MD5 算法

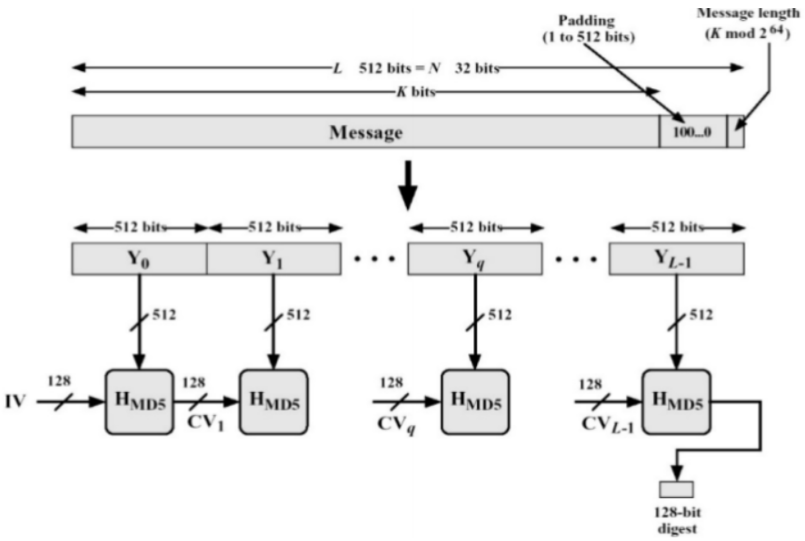
MD5 即 Message-Digest Algorithm 5 (信息-摘要算法 5) MD4 (1990)、MD5(1992, RFC 1321) 作者 Ron Rivest,是广泛使用的散列算法,**经常用于确保信息传输的完整性和一致性。**MD5 使用 little-endian,输入任意不定长度信息,以512位长进行分组,生成四个32位数据,最后联合起来输出固定128位长的信息摘要。MD5 算法的基本过程为:求余、取余、调整长度、与链接变量进行循环运算、得出结果。

- MD5 不是足够安全的
    - Hans Dobbertin 在1996年找到了两个不同的 512-bit 块,它们在 MD5 计算下产生相同的 hash 值。

MD5是一个hash算法，它通过吧任意长度的消息哈希成一个128位长的字符串，但是人们很难将这128位的字符串解出原来的字符串，因为有很多种组合的方式，而且在大部分的情况来看，至今还没有真正找到两个不同的消息,它们的 MD5 的 hash值相等。所以说它是可以保护密码的安全的。

1. 算法原理概述

1.1 算法原理图



1.2 自然语言描述

对MD5算法简要的叙述可以为：MD5以 512 位分组来处理输入的信息，且每一分组又被划分为16个32位子分组，经过了一系列的处理后，算法的输出由四个 32 位分组组成，将这四个 32 位分组级联后将生成一个 128 位散列值，其具体步骤为：

- step 1 -> 填充数据
- step 2 -> 记录输入信息长度
- step 3 -> 装入标准的幻数 A B C D
- step 4 -> 设置轮转算式，完成循环运算
- step 5 -> 拼凑A B C D ， 完成输出

2. 总体结构

- `main.cpp`：测试MD5算法有效性文件，来源于`RCF 1321`

- `MD5.h`: 定义了MD5类以及宏定义了一些计算方法
- `MD5.cpp`：实现了`MD5.h`中的类方法，包括

```
//声明构造函数
MD5(const string &str);
//初始化
void init() ;
//MD5算法操作核心
const string getMessage();
//补位操作
void Padding();
//这里完成数据的大端存储
void binarySort();
//增加输入数据长度
void appendLength();
//解码
void Decode(int begin, bit32* x);
//转32位bool类型vector<bool>为 bit32类型
bit32 convert(const vector<bool>& a);
//轮转操作函数
void transform(int beginIndex);
//转字符串输出
const string toStr();
```

3. 模块分解

3.1 标准输入模块

在main函数中，我选择使用了`RCF 1321`中给出的标准测试样例进行测试，这样做的好处在于，标准测试样例已经给出了正确的加密结果，只需要将所求的与已知的进行对比，即可得到答案。

```
// 标准输入:
in[7] = {
        "",
        "a",
        "abc",
        "message digest",
        "abcdefghijklmnopqrstuvwxyz",
        "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789",

 "12345678901234567890123456789012345678901234567890123456789012345678901234567890"};
// 标准输出为
 out[7] = {
        "d41d8cd98f00b204e9800998ecf8427e", "0cc175b9c0f1b6a831c399e269772661",
        "900150983cd24fb0d6963f7d28e17f72", "f96b697d7cb7938d525a2f31aaf161d0",
        "c3fcd3d76192e4007dfb496cca67e13b", "d174ab98d277d9f5a5611c2c9f419d9f",
        "57edf4a22be3c955ac49da2e2107b67a"};
```

