

密码学

平时实验报告

（ 2020 / 2021学年 第 二 学期）

题 目：分组密码实验

|  |  |
| --- | --- |
| **专 业** | 信息安全 |
| **组 长 姓 名** |  |
| **组 员 姓 名** |  |
| **班 级 学 号** |  |
| **指 导 教 师** |  |
| **指 导 单 位** | 信息安全系 |
| **日 期** | 2021年4月15日 |

|  |  |
| --- | --- |
| **成员分工** | |
| 组长（1） | 任远哲，B19031614，编写Hill加解密，撰写报告 |
| 组员（1） | 戴俊皓, B19030733,编写仿射加解密 |
| 组员（2） | 郭稼逸,B19030729，编写置换加解密 |
| **简短评语：**  **指导教师：年月日** | |

1. **课题内容和要求**

**1.实验环境**

实验主机操作系统为Windows 7

**2.实验内容**

实现分组密码AES

**二、课题需求分析**

对一个 128 位数据进行加密和解密。使用的密钥长度可以为 **128，192** 或 **256 bits**，这里选择**128bit。**



根据标准，密钥长度NK为4个字，分组长度Nb为4个字，加密轮数为10轮。

1. 密钥扩展

密钥按照矩阵列进行分组，如果前4列为k(0),k(1),k(2),k(3),那么新的列如下递归：

如果k(i)中，i不是4的倍数，那么i列由如下等式确定：

k(i)=k(i-4) XOR k(i-1)

如果k(i)中，i是4的倍数，那么i列由如下等式确定：

k(i)=k(i-4) XOR T[k(i-1)]

其中T[k(i-1)]是k(i-1)的一种转换形式，按以下方式实现：

循环地将k(i-1)的元素左移位，每次一个字节，如 abcd 变成bcda ，将这4个字节作为S盒的输入，输出新的4个字节efgh ，计算一轮的常量 r(i)=2 (i-1)/4，这样生成转换后的列：[e XOR r(i),f,g,h]

2． 加密

AES算法主要有四种操作处理，分别是轮密钥加、字节代换、行位移层、列混淆层。而明文x和密钥k都是由16个字节组成的数据，它是按照字节的先后顺序从上到下、从左到右进行排列的。AES算法在处理的轮数上只有最后一轮操作与前面的轮处理上有些许不同(最后一轮只是少了列混淆处理)，在轮处理开始前还单独进行了一次轮密钥加的处理。

2.1轮密钥加变换 AddRoundKey(byte mtx[4\*4], word k[4])，。将轮密钥与状态按比特异或。轮密钥是通过密钥扩展过程从密码密钥中得到的，轮密钥长度等于分组长度。

2.2字节代换SubBytes()，接受一个 4x4 的字节矩阵作为输入，对其中的每个字节，前四位组成十六进制数 x 作为行号，后四位组成的十六进制数 y 作为列号，查找表中对应的值替换原来位置上的字节。

在字节代换步骤中，矩阵中的各字节透过一个16\*16 的 S-box 进行转换。这个步骤提供了加密法非线性的变换能力。S-box 与 GF(28)GF(28) 上的乘法反元素有关，已知具有良好的非线性特性。为了避免简单代数性质的攻击，S-box 结合了乘法反元素及一个可逆的仿射变换矩阵建构而成。此外在建构 S-box 时，刻意避开了固定点与反固定点，即以 S-box 替换字节的结果会相当于错排的结果。

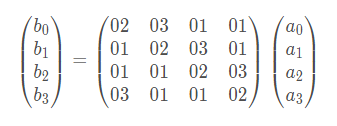
2.3行位移层ShiftRows()

行变换也很简单，它仅仅是将矩阵的每一行以字节为单位循环移位：第一行不变，第二行左移一位，第三行左移两位，第四行左移三位。

2.4列混淆层MixColumns()

函数MixColumns()同样接受一个 4x4 的字节矩阵作为输入，并对矩阵进行逐列变换

将状态矩阵的每一列看成G F ( 2^8)上的一个多项式，且与一个固定的多项式a(x)={03} x3+{01} x2+{01}x +{02} 相乘后模(x4 + 1)。把上述结果写成矩阵形式，就是



