8 \* (block - 1) + j);

DES\_Yangjinneng(c\_block, m\_block, 1);

for (int j = 0; j < 8; j++)

c[8 \* (block - 1) + j] = c\_block[j];

//显示密文

CString cipher = c;

SetDlgItemText(IDC\_EDIT3, cipher);

//密文写入cipher.txt

fstream file1;

file1.open("C://Users//杨//Desktop//cipher.txt", ios::binary | ios::out);

file1.write(c, sizeof(c));

file1.close();

}

}

1. **解密按钮事件**

void CDESYangjinnengDlg::OnBnClickedButton2()

{

// TODO: 在此添加控件通知处理程序代码

//分组解密

char m\_block[8] = { 0 };

char c\_block[8] = { 0 };

for (int i = 0; i < block; i++) {

for (int j = 0; j < 8; j++)

c\_block[j] = c[8 \* i + j];

DES\_Yangjinneng(m\_block, c\_block, 0);

for (int j = 0; j < 8; j++)

m1[8 \* i + j] = m\_block[j];

}

//显示解密文

CString me=m1;

SetDlgItemText(IDC\_EDIT4, me);

}

1. **文件加密**

void CYangjinnengDESDlg::OnBnClickedButton4()

{

// TODO: 在此添加控件通知处理程序代码

CString k;

GetDlgItemText(IDC\_EDIT2, k);

char\* key = k.GetBuffer(0);//转换成字符数组类型

if (strlen(key) < 8) {

AfxMessageBox(\_T("密钥长度必须8位！"));

}

else {

SetSubKey(key);

CFileDialog dlg(TRUE/\*这个参数为TRUE就是“打开”对话框，为FALSE就是“保存”对话框\*/, NULL/\*默认文件类型\*/, NULL/\*默认文件名\*/, OFN\_HIDEREADONLY/\*样式，这里设置为“隐藏只读”\*/, \_T("所有文件(\*.\*)|\*.\*||")/\*文件类型列表\*/, NULL, NULL, FALSE/\*指定文件打开对话框是否为Vista样式\*/);

if (dlg.DoModal() == IDOK)

{

CString strPathName = dlg.GetPathName();

way1.SetWindowText(strPathName);

}

CString path, path1;

GetDlgItemText(IDC\_EDIT5, path);

GetDlgItemText(IDC\_EDIT7, path1);

ifstream in;

ofstream out;

in.open(path, ios::binary);

out.open(path1, ios::binary);

char plain[8];

while (in.read(plain, sizeof(plain))) {

DES\_Yangjinneng(cipher, plain, 1);

out.write(cipher, sizeof(cipher));

}

in.close();

out.close();

}

}

1. **文件解密**

void CYangjinnengDESDlg::OnBnClickedButton5()

{

// TODO: 在此添加控件通知处理程序代码

//读入密钥并设置子密钥

CString k;

GetDlgItemText(IDC\_EDIT2, k);

char\* key = k.GetBuffer(0);//转换成字符数组类型

if (strlen(key) < 8) {

AfxMessageBox(\_T("密钥长度必须8位！"));

}

else {

SetSubKey(key);

char plain[8];

CFileDialog dlg(TRUE/\*这个参数为TRUE就是“打开”对话框，为FALSE就是“保存”对话框\*/, NULL/\*默认文件类型\*/, NULL/\*默认文件名\*/, OFN\_HIDEREADONLY/\*样式，这里设置为“隐藏只读”\*/, \_T("所有文件(\*.\*)|\*.\*||")/\*文件类型列表\*/, NULL, NULL, FALSE/\*指定文件打开对话框是否为Vista样式\*/);

if (dlg.DoModal() == IDOK)

{

CString strPathName = dlg.GetPathName();

way2.SetWindowText(strPathName);

}

CString path, path1;

GetDlgItemText(IDC\_EDIT7, path);

GetDlgItemText(IDC\_EDIT6, path1);

ifstream in;

ofstream out;

in.open(path, ios::binary);

out.open(path1, ios::binary);

while (in.read(plain, sizeof(plain))) {

DES\_Yangjinneng(temp, plain, 0);

out.write(temp, sizeof(temp));

