MD5算法实现实验报告

- 姓名：张淇
- 学号：17343153
- 邮箱：zhangq295@mail2.sysu.edu.cn

一、实验内容

完成一个 MD5 算法的程序设计和实现，包括：

- 算法原理概述
- 总体结构
- 模块分解
- 数据结构
- C 语言 (可选其它命令式语言) 源代码
- 编译运行结果

二、实验原理

1. 填充

- 在长度为 K bits 的原始消息数据尾部填充长度为 P bits 的标识100...0，1 ≤ P ≤ 512 (即至少要填充1个bit)，使得填充后的消息位数为：K + P ≡ 448 (mod 512).注意：当 K ≡ 448 (mod 512) 时，需要 P = 512.
- 再向上述填充好的消息尾部附加 K 值的低64位 (即 K mod 2^64)，最后得到一个长度位数为 K + P + 64 ≡ 0 (mod 512) 的消息。

2. 分块

- 把填充后的消息结果分割为 L 个 512-bit 分组：Y_0, Y_1, ..., Y_L-1。
- 分组结果也可表示成 N 个32-bit 字 M_0, M_1, ..., M_N-1，N = L*16。

3. 缓冲区初始化

- 初始化一个128-bit 的 MD 缓冲区，记为 CV_q，表示成4个32-bit 寄存器 (A, B, C, D)；CV_0 = IV。迭代在 MD 缓冲区进行，最后一步的128-bit 输出即为算法结果。
- 寄存器 (A, B, C, D) 置16进制初值作为初始向量 IV，并采用**小端存储 (little-endian)** 的存储结构：A = 0x67452301, B = 0xEFCDAB89, C = 0x98BADCFE, D = 0x10325476。（对应C语言中的0x67452301,0xefcdab89,0x98badcfe,0x10325476）

4. 循环压缩

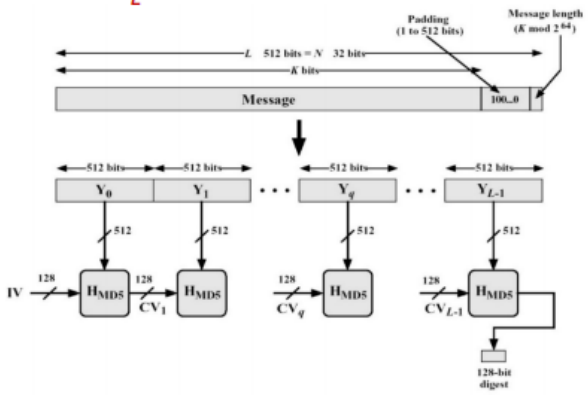
- 总控流程：

✧ 以512-bit 消息分组为单位，每一分组 Y_q (q = 0, 1, ..., L-1) 经过4个循环的压缩算法，表示为：

$$CV_0 = IV$$

$$CV_i = H_{MD5}(CV_{i-1}, Y_i)$$

✧ 输出结果：$MD = CV_L$.



- MD5压缩函数：

(1) H_MD5从 CV 输入128位，从消息分组输入512位，完成4轮循环后，输出128位，用于下一轮输入的 CV 值。

(2) 每轮循环分别固定不同的生成函数 F, G, H, I，结合指定的T表元素 T[] 和消息分组的不同部分 X[] 做16次迭代运算，生成下一轮循环的输入。

生成函数（轮函数）：

轮次	Function g	g(x,y,z)
1	F(x, y, z)	(((x) & (y))
2	G(x, y, z)	(((x) & (z))
3	H(x, y, z)	((x) ^ (y) ^ (z))
4	I(x, y, z)	((y) ^ ((x)

每轮循环中的一次迭代运算逻辑：

对 A 迭代：a := b + ((a + g(b, c, d) + X[k] + T[i]) <<<s)

缓冲区 (A, B, C, D) 作循环轮换：(B, C, D, A) := (A, B, C, D)

T表：

T[1.. 4] = { 0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee }
T[5.. 8] = { 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 }
T[9..12] = { 0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be }
T[13..16] = { 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 }
T[17..20] = { 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa }
T[21..24] = { 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 }
T[25..28] = { 0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed }
T[29..32] = { 0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a }
T[33..36] = { 0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c }
T[37..40] = { 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 }
T[41..44] = { 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 }
T[45..48] = { 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 }
T[49..52] = { 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 }
T[53..56] = { 0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 }

T[57..60] = { 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 }
T[61..64] = { 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 }

S表：

s[1..16] = { 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22 }
s[17..32] = { 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20 }
s[33..48] = { 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23 }
s[49..64] = { 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21 }