3．解密

分别实现 S 盒变换、行变换和列变换的逆变换InvSubBytes()、InvShiftRows()和InvMixColumns()。

**完整代码与注释**

#include <iostream>

#include <bitset>

#include <string>

using namespace std;

typedef bitset<8> byte;

typedef bitset<32> word;

const int Nr = 10; // AES-128需要 10 轮加密

const int Nk = 4; // Nk 表示输入密钥的 word 个数

byte S\_Box[16][16] = {

{0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5,

0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76},

{0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0,

0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0},

{0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC,

0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15},

{0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A,

0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75},

{0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0,

0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84},

{0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B,

0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF},

{0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85,

0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8},

{0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5,

0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2},

{0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17,

0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73},

{0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88,

0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB},

{0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C,

0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79},

{0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9,

0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08},

{0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6,

0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A},

{0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E,

0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E},

{0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94,

0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF},

{0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68,

0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16}

};

byte Inv\_S\_Box[16][16] = {

{0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38,

0xBF, 0x40, 0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB},

{0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87,

0x34, 0x8E, 0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB},

{0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D,

0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E},

{0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2,

0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25},

{0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16,

0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92},

{0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA,

0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84},

{0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A,

0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06},

{0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02,

0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B},

{0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA,

0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73},

{0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85,

0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E},

{0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89,

0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B},

{0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20,

0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4},

{0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31,

0xB1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F},

{0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D,

0x2D, 0xE5, 0x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF},

{0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0,

0xC8, 0xEB, 0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61},

{0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26,

0xE1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D}

};

// 轮常数，密钥扩展中用到。（AES-128只需要10轮）

word Rcon[10] = {0x01000000, 0x02000000, 0x04000000, 0x08000000, 0x10000000,

0x20000000, 0x40000000, 0x80000000, 0x1b000000, 0x36000000};

/\*\*

\* S盒变换 - 前4位为行号，后4位为列号

\*/

void SubBytes(byte mtx[4\*4])

{

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

{

int row = mtx[i][7]\*8 + mtx[i][6]\*4 + mtx[i][5]\*2 + mtx[i][4];

int col = mtx[i][3]\*8 + mtx[i][2]\*4 + mtx[i][1]\*2 + mtx[i][0];

mtx[i] = S\_Box[row][col];

}

}

/\*\*

\* 行变换 - 按字节循环移位

\*/

void ShiftRows(byte mtx[4\*4])

{

// 第二行循环左移一位

byte temp = mtx[4];

for(int i=0; i<3; ++i)

mtx[i+4] = mtx[i+5];

mtx[7] = temp;

// 第三行循环左移两位

for(int i=0; i<2; ++i)

{

temp = mtx[i+8];

mtx[i+8] = mtx[i+10];

mtx[i+10] = temp;

}

// 第四行循环左移三位

temp = mtx[15];

for(int i=3; i>0; --i)

mtx[i+12] = mtx[i+11];

mtx[12] = temp;

}

/\*\*

\* 有限域上的乘法 GF(2^8)

\*/

byte GFMul(byte a, byte b) {

byte p = 0;

byte hi\_bit\_set;

for (int counter = 0; counter < 8; counter++) {

if ((b & byte(1)) != 0) {

p ^= a;

}

hi\_bit\_set = (byte) (a & byte(0x80));

a <<= 1;

if (hi\_bit\_set != 0) {

a ^= 0x1b; /\* x^8 + x^4 + x^3 + x + 1 \*/

}

b >>= 1;

}

return p;

}

/\*\*

\* 列变换

\*/

void MixColumns(byte mtx[4\*4])

{

byte arr[4];

for(int i=0; i<4; ++i)

{

for(int j=0; j<4; ++j)

arr[j] = mtx[i+j\*4];

mtx[i] = GFMul(0x02, arr[0]) ^ GFMul(0x03, arr[1]) ^ arr[2] ^ arr[3];

mtx[i+4] = arr[0] ^ GFMul(0x02, arr[1]) ^ GFMul(0x03, arr[2]) ^ arr[3];

mtx[i+8] = arr[0] ^ arr[1] ^ GFMul(0x02, arr[2]) ^ GFMul(0x03, arr[3]);

mtx[i+12] = GFMul(0x03, arr[0]) ^ arr[1] ^ arr[2] ^ GFMul(0x02, arr[3]);

}

}

/\*\*

\* 轮密钥加变换 - 将每一列与扩展密钥进行异或

\*/

void AddRoundKey(byte mtx[4\*4], word k[4])

{

for(int i=0; i<4; ++i)

{

word k1 = k[i] >> 24;

word k2 = (k[i] << 8) >> 24;

word k3 = (k[i] << 16) >> 24;

word k4 = (k[i] << 24) >> 24;

mtx[i] = mtx[i] ^ byte(k1.to\_ulong());

mtx[i+4] = mtx[i+4] ^ byte(k2.to\_ulong());

mtx[i+8] = mtx[i+8] ^ byte(k3.to\_ulong());

mtx[i+12] = mtx[i+12] ^ byte(k4.to\_ulong());