}

in.close();

out.close();

}

}

1. **扩展要求实现**

**要求：考虑使用DES实现文件加解密。**

**实现：（1）利用CFileDialog函数调用资源管理器实现对文件的选择功能。**

CFileDialog dlg(TRUE/\*这个参数为TRUE就是“打开”对话框，为FALSE就是“保存”对话框\*/, NULL/\*默认文件类型\*/, NULL/\*默认文件名\*/, OFN\_HIDEREADONLY/\*样式，这里设置为“隐藏只读”\*/, \_T("所有文件(\*.\*)|\*.\*||")/\*文件类型列表\*/, NULL, NULL, FALSE/\*指定文件打开对话框是否为Vista样式\*/);

1. **用资源管理器得到文件路径后，用ifstream和ofstream来打开要加密的文件和加密后的文件。**

GetDlgItemText(IDC\_EDIT5, path);

GetDlgItemText(IDC\_EDIT7, path1);

ifstream in;

ofstream out;

in.open(path, ios::binary);

out.open(path1, ios::binary);

1. **对文件进行DES加解密后将结果文件存放到保存地址中。**

while (in.read(plain, sizeof(plain))) {

DES\_Yangjinneng(cipher, plain, 1);

out.write(cipher, sizeof(cipher));

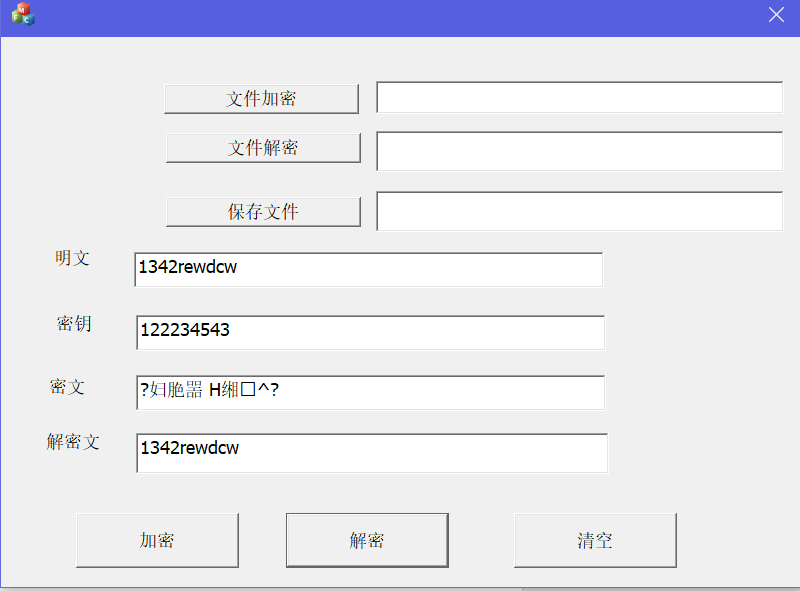
}

in.close();

