《应用密码学》实验报告

课程：应用密码学　　实验名称：AES 基本变换

姓名：杨佳倪　　实验日期：2024.4.29

学号：2022132006　　实验报告日期：2024.4.29

班级：信安实验 221

教师评语：	成绩：
签名： 日期：	

一、实验名称

消息摘要函数 SHA-1 算法的实现

二、实验环境（详细说明运行的系统、平台及代码等）

1．编译环境：VC

2. 系统：Windows

三、实验目的

（1）加深对消息摘要函数 SHA-1 的理解；

（2）掌握消息摘要函数 SHA-1；

（3）提高编程实践能力。

四、实验内容、步骤及结果

1．实验内容

（1）按照标准 FIPS-180-2 中 SHA-1 算法，从文件或者屏幕中读取消息，然后对消息分组，并对最后一个分组进行填充，并对每组通过数据扩充算法扩充到 80 个字，然后执行 SHA-1 算法，并显示输出。

（2）完成填充过程，消息的长度在 1-200 个字符。

2．实验步骤

（1）输入待 Hash 消息字符串，编码方式为 ASCII 码。例如程序的默认输入为 FIPS-180-2 中示例的“abc”， 消息的长度在 1-200 个字符。

（2）按照 SHA-1 算法进行填充，然后 512 比特分组，分为多组，然后对每组消息进行处理，数据扩充到 80 个字。

（3）输出每一分组中的 W_0, W_1，W_{14}，W_{15}，W_{16}, W_{79} （十六进制）

（4）填充过程写成一个函数，数据扩充过程写成一个函数，数据扩充中循环移位也可以写成一个函数。

（5）认真填写实验报告，测试结果与标准对比。

3.实验结果

测试结果要求：

（1）输入为 ASCII 码，程序的默认输入为 FIPS-180-2 中示例的“abc”，输出按照标准如下：

W[0]=61626380

W[1]=00000000

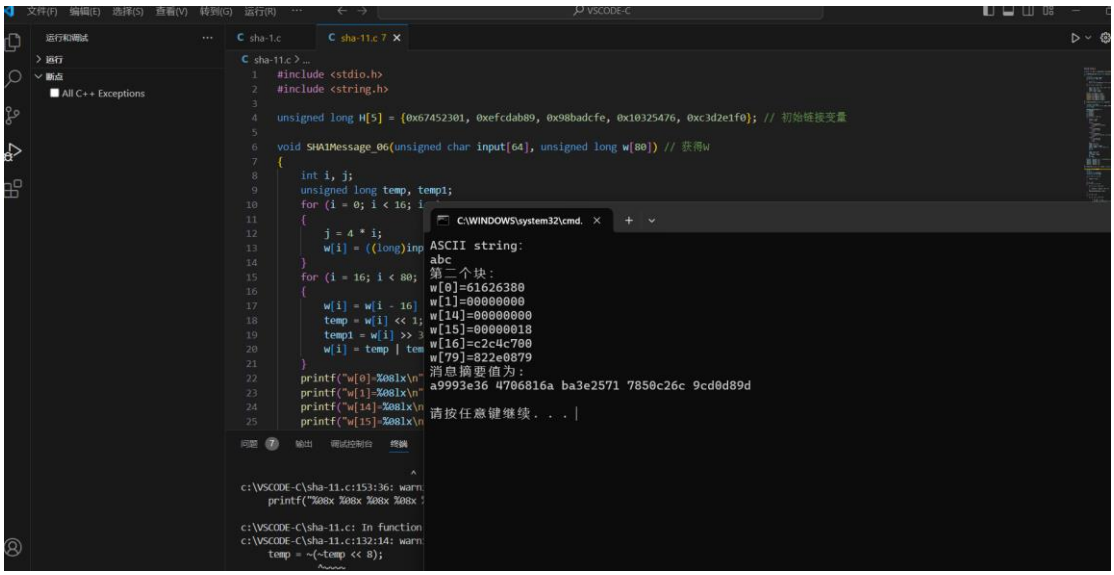
W[14]=00000000

W[15]=00000018

W[16]=c2c4c700

W[79]=822e0879

最终的消息摘要值为：a9993e36 4706816a ba3e2571 7850c26c 9cd0d89d



（2）输入为 ASCII 码,程序输入为 FIPS-180-2 实例中的标准输入：

“**abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq**”

