一、实验目的

熟悉摘要算法SHA-1，SHA是英文Secure Hash Algorithm的缩写。通过运用高级程序设计语言，编程实现SHA-1算法，加深对单向哈希函数进行消息压缩的理解。

二、实验原理

哈希函数又称为杂凑函数、散列函数，它可以将任意长度的消息压缩成某一固定长度的消息摘要。目前使用的大多数哈希函数如SHA-1(Secure Hash Algorithm)都是采用Merkle于1979年提出的基于压缩函数*f*的哈希函数结构，具有单向函数性质的摘要算法。函数*f*的输入消息*M*被分为*l*个分组，*M*0，*M*1，…，*Ml*-1，每个分组的长度为*b*比特。如果最后一个分组的长度不够，需对其进行填充，并且最后一个分组还包括消息*M*的长度值。

SHA-1算法主要适用于数字签名标准DSS(Digital Signature Standard)里面定义的数字签名算法DSA(Digital Signature Algorithm)。SHA-1具有单向函数的特性，即不可以从消息摘要中复原信息，并且两个不同的消息不会产生同样的消息摘要。消息摘要可以用来验证接收到的消息的数据完整性。如果消息在传输的过程中发生变化，那么变化后的消息会产生不同的消息摘要。

对于长度小于264比特的消息，SHA-1算法对消息以512比特的固定长度为单位，分组进行处理。如果原始消息的长度超过了512比特，就把整个消息分成若干个512比特的消息块，分别处理。最终，SHA-1算法产生一个160比特的消息摘要（哈希值）。如果消息长度大于264比特，则以264为模数取模。SHA-1算法包括以下5个步骤：

**(1) 消息填充**

SHA-1算法首先对输入消息进行填充，使得消息长度的比特数模512后余数是448，即消息扩展至*k*\*512+448二进制位，*k*为非负整数。需要注意的是即使消息长度一开始就满足模512后余448的要求，仍然需要填充。具体的填充操作为，先填充一个1，然后再填充若干个0，直到消息长度满足模512后余数为448。因此，填充的比特数在1～512比特之间，最少补1比特，最多补512比特。

【举例1】假设我们对字符串“abc”产生消息摘要，以下说明填充的过程。

首先，将字符串“abc”转换成比特串的形式：

‘a’=97‘b’=98‘c’=99

原始信息：01100001 01100010 01100011

填充第一步：01100001 01100010 01100011 1

首先填充一个“1”

填充第二步：01100001 01100010 01100011 10…..0

然后填充423个“0”

我们把完成填充后的消息用十六进制表示如下：

61626380 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000

现在，消息的长度是448比特了，可以进行下一步操作。

**(2) 填充长度**

所谓填充长度是将原始消息的长度填充到已经进行了填充操作的消息后面。由消息填充的操作可知，经过填充后的最后一个消息块为448比特。对于512比特的数据块，除掉448比特用于放置最后一个消息块，还有64比特用于存放原始消息（填充前的消息）的长度。填充长度后，整个数块长度为*k*\*512+448+64 = (*k*+1)\*512比特。

【举例2】假设对字符串“abc”产生消息摘要，填充长度后，整个数据块变为下面这样（十六进制格式）：

61626380 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

**00000000 00000000 00000000 00000018**

(**3) 初始化MD缓冲区**

SHA-1算法的中间结果和最终结果保存在160比特的缓冲区中。缓冲区用5个32比特长的寄存器（*A*、*B*、*C*、*D*、*E*）表示。每个寄存器占用一个字（每个字占用4个字节，即1 word = 4 bytes）。这些寄存器初始化为下列32比特的整数（十六进制）：

A= 0x67452301

B= 0xEFCDAB89

C= 0x98BADCFE

D= 0x10325476

E= 0xC3D2E1F0.

令H0= A, H1= B, H2= C, H3= D, H4= E。

其中，前四个值与MD5算法中使用的值相同，但在SHA-1算法中这些值按最高有效字节优先的顺序存储数据，即最高有效字节存储在低地址字节位置。SHA-1算法的这种存储顺序类似处理器处理字节与字转换操作的大端字节顺序(BigEndian)。

**(4) 消息压缩**

SHA-1算法的压缩函数由四轮运算组成，每一轮都对寄存器*A*、*B*、*C*、*D*、*E*进行20步迭代运算。这四轮运算结构相同，但每轮使用的逻辑函数不同。每个逻辑函数都操作寄存器*B*，*C*，*D*，产生32比特的输出。四轮的逻辑函数定义如下：

*ft*(*B*,*C*,*D*) = (*B*AND *C*) or ((NOT *B*) AND *D*) ( 0 ≤*t*≤19)

*ft*(*B*,*C*,*D*) = *B*XOR *C*XOR *D*(20 ≤*t*≤39)

*ft*(*B*,*C*,*D*) = (*B*AND *C*) or (*B*AND *D*) or (*C*AND *D*) (40 ≤*t*≤59)

*ft*(*B*,*C*,*D*) = *B*XOR *C*XOR *D*(60 ≤*t*≤79).

SHA-1算法的四轮运算还要用到一系列的加法常量字*K*，定义如下（十六进制）：

*Kt*= 0x5A827999 (0 ≤t ≤19)

*Kt*= 0x6ED9EBA1 (20 ≤t ≤39)

*Kt*= 0x8F1BBCDC (40 ≤t ≤59)

*Kt*= 0xCA62C1D6 (60 ≤t ≤79).