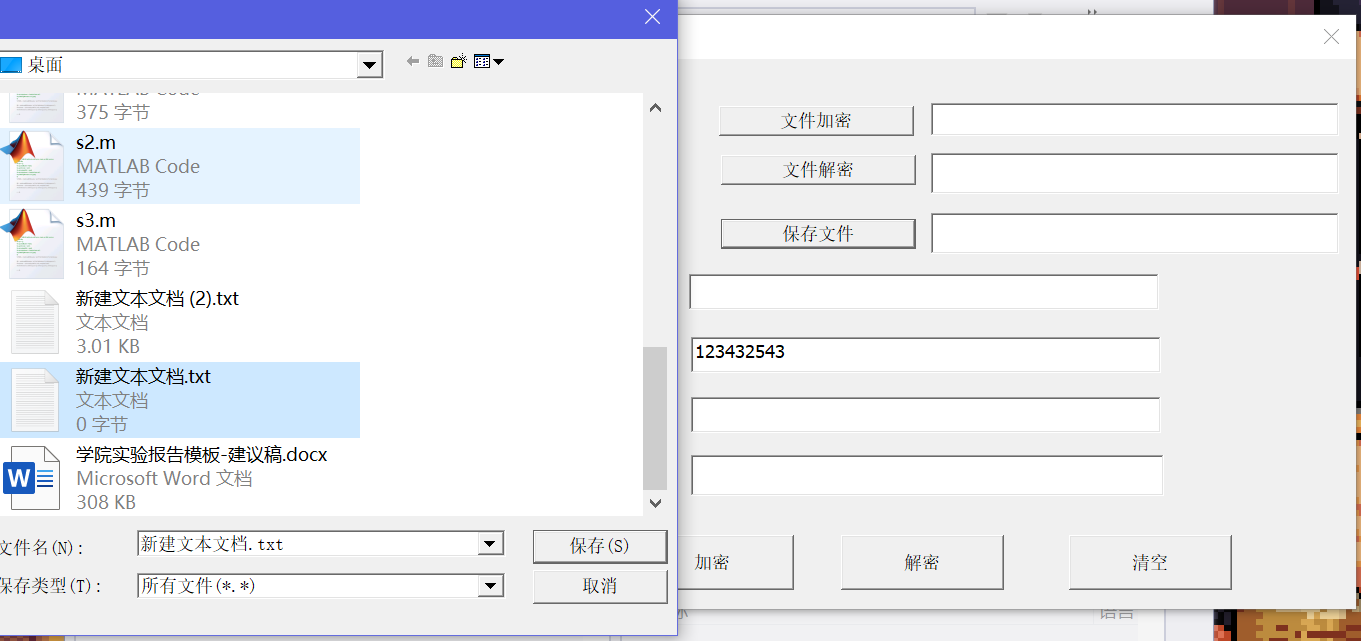
out.close();