的测试结果（**不带引号**），以及每一个 512bit 块的下述子块的输出结果：

W_0, W_1，W_{14}，W_{15}，W_{16}，W_{79}. 按照标准，输出的值如下：

第一个块：

W[14]=80000000

W[15]=00000000

W[16]=0a063a3e

W[79]=1ff69958

第二个块：

W[14]=00000000

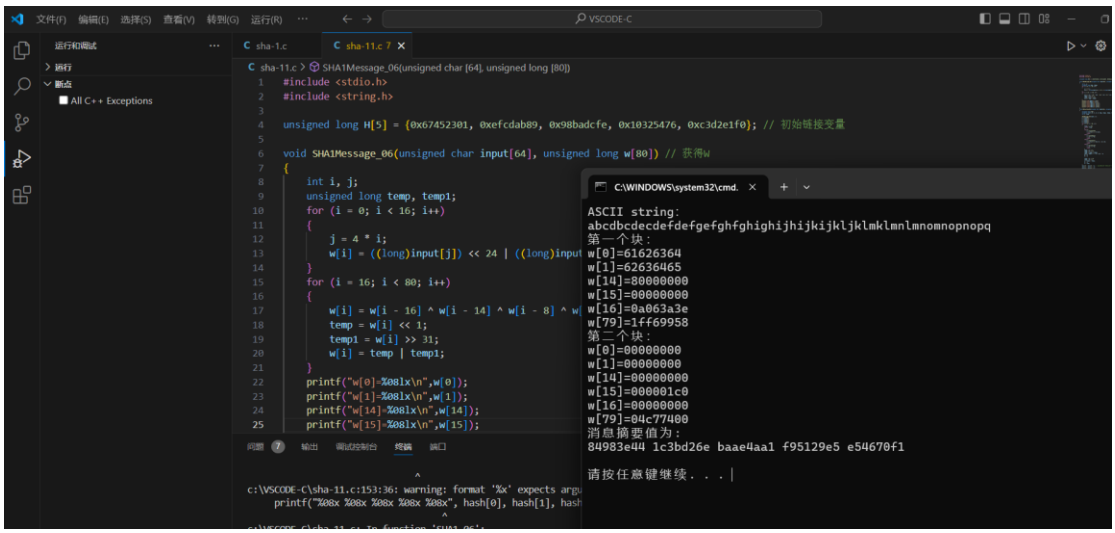
W[15]=000001c0

W[16]=00000000

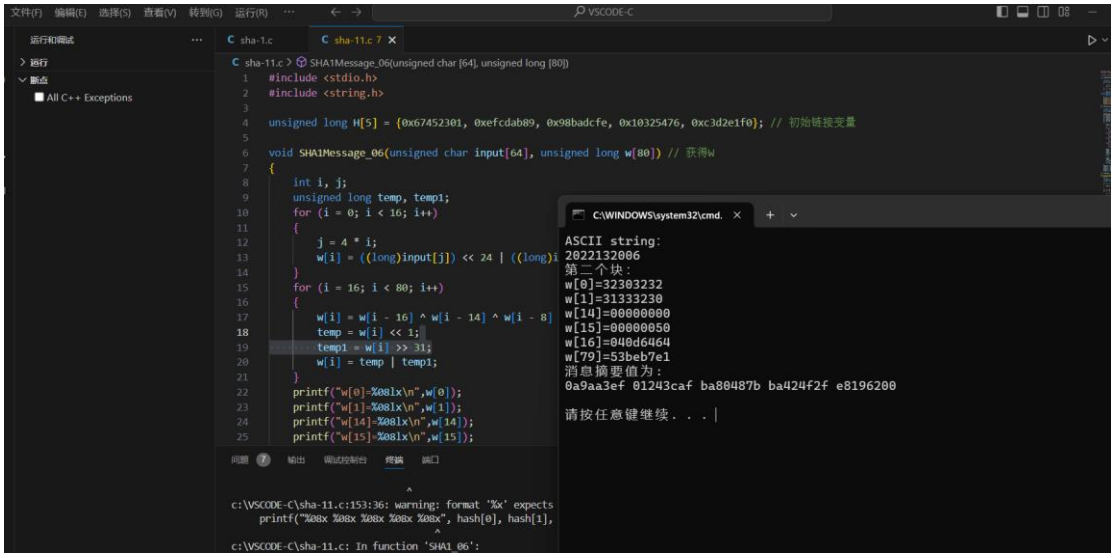
W[79]=04c77400

最终的消息摘要值为：

84983e44 1c3bd26e baae4aa1 f95129e5 e54670f1



（3）输入 ASCII 码为学号 2022132006 的测试结果，以及每一个块的下述子块的输出结果：W_0, W_1，W_{14} ，W_{15}. W16，W79 .