3.2 init 模块

由于在测试程序中需要多次建立新的对象，所以我们需要对上一次运算的结果进行清除，而初始化模块就是将每一次的运算的前一次过程进行清除，保证运算结果的正确性。

```
 //先清除原先存在输入信息
binary_input.clear();
for(int i = 0 ; i < input.size() ; i ++ )
{
    auto temp = bitset<8> (input[i]);
    for(int j = 7 ; j >= 0 ; j -- )
    {
        binary_input.push_back(temp[j]) ;
    }
}
//清除原先存在的长度二进制信息
binary_length.clear();
length = binary_input.size();
bitset<64> temp(length) ;
for(int i = 63 ; i >= 0 ; i -- )
{
    binary_length.push_back(temp[i]);
}
```

3.3 Padding 模块

在 Padding 模块中，要做的工作就是补位（不包括加上数据长度的64位），若输入信息的长度对 512 求余的结果不等于 448，就需要填充使得对 512 求余的结果等于448。填充的方法是填充一个 1 和 n 个 0 。填充完后，信息的长度就为`N * 512 + 448`

```
int difference = 0 ;
if( length % 512 < 448)
{
    difference = 448 - length%512 ;
}
else
{
    difference = 960 - length%512 ;
}
//末尾补1 再加0
binary_input.push_back(1);
vector<bool> pad(difference-1,0);
binary_input.insert(binary_input.end(),pad.begin(),pad.end());
```

3.4 appendLength 模块

在 appendLength 模块中，对于原始信息（不包括第一步padding补充的位数）的位数长度b，化成二进制表示后选取低64位，每个word按照小端规则添加到第一步处理后的消息数据的尾部。例如，对于消息"a"，消息长度应该是8（8位代表一个字符串），所以二进制表示是"0000...0001000", 接着补充的情况是"0000...1000 000...000(32个0)"

```
//采用小端存储的方式插入
binary_input.insert(binary_input.end(), binary_length.begin() + 32,
binary_length.end()) ;
    binary_input.insert(binary_input.end(), binary_length.begin(),
binary_length.begin() + 32) ;
```

3.5 transform 模块

在 transform 模块中，使用之前的宏定义完成四轮轮转运算，随后在末尾分别再加上原始数据，即可得到初步加密运算结果A B C D；轮转运算的具体规则如下：

```
 // Round 1
FF(A, B, C, D, x[0], S11, 0xd76aa478);
FF(D, A, B, C, x[1], S12, 0xe8c7b756);
FF(C, D, A, B, x[2], S13, 0x242070db);
FF(B, C, D, A, x[3], S14, 0xc1bdceee);
FF(A, B, C, D, x[4], S11, 0xf57c0faf);
FF(D, A, B, C, x[5], S12, 0x4787c62a);
FF(C, D, A, B, x[6], S13, 0xa8304613);
FF(B, C, D, A, x[7], S14, 0xfd469501);
FF(A, B, C, D, x[8], S11, 0x698098d8);
FF(D, A, B, C, x[9], S12, 0x8b44f7af);
FF(C, D, A, B, x[10], S13, 0xffff5bb1);
FF(B, C, D, A, x[11], S14, 0x895cd7be);
FF(A, B, C, D, x[12], S11, 0x6b901122);
FF(D, A, B, C, x[13], S12, 0xfd987193);
FF(C, D, A, B, x[14], S13, 0xa679438e);
FF(B, C, D, A, x[15], S14, 0x49b40821);

// Round 2
GG(A, B, C, D, x[1], S21, 0xf61e2562);
GG(D, A, B, C, x[6], S22, 0xc040b340);
GG(C, D, A, B, x[11], S23, 0x265e5a51);
GG(B, C, D, A, x[0], S24, 0xe9b6c7aa);
GG(A, B, C, D, x[5], S21, 0xd62f105d);
GG(D, A, B, C, x[10], S22, 0x2441453);
GG(C, D, A, B, x[15], S23, 0xd8a1e681);
GG(B, C, D, A, x[4], S24, 0xe7d3fbc8);
GG(A, B, C, D, x[9], S21, 0x21e1cde6);
GG(D, A, B, C, x[14], S22, 0xc33707d6);
GG(C, D, A, B, x[3], S23, 0xf4d50d87);
GG(B, C, D, A, x[8], S24, 0x455a14ed);
GG(A, B, C, D, x[13], S21, 0xa9e3e905);
GG(D, A, B, C, x[2], S22, 0xfcefa3f8);
GG(C, D, A, B, x[7], S23, 0x676f02d9);
GG(B, C, D, A, x[12], S24, 0x8d2a4c8a);

// Round 3
HH(A, B, C, D, x[5], S31, 0xfffa3942);
HH(D, A, B, C, x[8], S32, 0x8771f681);
HH(C, D, A, B, x[11], S33, 0x6d9d6122);
HH(B, C, D, A, x[14], S34, 0xfde5380c);
```

```
HH(A, B, C, D, x[1], S31, 0xa4beea44);
HH(D, A, B, C, x[4], S32, 0x4bdecfa9);
HH(C, D, A, B, x[7], S33, 0xf6bb4b60);
HH(B, C, D, A, x[10], S34, 0xbebfbc70);
HH(A, B, C, D, x[13], S31, 0x289b7ec6);
HH(D, A, B, C, x[0], S32, 0xeaa127fa);
HH(C, D, A, B, x[3], S33, 0xd4ef3085);
HH(B, C, D, A, x[6], S34, 0x4881d05);
HH(A, B, C, D, x[9], S31, 0xd9d4d039);
HH(D, A, B, C, x[12], S32, 0xe6db99e5);
HH(C, D, A, B, x[15], S33, 0x1fa27cf8);
HH(B, C, D, A, x[2], S34, 0xc4ac5665);

// Round 4
II(A, B, C, D, x[0], S41, 0xf4292244);
II(D, A, B, C, x[7], S42, 0x432aff97);
II(C, D, A, B, x[14], S43, 0xab9423a7);
II(B, C, D, A, x[5], S44, 0xfc93a039);
II(A, B, C, D, x[12], S41, 0x655b59c3);
II(D, A, B, C, x[3], S42, 0x8f0ccc92);
II(C, D, A, B, x[10], S43, 0xffeff47d);
II(B, C, D, A, x[1], S44, 0x85845dd1);
II(A, B, C, D, x[8], S41, 0x6fa87e4f);
II(D, A, B, C, x[15], S42, 0xfe2ce6e0);
II(C, D, A, B, x[6], S43, 0xa3014314);
II(B, C, D, A, x[13], S44, 0x4e0811a1);
II(A, B, C, D, x[4], S41, 0xf7537e82);
II(D, A, B, C, x[11], S42, 0xbd3af235);
II(C, D, A, B, x[2], S43, 0x2ad7d2bb);
II(B, C, D, A, x[9], S44, 0xeb86d391);
```