1. **运行截图**

**对明文进行加密：**

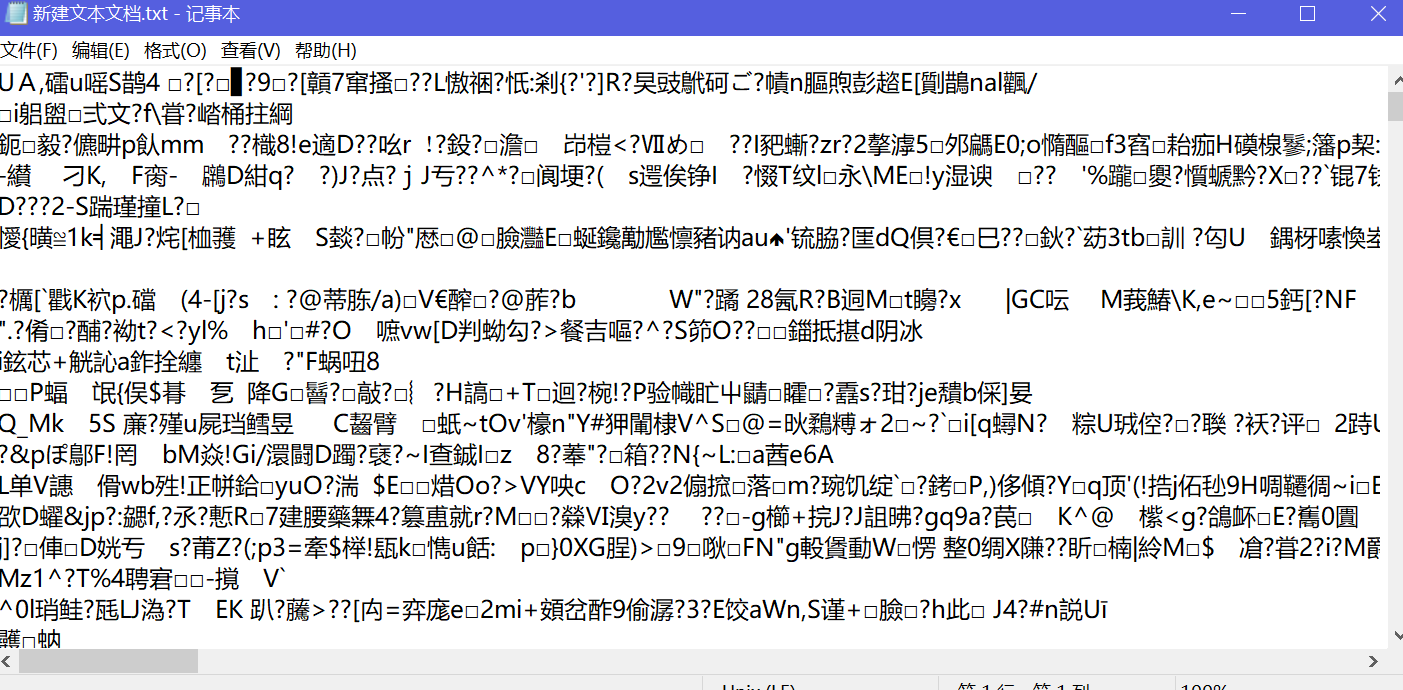


**对文件进行加密：**

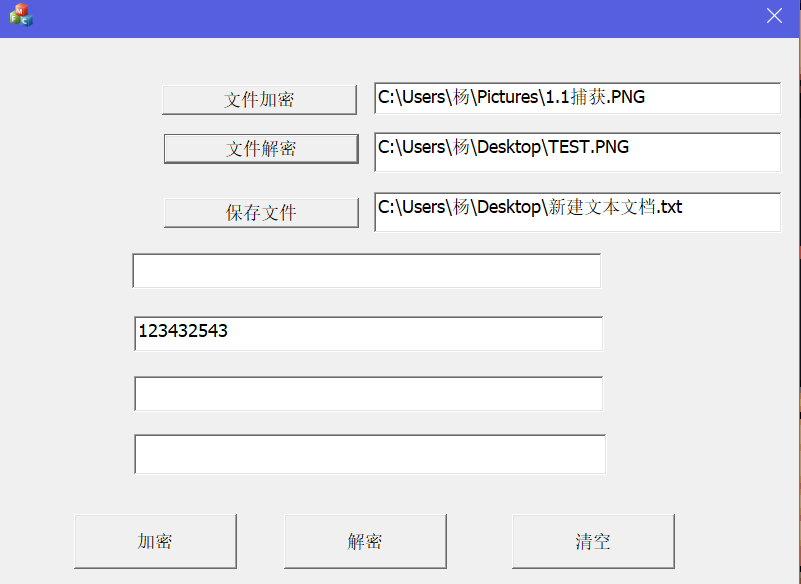
****

****

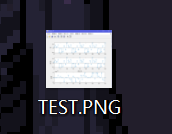
**加密结果：**

****

**文件解密：**

****

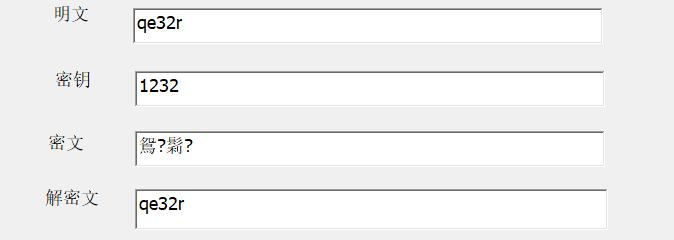
**解密结果：**

****

1. **实验分析总结及心得**

**1、遇到的问题和解决方法：**

**（1）问题：运行程序进行加解密时，没有限制密钥为8个字节，导致无论密钥长度多长都有结果输出。