五、实验中的问题及心得

对于消息摘要函数 SHA-1 算法，在编写程序过程中，核心部分的程序要掌握算法逻辑过程，课后自己也敲了一遍代码，对消息摘要函数 SHA-1 算法有了更深的了解和掌握，也掌握了消息摘要函数 SHA-1 算法的规则。

附件：程序代码

```
#include <stdio.h>
#include <string.h>

unsigned long H[5] = {0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476, 0xc3d2e1f0}; // 初始链接变量
```

```
void SHA1Message_06(unsigned char input[64], unsigned long w[80]) // 获得W
{
    int i, j;
    unsigned long temp, temp1;
    for (i = 0; i < 16; i++)
    {
        j = 4 * i;
        w[i] = ((long)input[j]) << 24 | ((long)input[1 + j]) << 16 | ((long)input[2 + j]) << 8 | ((long)input[3 + j]) << 0;
    }
    for (i = 16; i < 80; i++)
```

```
    {
        w[i] = w[i - 16] ^ w[i - 14] ^ w[i - 8] ^ w[i - 3];
        temp = w[i] << 1;
        temp1 = w[i] >> 31;
        w[i] = temp | temp1;
    }
    printf("w[0]=%08lx\n",w[0]);
    printf("w[1]=%08lx\n",w[1]);
    printf("w[14]=%08lx\n",w[14]);
    printf("w[15]=%08lx\n",w[15]);
    printf("w[16]=%08lx\n",w[16]);
    printf("w[79]=%08lx\n",w[79]);
}
void SHA1_group_06(unsigned char input[64], unsigned long hash[5]) // 对 512 位的消息组进行加密
{
    unsigned long w[80];
    unsigned long A, B, C, D, E, temp, temp1, temp2, temp3, k, f;
    int i, flag;
```

```
    SHA1Message_06(input, w);
    A = hash[0];
    B = hash[1];
    C = hash[2];
    D = hash[3];
    E = hash[4];
    for (i = 0; i < 80; i++)
    {
        flag = i / 20;
        switch (flag)
        {
        case 0:
            k = 0x5a827999;
            f = (B & C) | (~B & D);
            break;
        case 1:
            k = 0x6ed9eba1;
            f = B ^ C ^ D;
            break;
        case 2:
            k = 0x8f1bbcdc;
            f = (B & C) | (B & D) | (C & D);
            break;
        case 3:
            k = 0xca62c1d6;
```

```
            f = B ^ C ^ D;
            break;
        }
        temp1 = A << 5;
        temp2 = A >> 27;
        temp3 = temp1 | temp2;
        temp = temp3 + f + E + w[i] + k;
        E = D;
        D = C;
```

```
        temp1 = B << 30;
        temp2 = B >> 2;
        C = temp1 | temp2;
        B = A;
        A = temp;
    }
    hash[0] = hash[0] + A; // 将获得的 160 位变量依旧放在 hash 数组中，可作为下一组明文的链接变量
    hash[1] = hash[1] + B;
    hash[2] = hash[2] + C;
    hash[3] = hash[3] + D;
    hash[4] = hash[4] + E;
}
void SHA1_06(unsigned char *input, unsigned long hash[5]) // SHA1 算法
{
    int n = strlen(input);
    int m;
    int i, j;
    unsigned char group[64];
    unsigned long long temp;
```

```
    for (i = 0; i < 5; i++)
    {
        hash[i] = H[i];
    }
```

```
    m = n / 64;
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < 64; j++)
        {
            group[j] = input[i * 64 + j];
        }
        SHA1_group_06(group, hash);
    }
```

```
    if (n % 64 >= 56)
    {
        for (j = 0; j < 64; j++)
        {
            if (m * 64 + j < n)
                group[j] = input[m * 64 + j];
            else if (m * 64 + j == n)
                group[j] = 0X80;
            else
                group[j] = 0;
        }
        printf("第一个块：\n");
        SHA1_group_06(group, hash);
        m++;
    }
```

```
    for (j = 0; j < 64; j++)
    {
        if (m * 64 + j < n)
            group[j] = input[m * 64 + j];
        else if (m * 64 + j == n)
            group[j] = 0X80;
        else if (j < 56)
            group[j] = 0;
        else
            break;
    }
    temp = ~(~temp << 8);
    n = n * 8;
    for (i = 0; i < 8; i++)
    {
        j = 8 * i;
        group[63 - i] = (char)((n & (temp << j)) >> j);
    }
    printf("第二个块：\n");
    SHA1_group_06(group, hash); // 加密最后一组明文，获得密文
}
```

```
int main()
{
    unsigned char input[300];
    unsigned long hash[5];
```

```
    printf("ASCII string: \n");
    scanf("%s", input);

    SHA1_06(input, hash);
    printf("消息摘要值为:\n");
    printf("%08x %08x %08x %08x %08x", hash[0], hash[1], hash[2], hash[3], hash[4]); // 获得 SHA1
加密
    printf("\n");

    return 0;
}
```