(3) 4轮循环共有64次迭代运算。

5. 得到结果

三、实验过程与结果

1. 数据结构

```
typedef unsigned char byte; // 存储字节，用于输入输出
typedef unsigned int bit32; // 存储32位的MD5中的ABCD寄存器
```

以及定义了相关的数组来存储补位、X表、T表、循环左移的s值：

```
// myMD5.hpp
// 用于补位字符串，最长512bits = 64bytes
static byte PADDING[64] = {
  0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

// 各轮循环中第 i 次迭代 (i = 1..16) 使用的 X[k] 的确定:
static bit32 X[64] = {
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
  1, 6, 11, 0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12,
  5, 8, 11, 14, 1, 4, 7, 10, 13, 0, 3, 6, 9, 12, 15, 2,
  0, 7, 14, 5, 12, 3, 10, 1, 8, 15, 6, 13, 4, 11, 2, 9
};

// 各次迭代运算 (1..64) 采用的 T 值的计算结果
static bit32 T[64] = {
  0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee, 0xf57c0faf, 0x4787c62a,
0xa8304613, 0xfd469501,
  0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be, 0x6b901122, 0xfd987193,
0xa679438e, 0x49b40821,
  0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa, 0xd62f105d, 0x02441453,
0xd8a1e681, 0xe7d3fbc8,
  0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed, 0xa9e3e905, 0xfcefa3f8,
0x676f02d9, 0x8d2a4c8a,
  0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c, 0xa4beea44, 0x4bdecfa9,
0xf6bb4b60, 0xbebfbc70,
```

```
  0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05, 0xd9d4d039, 0xe6db99e5,
0x1fa27cf8, 0xc4ac5665,
  0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039, 0x655b59c3, 0x8f0ccc92,
0xffeff47d, 0x85845dd1,
  0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1, 0xf7537e82, 0xbd3af235,
0x2ad7d2bb, 0xeb86d391
};

// 各次迭代运算 (1..64) 采用的左循环移位的 s 值:
static bit32 s[64] = {
  7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
  5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20,
  4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,
  6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21
};
```

2. 程序模块

程序基本上是按照老师课件（上述原理）来写的。

- 填充与分块：
  在程序中有两个地方涉及到，分别是`void calculate_digest(string msg)`和`void MD5(byte* input, size_t len)`,详情请看代码。
- 缓冲区初始化：

  ```
  // myMD5.hpp
  // ABCD寄存器（小端模式）
  static bit32 state[4] = {
      0x67452301,
      0xefcdab89,
      0x98badcfe,
      0x10325476
  };
  ```

- 循环压缩：
  这个过程对应的函数为：`void MD5(byte* input, size_t len)`和`void round(byte block[64])`详情请看代码。
  对于轮函数，在网上搜索资料的时候发现有博主将其写在宏定义中，可以提高效率，而且对于位操作比较难debug，所以在程序中借鉴了一下：

```
    // myMD5.hpp
    // 轮函数F
    #define F(x, y, z) (((x) & (y)) | ((~x) & (z)))

    // 轮函数G
    #define G(x, y, z) (((x) & (z)) | ((y) & (~z)))

    // 轮函数H
    #define H(x, y, z) ((x) ^ (y) ^ (z))

    // 轮函数I
    #define I(x, y, z) ((y) ^ ((x) | (~z)))
```

```
    // 循环座椅n位
    #define SHIFT_LEFT(num, n) (((num) << (n)) | ((num) >> (32-(n))))

    // 每轮循环中F函数的迭代运算逻辑
    #define FF(a, b, c, d, x, s, ac) { \
    (a) += F ((b), (c), (d)) + (x) + ac; \
    (a) = SHIFT_LEFT ((a), (s)); \
    (a) += (b); \
    }

    // 每轮循环中G函数的迭代运算逻辑
    #define GG(a, b, c, d, x, s, ac) { \
    (a) += G ((b), (c), (d)) + (x) + ac; \
    (a) = SHIFT_LEFT ((a), (s)); \
    (a) += (b); \
    }

    // 每轮循环中H函数的迭代运算逻辑
    #define HH(a, b, c, d, x, s, ac) { \
    (a) += H ((b), (c), (d)) + (x) + ac; \
    (a) = SHIFT_LEFT ((a), (s)); \
    (a) += (b); \
    }

    // 每轮循环中I函数的迭代运算逻辑
    #define II(a, b, c, d, x, s, ac) { \
    (a) += I ((b), (c), (d)) + (x) + ac; \
    (a) = SHIFT_LEFT ((a), (s)); \
    (a) += (b); \
    }
```