**

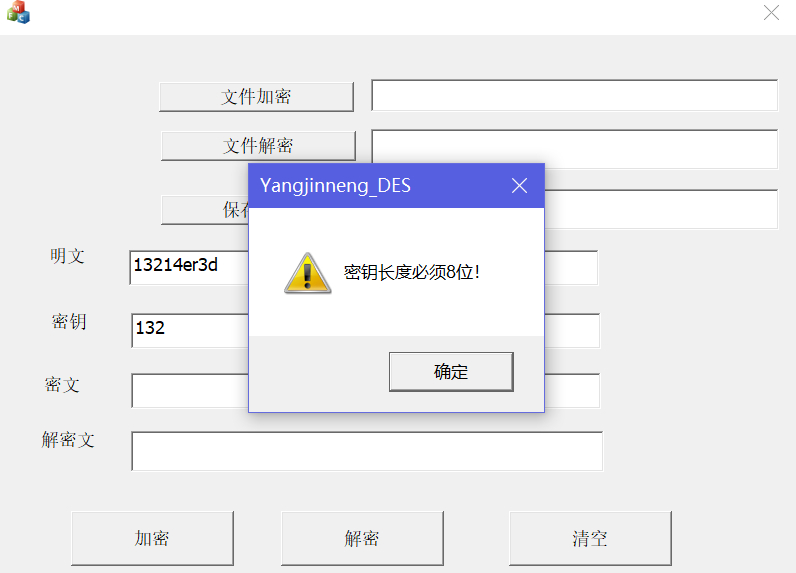


**解决方法：添加if语句对密钥长度进行检查，若少于8个字节则给予弹窗警告，多于8个字节则只取前8个。**

if (strlen(key) < 8) {

AfxMessageBox(\_T("密钥长度必须8位！"));

}



**（2）问题：刚开始进行文件加解密时，我采用的是直接在程序中写出目标路径的方式，不能自由进行选择。要改加密文件路径或解密目标路径必须在程序中手动更改。**

**加密：**

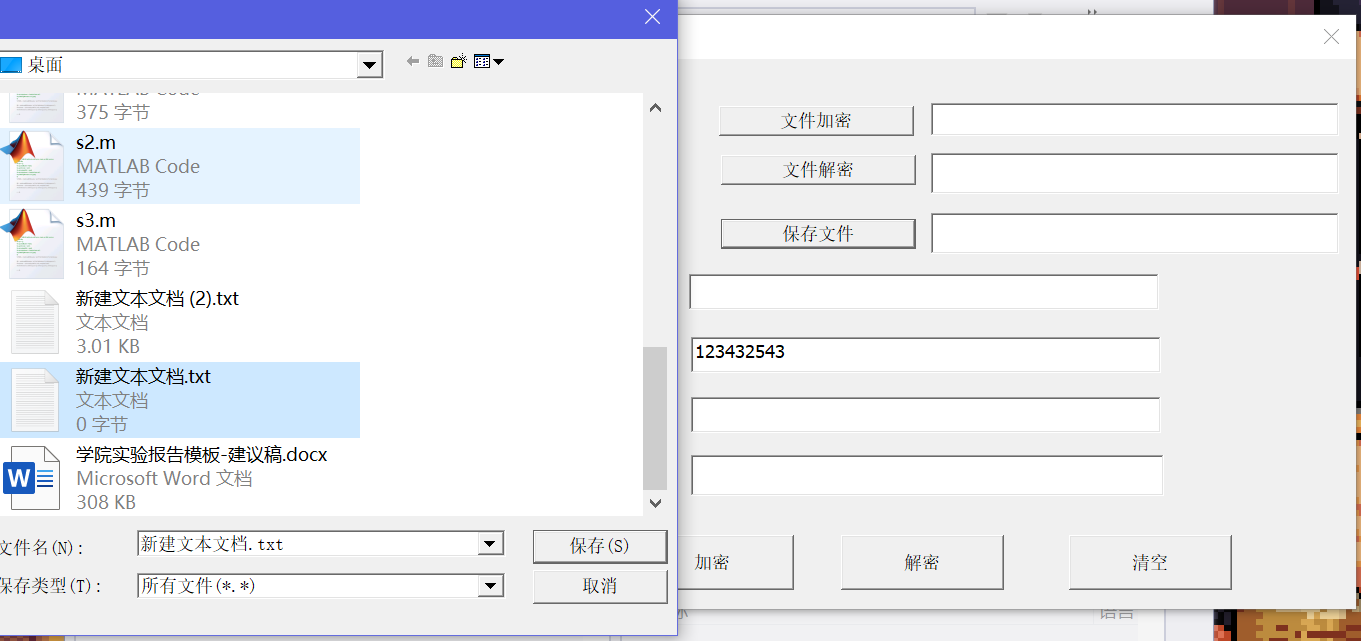
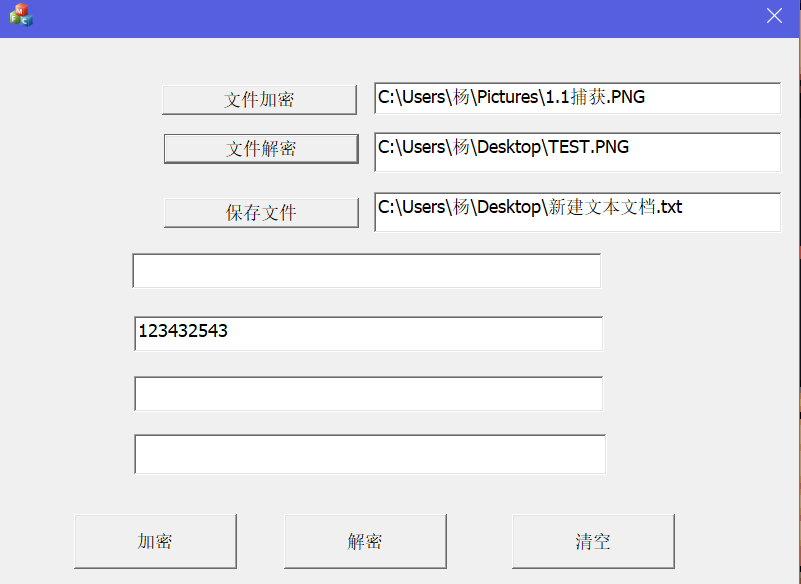
****