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*下面是解密的逆变换函数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* 逆S盒变换

\*/

void InvSubBytes(byte mtx[4\*4])

{

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

{

int row = mtx[i][7]\*8 + mtx[i][6]\*4 + mtx[i][5]\*2 + mtx[i][4];

int col = mtx[i][3]\*8 + mtx[i][2]\*4 + mtx[i][1]\*2 + mtx[i][0];

mtx[i] = Inv\_S\_Box[row][col];

}

}

/\*\*

\* 逆行变换 - 以字节为单位循环右移

\*/

void InvShiftRows(byte mtx[4\*4])

{

// 第二行循环右移一位

byte temp = mtx[7];

for(int i=3; i>0; --i)

mtx[i+4] = mtx[i+3];

mtx[4] = temp;

// 第三行循环右移两位

for(int i=0; i<2; ++i)

{

temp = mtx[i+8];

mtx[i+8] = mtx[i+10];

mtx[i+10] = temp;

}

// 第四行循环右移三位

temp = mtx[12];

for(int i=0; i<3; ++i)

mtx[i+12] = mtx[i+13];

mtx[15] = temp;

}

void InvMixColumns(byte mtx[4\*4])

{

byte arr[4];

for(int i=0; i<4; ++i)

{

for(int j=0; j<4; ++j)

arr[j] = mtx[i+j\*4];

mtx[i] = GFMul(0x0e, arr[0]) ^ GFMul(0x0b, arr[1])

^ GFMul(0x0d, arr[2]) ^ GFMul(0x09, arr[3]);

mtx[i+4] = GFMul(0x09, arr[0]) ^ GFMul(0x0e, arr[1])

^ GFMul(0x0b, arr[2]) ^ GFMul(0x0d, arr[3]);

mtx[i+8] = GFMul(0x0d, arr[0]) ^ GFMul(0x09, arr[1])

^ GFMul(0x0e, arr[2]) ^ GFMul(0x0b, arr[3]);

mtx[i+12] = GFMul(0x0b, arr[0]) ^ GFMul(0x0d, arr[1])

^ GFMul(0x09, arr[2]) ^ GFMul(0x0e, arr[3]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*下面是密钥扩展部分\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* 将4个 byte 转换为一个 word.

\*/

word Word(byte& k1, byte& k2, byte& k3, byte& k4)

{

word result(0x00000000);

word temp;

temp = k1.to\_ulong(); // K1

temp <<= 24;

result |= temp;

temp = k2.to\_ulong(); // K2

temp <<= 16;

result |= temp;

temp = k3.to\_ulong(); // K3

temp <<= 8;

result |= temp;

temp = k4.to\_ulong(); // K4

result |= temp;

return result;

}

/\*\*

\* 按字节 循环左移一位

\* 即把[a0, a1, a2, a3]变成[a1, a2, a3, a0]

\*/

word RotWord(word& rw)

{

word high = rw << 8;

word low = rw >> 24;

return high | low;

}

/\*\*

\* 对输入word中的每一个字节进行S-盒变换

\*/

word SubWord(const word& sw)

{

word temp;

for(int i=0; i<32; i+=8)

{

int row = sw[i+7]\*8 + sw[i+6]\*4 + sw[i+5]\*2 + sw[i+4];

int col = sw[i+3]\*8 + sw[i+2]\*4 + sw[i+1]\*2 + sw[i];

byte val = S\_Box[row][col];

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

temp[i+j] = val[j];

}

return temp;

}

/\*\*

\* 密钥扩展函数 - 对128位密钥进行扩展得到 w[4\*(Nr+1)]

\*/

void KeyExpansion(byte key[4\*Nk], word w[4\*(Nr+1)])

{

word temp;

int i = 0;

// w[]的前4个就是输入的key

while(i < Nk)

{

w[i] = Word(key[4\*i], key[4\*i+1], key[4\*i+2], key[4\*i+3]);

++i;

}

i = Nk;

while(i < 4\*(Nr+1))

{

temp = w[i-1]; // 记录前一个word

if(i % Nk == 0)

w[i] = w[i-Nk] ^ SubWord(RotWord(temp)) ^ Rcon[i/Nk-1];

else

w[i] = w[i-Nk] ^ temp;

++i;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*下面是加密和解密函数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* 加密

\*/

void encrypt(byte in[4\*4], word w[4\*(Nr+1)])

{

word key[4];

for(int i=0; i<4; ++i)

key[i] = w[i];

AddRoundKey(in, key);

for(int round=1; round<Nr; ++round)

{

SubBytes(in);

ShiftRows(in);