3. 得到结果

经过上述操作得到的结果存储在下面的数组中：

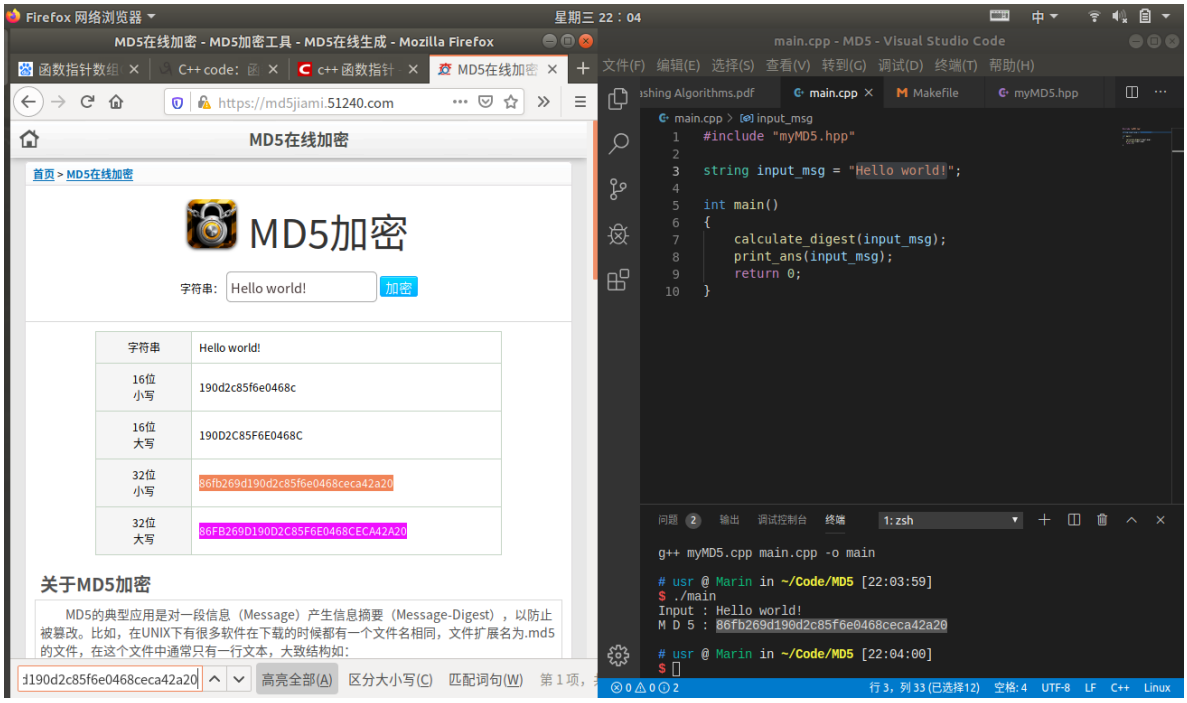
```
// myMD5.hpp
// 信息摘要（输出结果）
static byte digest[16];
```

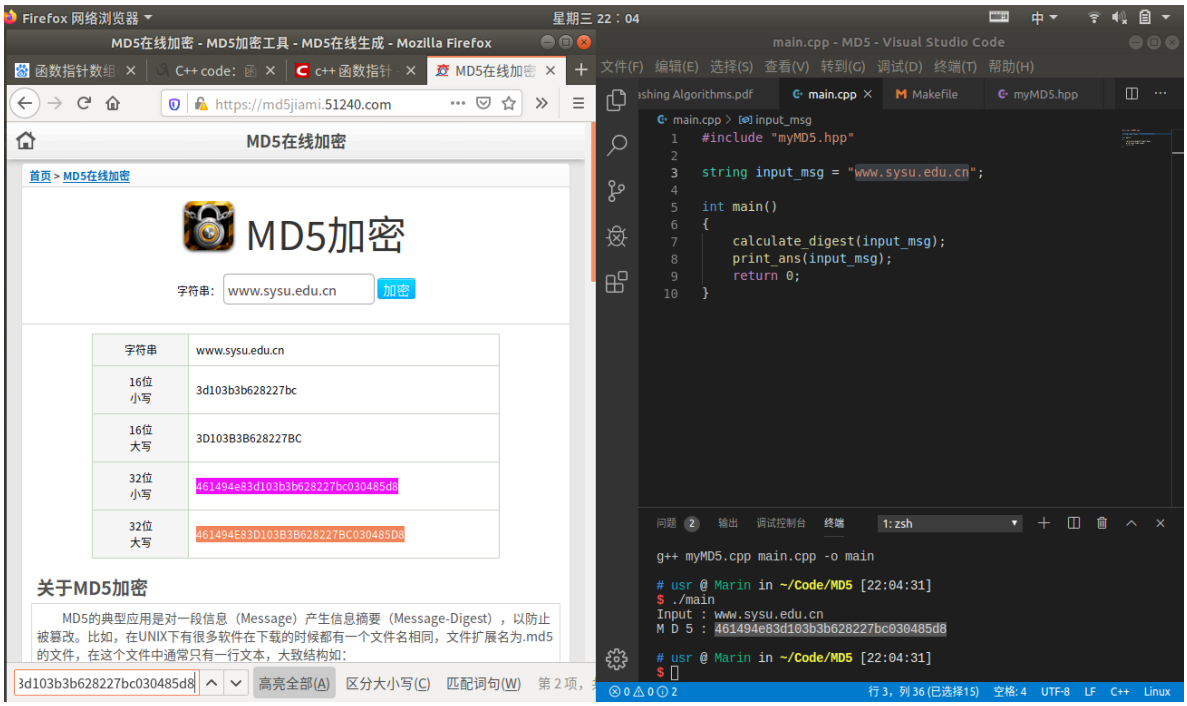
在结果使用的是数据类型是`byte`，不适合用来打印输出，所以需要将其转化为十六进制字符，才能进行打印输出，所以构建了下图的数组，相应函数为`void print_ans(string msg)`。