**解密：**

****

**解决方法：**

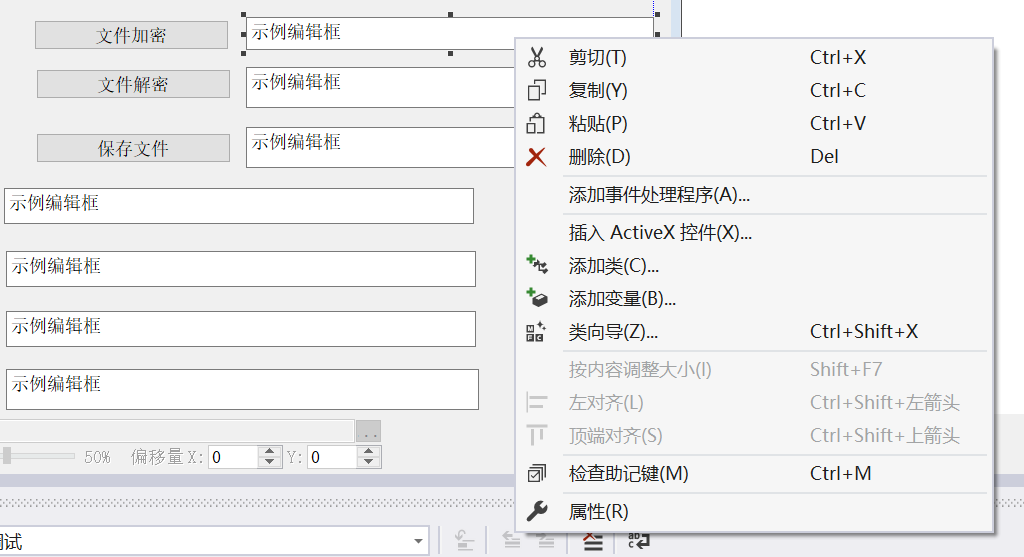
**使用CFileDialog函数，分别为加解密文件和保存文件弹出“打开”和“保存”对话框，并将选择后的文件路径写入文本框中，然后用ifstream和outstream对目标文件进行读取并加密，并保存在对应路径中。**