3.6 output 模块

在最后输出阶段，使用`snprintf`函数对加密完成的数据进行格式规范与处理，将数据从小端存储转移到大端存储，8位8位进行还原，最终得到我们所需要的加密序列

```
for (int i = 0; i < 4; ++i)
{
    for (int j = 0; j < 4; ++j)
    {
        snprintf(buffer, 4, "%02x", input[i] >> j * 8 & 0xff);
        ret += buffer;
    }
}
```

4. 数据结构

4.1 MD5类

这里为了方便多次进行序列的加密测试，我创建了一个**MD5类**，在编写具体方法和数据类型时候对类的定义与使用也进行了复习；选择使用类的好处在于我们可以将需要用到的方法和数据（如`vector`和`bitset`）进行封装，这样就可以进一的保证程序的严整性，同时在测试时候进行初始化操作也会比较快。

```
class MD5
{
    public:
        MD5(const string &str);
        void init() ;
        const string getMessage();
        void Padding();
        void binarySort();
        void appendLength();
        void Decode(int begin, bit32* x);
        bit32 convert(const vector<bool>& a);
        void transform(int beginIndex);
        const string toStr();
    private:
        string input ;
        vector<bool> binary_input ;
        int length ;
        vector<bool> binary_length ;
        bit32 A , B , C , D ;
};
```

4.2 轮转计算数据

在定义轮转方法时候，我直接使用了 `RCF 1321` 中给出的宏定义，使用宏随可以加快程序运行的速度，但是不太方便源码的阅读；刚从 `RCF 1321` 复制过来，自己也是费了一点功夫去理解这段代码，但是在熟悉之后进入编写函数的步骤，这时候自己写代码的效率就可以提高很多。

```
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))
#define ROTATELEFT(num, n) (((num) << (n)) | ((num) >> (32 - (n))))

#define FF(a, b, c, d, x, s, ac)            \
    {                                       \
        (a) += F((b), (c), (d)) + (x) + ac; \
        (a) = ROTATELEFT((a), (s));         \
        (a) += (b);                         \
    }
#define GG(a, b, c, d, x, s, ac)            \
    {                                       \
        (a) += G((b), (c), (d)) + (x) + ac; \
        (a) = ROTATELEFT((a), (s));         \
        (a) += (b);                         \
    }
#define HH(a, b, c, d, x, s, ac)            \
    {                                       \
        (a) += H((b), (c), (d)) + (x) + ac; \
        (a) = ROTATELEFT((a), (s));         \
        (a) += (b);                         \
    }
```

```
#define II(a, b, c, d, x, s, ac)            \
    {                                       \
        (a) += I((b), (c), (d)) + (x) + ac; \
        (a) = ROTATELEFT((a), (s));         \
        (a) += (b);                         \
    }
```

5. 运行结果

main.cpp文件为测试算法正确性的文件，其使用`RCF 1321`中main.c的测试代码，并在此基础上略做修改，进而来验证加密算法的正确性。

程序编译运行结果如图：

这里有个问题：

由于我的函数方法里面使用了**snprintf**方法，在windows环境下编译时即使是加入了头文件**<stdio.h>**的也依旧会有缺少该函数方法的报错，所以改用在linux的命令行环境下进行编译与运行，同时加上 **std=c++11** 的后缀，即可成功运行与编译。

---

6. 源代码

Github

7. 参考文献

- RCF 1321（MD5算法的C源码txt参考）
- bitset的使用方法
- MD5算法的C实现