计算消息摘要还需要80个32比特的缓冲区，记为*W*0, *W*1,..., *W*79。另外还需要一个32比特的TEMP缓冲区。

为了产生消息摘要，将经过了消息填充和长度填充后的一个512比特的数据块分组存放在16个32比特长的数据块*M*1, *M*2,..., *M*15中。依次对于*Mn*(0 ≤*n*≤15)进行四轮、八十步处理。具体的处理步骤如下：

①初始化*Wt*(0 ≤*t*≤79)。对于0≤*t*≤15，*W*0= *M*0, *W*1= *M*1, ... , *W*15= *M*15。对于16≤*t*≤79，令*Wt*= CLS1(*Wt*-3XOR *Wt*-8XOR *Wt*-14XOR *Wt*-16)。CLS*n*表示32比特的变量循环左移*n*位。例如，CLS1表示32比特的变量循环左移1位。

②对于0≤*t*≤79，执行下面的操作：

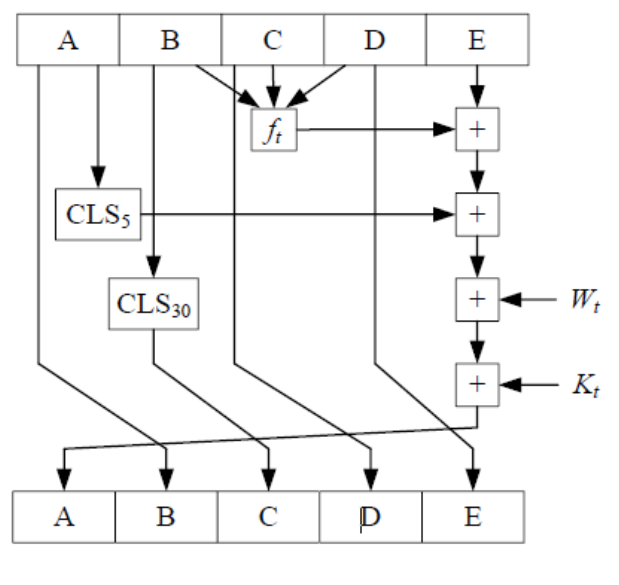
TEMP = CLS5(*A*) + *ft*(*B*,*C*,*D*) + *E*+ *Wt*+ *Kt*;

*E*= *D*; *D*= *C*; *C*= CLS30(*B*); *B*= *A*; *A*= TEMP。

下图示意出SHA-1执行一步迭代的过程。

每个512比特的分组处理完成后，执行：

H0= H0+ A, H1= H1+ B, H2= H2+ C, H3= H3+ D, H4= H4+ E。





(5) 输出摘要

在处理完所有的*k*+1个512比特的数据块后，最后一个数据块的消息摘要就是整个(*k*+1)\*512比特数块的消息摘要，以ABCDE的顺序标识。

SHA256，SHA384，SHA512采用相似的算法计算消息摘要。对消息进行填充的方法完全一样。

SHA-1算法函数的实现涉及较多的数组操作和位运算。例如，SHA-1算法第4步消息压缩中的循环左移和80个缓冲区*Wt*(0 ≤*t*≤79)的操作。

1. 循环左移的实现

#define SHA1CircularShift(value, bits) (((value) << (bits)) | ((value) >> (32 -(bits))))

2. 80个缓冲区*Wt*(0 ≤*t*≤79)的操作

for (t = 0; t < 16; t++) {

W[t] = ((unsigned) context->Message\_Block[t \* 4]) << 24;

W[t] |= ((unsigned) context->Message\_Block[t \* 4 + 1]) << 16;

W[t] |= ((unsigned) context->Message\_Block[t \* 4 + 2]) << 8;

W[t] |= ((unsigned) context->Message\_Block[t \* 4 +3]);

}

for(t = 16; t < 80; t++) {

W[t] = SHA1CircularShift(1,W[t-3] ^ W[t-8] ^ W[t-14] ^ W[t-16]);

}

3. 四轮八十步的操作TEMP = CLS5(*A*) + *ft*(*B*,*C*,*D*) + *E*+ *Wt*+ *Kt*( 0≤*t*≤79)的实现如下：

(1) 对于0 ≤*t*≤19的情况

temp = SHA1CircularShift(5,A) + ((B & C) | ((~B) & D)) + E + W[t] + K[0];

temp &= 0xFFFFFFFF;

(2)对于20 ≤*t*≤39的情况

temp = SHA1CircularShift(5,A) + (B ^ C ^ D) + E + W[t] + K[1];

temp &= 0xFFFFFFFF;

(3)对于40 ≤*t*≤59的情况

temp = SHA1CircularShift(5,A) +((B & C) | (B & D) | (C & D)) + E + W[t] + K[2];

temp &= 0xFFFFFFFF;

(4)对于60 ≤*t*≤79的情况

temp = SHA1CircularShift(5,A) + (B ^ C ^ D) + E + W[t] + K[3];

temp &=0xFFFFFFFF;

三、实验内容和任务

本实验要求学生在掌握哈希函数SHA-1算法的基础上，编程实现SHA-1算法,demo文件夹作为编程实现的参考。