**** ****

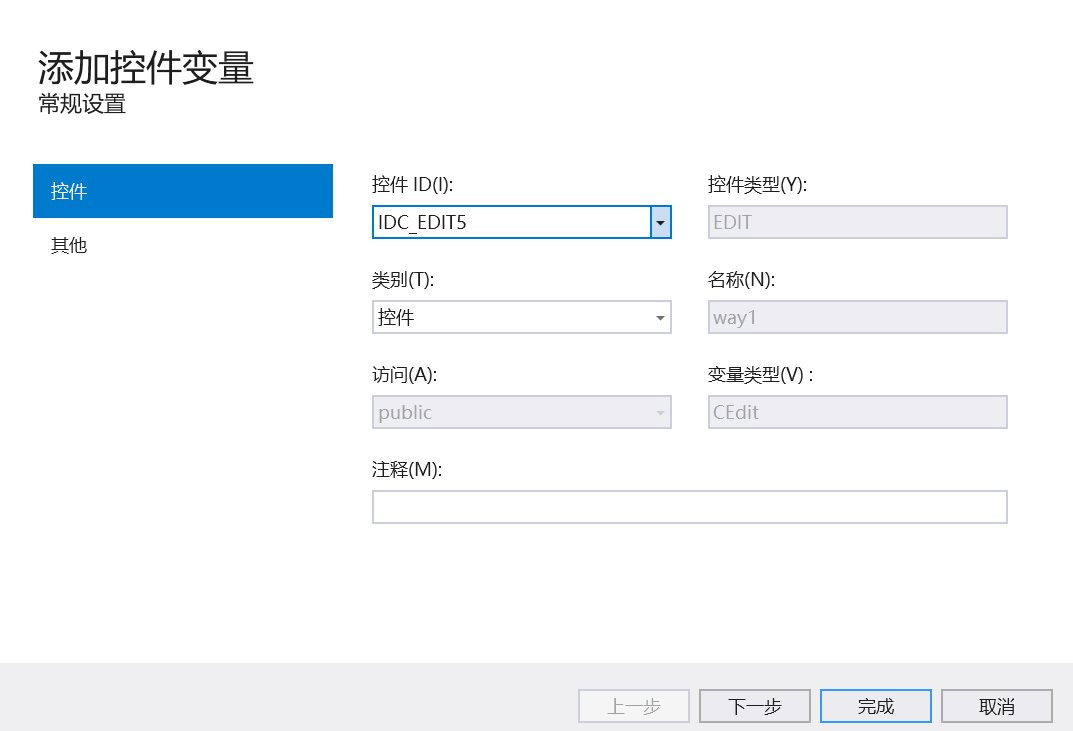
**（3）问题：在对话框交互中，用GetPathName()函数获取到选择文件路径后，不知道该怎么把路径赋值到文本框中。这个问题真是困扰了我好久。**



**解决方法：在对话框控件中，右键选中相应的文本框，选中“添加变量”，**



**在名称中输入变量名：**



**然后就可以在程序中直接对变量进行操作，如用SetWindowText把文本输入文本框中。**



**这个技巧救了我呀，不然真不知道要怎么继续。**

**2、 总结经验：**

（1）对于MFC来说，知道如何去管理和操作相应的控件是很重要的。如SetDlgItemText、GetDlgItemText等函数负责对文本框进行写入和读出其中数据；AfxMessageBox、CFileDialog等类负责弹出对话框来和MFC进行交互。学会使用这些会让我们能更好地完成任务。

（2）无论是Caesar密码还是DES加密算法，尽管它们的加密方式不同，但我们要想实现它们，都需要首先了解它们的整体逻辑结构，然后根据它们的逻辑结构找出要实现它们所需的函数，然后采用模块化编程，分别编写出各个子函数功能，最后在用主函数把它们连接起来，输出最终结果。