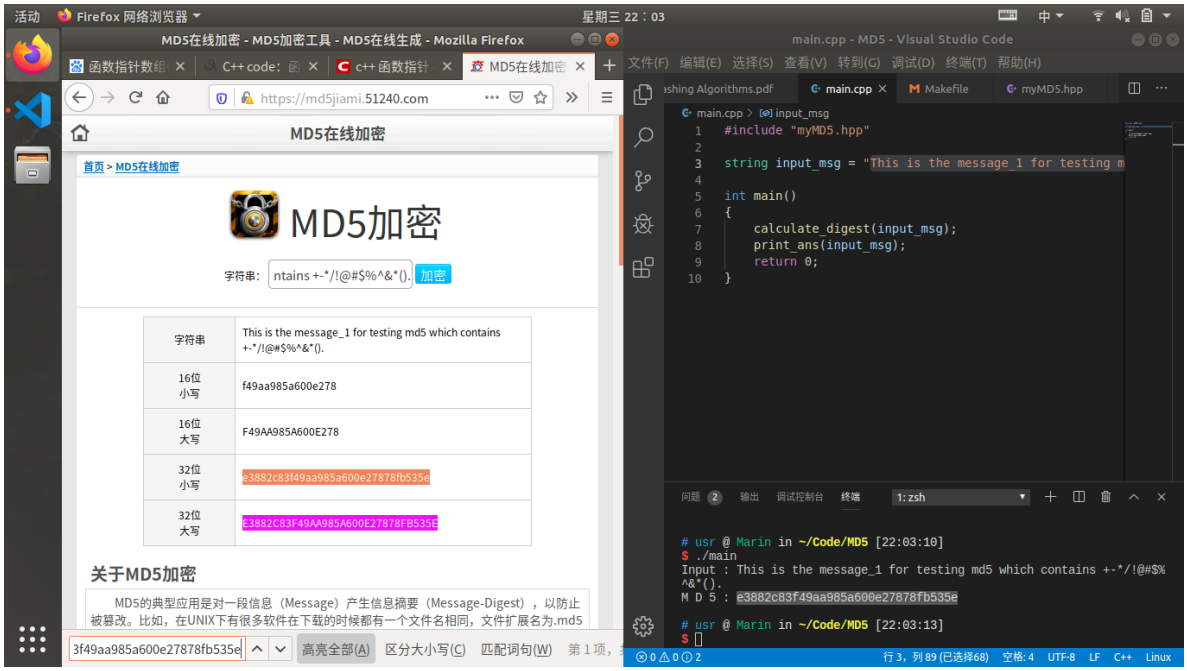
```
// myMD5.hpp
// 十六进制下的字符，用于打印
static char PRINT_CHAR[16] = {
  '0', '1', '2', '3',
  '4', '5', '6', '7',
  '8', '9', 'a', 'b',
  'c', 'd', 'e', 'f'
};
```

4. 实验结果截图

为了保证实验结果的准确性，我将同样的字符串输入到我的程序和一个在线生成字符串MD5码的网站，并对两个结果进行比较：







由上图可知：结果均一致（网页中的高亮是因为我将程序得到的结果在网页中进行“搜索”，其匹配的结果高亮显示），可以认为本程序生成的MD5码正确。