MixColumns(in);

for(int i=0; i<4; ++i)

key[i] = w[4\*round+i];

AddRoundKey(in, key);

}

SubBytes(in);

ShiftRows(in);

for(int i=0; i<4; ++i)

key[i] = w[4\*Nr+i];

AddRoundKey(in, key);

}

/\*\*

\* 解密

\*/

void decrypt(byte in[4\*4], word w[4\*(Nr+1)])

{

word key[4];

for(int i=0; i<4; ++i)

key[i] = w[4\*Nr+i];

AddRoundKey(in, key);

for(int round=Nr-1; round>0; --round)

{

InvShiftRows(in);

InvSubBytes(in);

for(int i=0; i<4; ++i)

key[i] = w[4\*round+i];

AddRoundKey(in, key);

InvMixColumns(in);

}

InvShiftRows(in);

InvSubBytes(in);

for(int i=0; i<4; ++i)

key[i] = w[i];

AddRoundKey(in, key);

}

int main()

{

byte key[16] = {0x3c, 0x57, 0x90, 0xac,

0xa1, 0xf0, 0x2e, 0xc1,

0x0b, 0x19, 0x13, 0x07,

0x21, 0x16, 0x80, 0xbd};

byte plain[16] = {0x32, 0x88, 0x31, 0xe0,

0x43, 0x5a, 0x31, 0x37,

0xf6, 0x30, 0x98, 0x07,

0xa8, 0x8d, 0xa2, 0x34};

// 输出密钥

cout << "密钥是：";

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

cout << hex << key[i].to\_ulong() << " ";

cout << endl;

word w[4\*(Nr+1)];

KeyExpansion(key, w);

// 输出待加密的明文

cout << endl << "待加密的明文："<<endl;

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

{

cout << hex << plain[i].to\_ulong() << " ";

if((i+1)%4 == 0)

cout << endl;

}

cout << endl;

// 加密，输出密文

encrypt(plain, w);

cout << "加密后的密文："<<endl;

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

{

cout << hex << plain[i].to\_ulong() << " ";

if((i+1)%4 == 0)

cout << endl;

}

cout << endl;

// 解密，输出明文

decrypt(plain, w);

cout << "解密后的明文："<<endl;

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

{

cout << hex << plain[i].to\_ulong() << " ";

if((i+1)%4 == 0)

cout << endl;

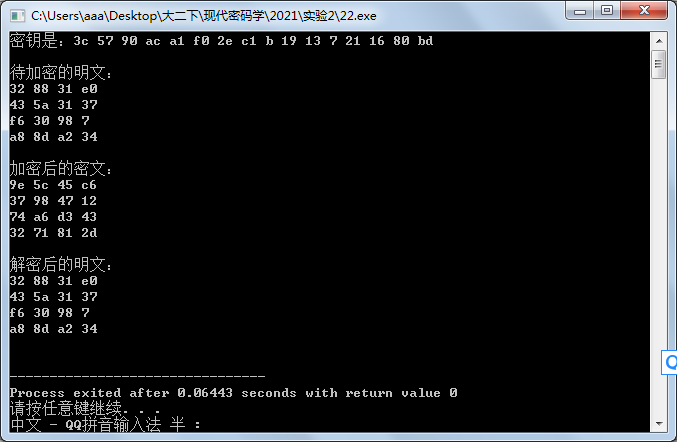
}

cout << endl;

return 0;

}

**三.测试数据及其结果分析**

** 实验运行结果**

**四、课题完成过程中遇到的问题及解决方法**

问题：伽罗华域GF上的乘法不会写

解决方法：查阅资料，找到一种叫做[peasant's algorithm](https://en.wikipedia.org/wiki/Multiplication_algorithm#Binary_or_Peasant_multiplication)的算法

**五、总结**

AES加密属于分组密码，明文和密文的长度128bit，密钥长度可变（128bit,192bit,256bit等，现采用128bit）。面向二进制的密码算法，能够加解密任何形式的计算机数据

不是对合运算，加解密使用不同的算法，巧妙的是其结构相似，可以即保证其安全性又可方便操作。综合运用多种密码技术，置换，代替，代数。整体结构：SPN结构（S为非线性部件，是安全性的最重要部分）， 基本轮函数迭代，迭代轮数可变（≥10）

经过本次实验，我深刻理解了分组密码中的AES密码的基本步骤。纸上得来终觉浅，绝知此事要躬行，理论上的推导和如何把理论转换成代码都是重要的能力，把人脑理解的步骤转换成机器能识别的选择分支循环逻辑有一定的难度，这也是我们计算机专业的学生应该重点培养的。